Computer Communications 128 (2018) 106–125

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Software-defined Radios: Architecture, state-of-the-art, and challenges

Rami Akeela, Behnam Dezfouli*

Internet of Things Research Lab, Department of Computer Engineering, Santa Clara University, USA

ARTICLE INFO

Keywords:
SDR
Wireless communication
Programmability
Co-design
LTE
WiFi
IoT

ABSTRACT

Software-defined Radio (SDR) is a programmable transceiver with the capability of operating various wireless communication protocols without the need to change or update the hardware. Progress in the SDR field has led to the escalation of protocol development and a wide spectrum of applications, with a greater emphasis on programmability, flexibility, portability, and energy efficiency in cellular, WiFi, and M2M communication. Consequently, SDR has earned a lot of attention and is of great significance to both academia and industry. SDR designers intend to simplify the realization of communication protocols while enabling researchers to experiment with prototypes on deployed networks. This paper is a survey of the state-of-the-art SDR platforms in the context of wireless communication protocols. We offer an overview of SDR architecture and its basic components, and then discuss the significant design trends and development tools. In addition, we highlight key contrasts between SDR architectures with regards to energy, computing power, and area, based on a set of metrics. We also review existing SDR platforms and present an analytical comparison as a guide to developers. Finally, we recognize a few of the related research topics and summarize potential solutions.

1. Introduction

Advances in wireless technologies have altered consumers' communication habits. Wireless technologies are an essential part of users' daily lives, and their impact will become even greater in the future. In a technical report, the World Wireless Research Forum (WWRF) has predicted that for 7 billion people, 7 trillion wireless devices will be deployed by 2020 [1]. When these devices are connected to the Internet to form an Internet of Things (IoT) network, the first challenge is to adjust the basic connectivity and networking layers to handle the large number of end points. There is an increasing number of wireless protocols that have been developed, such as ZigBee, Bluetooth Low Energy (BLE), Long Term Evolution (LTE), and new WiFi protocols, that have been developed to meet the demanding requirements of various domains such as 5G, IoT, and cyber-physical systems [2–4]. Wireless standards, in general, are adapting quickly in order to accommodate different user needs and hardware specifications [5,6]. To meet these specifications, a transceiver needs to be designed with the ability to handle several protocols, including the existing ones and those being developed. In order to accomplish this task, one needs to recognize the protocols' need for a *flexible, re-configurable*, and *programmable* framework.

Both consumer enterprise and military frameworks have a need for programmable platforms. Due to the rapid and consistent advancement of wireless protocols, programmability is of central significance to designers in the industry. Hardware needs to be able to keep up with both the evolution of technology and the changing user demands. For example, the authors in [7] proposed a platform called OpenRadio for programming both Physical (PHY) and Medium Access Control (MAC) layers while offering a high level of abstraction. Rather than including yet another piece of equipment to deal with a new standard or recurrence band, the equipment of a formerly introduced platform is able to adjust to the features of another standard. In a military scenario, for example, the needs of these platforms can change in light of the highly unpredictable conditions that arise during a mission. While these needs might not have been envisioned when designed initially, they led to the development and utilization of new protocols.

Software-defined Radio (SDR) is a technology for radio communication. This technology is based on software-defined wireless protocols, as opposed to hardware-based solutions. This translates to supporting various features and functionalities, such as updating and upgrading through reprogramming, without the need to replace the hardware on which they are implemented. This opens the door to the possibility of realizing multi-band and multi-functional wireless devices.

The driving factors for the high demand of SDR include network interoperability, readiness to adapt to future updates and new protocols, and most importantly, lower hardware and development costs. In

* Corresponding author.
E-mail addresses: rakeela@scu.edu (R. Akeela), bdezfouli@scu.edu (B. Dezfouli).

https://doi.org/10.1016/j.comcom.2018.07.012
Received 24 March 2018; Received in revised form 27 June 2018; Accepted 5 July 2018

a report [8], the SDR market is projected to be worth more than $29 billion by the year 2021. Global Industry Analysts, Inc. [9] highlights some of the market trends for SDR as follows: (i) increasing interest from the military sector in building communication systems and large-scale deployment in developing countries, (ii) growing demand for public safety and disaster preparedness applications, and (iii) building virtualized base stations (BSs). SDRs are also ideal for developing future space communications [10–12], Global Navigation Satellite System (GNSS) sensors [13], Vehicle-to-Vehicle (V2V) communication [14–16], and IoT applications [17,18], where relatively small and low-power SDRs can be utilized.

The SDR industry flourished due to the Joint Tactical Radio System (JTRS) program, which was responsible for producing SDRs for the military. In turn, this led to the creation of an entire world of new technologies, Software Communications Architecture (SCA), and Electronic Design Automation (EDA) tools that facilitate the development of SDRs [19]. The newly abundant resources made it relatively feasible to fuel the effort to develop more SDRs, not only for the military, but also for civil applications. The first commercial SDR, named Anywave [20], was a dual-mode base station that supported both Global System for Mobile communication (GSM) and Code Division Multiple access (CDMA) concurrently and ran on GPPs. Another technological advancement with a huge impact on the SDR industry was the development and release of Radio Frequency Integrated Circuit (RFIC), which supports most frequency bands in the MHz to GHz range.

Researchers have been studying SDRs for several years and are striving to find better means of implementing them in order to optimize their processing and energy efficiency. SDRs are implemented using various types of hardware platforms, such as General Purpose Processors (GPPs), Graphics Processing Units (GPUs), Digital Signal Processors (DSPs), and Field Programmable Gate Arrays (FPGAs). Each of these platforms is associated with its own set of challenges. Some of these challenges are: utilizing the computational power of the selected hardware platform, keeping the power consumption at a minimum, ease of design process, and cost of tools and equipment. Both the research community and industry have developed SDRs that are based on the aforementioned hardware platforms. A few examples include USRP [21], Sora [22], Atomix [23], Airblue [24], and Wireless Open Access Research Platform (WARP) [25]. Each SDR is unique with regards to the design methodology, development tools, performance, and end application.

In this paper, we first present an overview of the SDR architecture, as well as the analog and digital divides of the system and interconnection of components. Then, we introduce the criteria that defines how the different hardware platforms are classified. We thoroughly examine the architecture and design approaches employed by these hardware platforms and present their strengths and weaknesses in the context of SDR implementation. Furthermore, we provide an analytical comparison of hardware platforms as a guide for design decision making. Moreover, we discuss the use of development tools and present a summary to give a streamlined explanation of their functionalities and the platforms they support. Afterwards, we review the SDR platforms developed by both industry and academia, analyze them, and compare them using the criteria that was discussed earlier. Finally, we identify the current challenges and open research topics that are related to future SDR development.

This paper is organized as follows: Section 2 provides a description of SDR architecture and the classification process that is used to summarize the various design approaches adopted. Section 3 provides a comprehensive study of all the hardware platforms and associated design methodologies that are used to build SDR platforms. Section 4 lists some of the corresponding development tools and platforms. Section 5 presents an analysis and comparison of the commercially and academically developed SDR platforms. Section 6 highlights research questions and future trends. Section 7 presents an analysis of the existing literature on SDR surveys. We conclude the paper in Section 8. A list of key abbreviations used in this paper can be found in Table 1.

Table 1
Key abbreviations.

ADC	Analog-to-Digital Converter
ASIC	Application-Specific Integrated Circuit
BS	Base Station
CUDA	Compute Unified Device Architecture
DAC	Digital-to-Analog Converter
DSP	Digital Signal Processor
FFT	Fast Fourier Transform
FLOPS	Floating Point Operations Per Second
FPGA	Field Programmable Gate Array
GPP	General Purpose Processor
GPU	Graphics Processing Unit
HLS	High Level Synthesis
NFV	Network Function Virtualization
RTL	Register-Transfer Level
SDR	Software-Defined Radio
SDN	Software-Defined Network
SNR	Signal-to-Noise Ratio
SoC	System on Chip
USRP	Universal Software Radio Peripheral

2. Concepts and architecture

In this section, we examine the general architecture of SDRs, their main components, and their processing requirements. As explained in the previous section, SDRs play a vital role in wireless standard development due to their flexibility and ease of programmability. This is due to the fact that most digital signal processing and digital front end, which includes channel selection, modulation and demodulation, takes place in the digital domain. This is usually performed in the software running on processors, such as GPPs and DSPs. However, it can also run on programmable hardware, i.e., FPGAs.

In general, from the transmitter's point of view, first a baseband waveform needs to be produced and then an Intermediate Frequency (IF) waveform. A RF waveform will be generated and then sent through the antenna. From the receiver's point of view, this RF signal is sampled, demodulated, and then decoded. To provide more details to the process, we study the receiving end of the system as follows.

The RF signal from the antenna is amplified with a tuned RF stage, which amplifies a range of the frequency band. This amplified RF signal is then converted to an analog IF signal. The Analog-to-Digital Converter (ADC) digitizes this IF signal into digital samples. Then, it is fed into a mixer stage. The mixer, which is an electrical circuit that takes in two signals and yields a new frequency, has another input coming from a local oscillator with a frequency that is set by the tuning control. The mixer then translates the input signal to a baseband. The next stage is a Finite Impulse Response (FIR) filter that permits only one signal. The FIR is a combination of multiply-add units and shift registers. This filter limits the signal bandwidth and acts as a decimating low-pass filter. The digital down-converter includes a large number of multipliers, adders, and shift-registers in the hardware in order to accomplish the aforementioned tasks. Next, the signal processing stage performs tasks such as demodulation and decoding. This stage is typically handled by a dedicated hardware like an Application Specific Integrated Circuit (ASIC) or other programmable alternatives like FPGA or DSP [26].

As shown in Fig. 1(a) and (b), at a high level, a typical SDR transceiver consists of the following components: Signal Processing, Digital Front End, Analog RF Front End, and an antenna.

2.1. Antenna

SDR platforms usually employ several antennas to cover a wide range of frequency bands [27]. Antennas are often referred to as

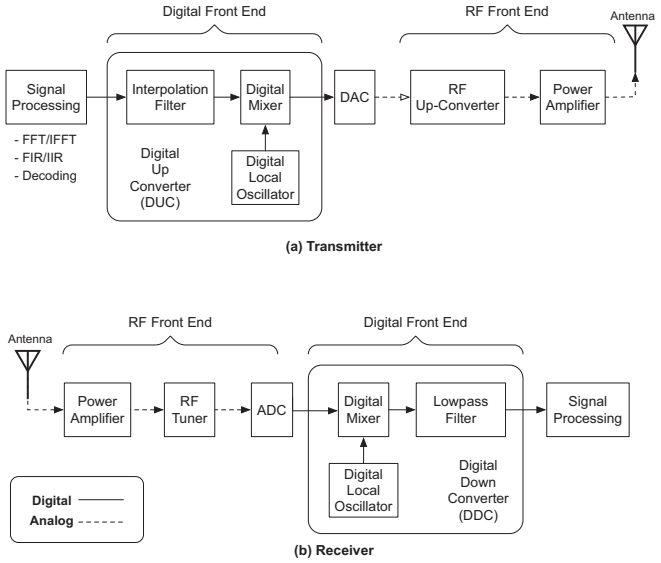

Fig. 1. SDR architecture. Sub-figure (a) shows SDR from a receiver's point of view, and sub-figure (b) shows SDR from a transmitter's point of view.

"intelligent" or "smart" due to their ability to select a frequency band and adapt with mobile tracking or interference cancellation [26,28]. In the case of SDRs, an antenna usually needs to meet a certain list of requirements such as self-adaptation (i.e., flexibility to tuning to several bands), self-alignment (i.e., beamforming capability), and self-healing (i.e., interference rejection) [28].

2.2. RF Front End

This is a RF circuitry where the main function is to transmit and receive the signal at various operating frequencies. Its other function is to change the signal to/from the Intermediate Frequency (IF). The process of operation is divided into two, depending on the direction of the signal (i.e., Tx or Rx mode):

- In the transmission path, digital samples are converted into an analog signal by the Digital-to-Analog Converter (DAC), which in turn feeds the RF Front End. This analog signal is mixed with a preset RF frequency, modulated, and then transmitted.
- In the receiving path, the antenna captures the RF signal. The antenna input is connected to the RF Front End using a matching circuitry to guarantee an optimal signal power transfer. Then, it passes through a Low Noise Amplifier (LNA), which resides in a close proximity to the antenna, in order to amplify weak signals and minimize the noise level. This amplified signal, in conjunction with a signal from the Local Oscillator (LO), is fed into the mixer in order to down-convert it to the IF [29].

2.3. Analog-to-Digital and Digital-to-Analog Conversion

The DAC, as mentioned in the previous section, is responsible for producing the analog signal that will be transmitted from the digital samples. The ADC resides on the receiver side and is an essential component in radio receivers. The ADC is responsible for converting continuous-time signals to discrete-time, binary-coded signals. ADC performance can be described by various parameters [30,31] including: (i) Signal-to-Noise Ratio (SNR): the ratio of signal power to noise power in the output, (ii) resolution: number of bits per sample, (iii) Spurious-free Dynamic Range (SFDR): the strength ratio of the carrier signal to the next strongest noise component or spur, and (iv) power dissipation. Advances in SDR development have provided momentum for ADC performance improvements. For example, since ADC's power consumption affects the lifetime of battery-powered SDRs, more energy efficient ADCs have been developed [32].

2.4. Digital Front End

The Digital Front End performs two functions [31]:

- Sample Rate Conversion (SRC), which is a functionality that converts the sampling from one rate to another. This is necessary since the two communication parties must be synchronized.
- Channelization, which includes up/down conversion in the transmitter and receiver side, respectively. It also includes channel filtering, where channels that are divided by frequency are extracted. Some examples include interpolation and low-pass filters, as depicted in Fig. 1.

In a SDR transceiver, the following tasks are executed in the digital front end:

- On the transmitting side (Fig. 1(a)), the Digital Up Converter (DUC) translates the baseband signal to IF. The DAC, which is connected to the DUC, then converts the digital IF samples into an analog IF signal. Afterwards, the RF up-converter converts the analog IF signal to RF frequencies.
- On the receiving side (Fig. 1(b)), the ADC converts the IF signal into digital samples. These samples are subsequently fed into the next block, which is the Digital Down Converter (DDC). The DDC includes a digital mixer and a numerically-controlled oscillator. The DDC extracts the baseband digital signal from the ADC. After it is processed by the Digital Front End, this digital baseband signal is forwarded to a high-speed digital signal processing block [33].

A new alternative to the classical approach is a concept known as Direct RF Sampling (DRFS). In DRFS, the RF sampling ADC replaces the analog processing blocks, such as the mixer, local oscillator and filters, and moves the processing to the digital domain. This significantly improves the design for the receiver. Here, the signal is converted by the ADC and handed over to the signal processing block in order to extract the data [34]. Based on the sub-sampling or bandpass sampling theory, which uses the alias of the signal in order to sample, it requires a much lower sampling rate. A bandpass filter is placed in front of the ADC to avoid sensitivity loss. The advantages of DRFS are supporting a very wide bandwidth and offering higher power efficiency [35].

2.5. Signal processing

Signal processing operations, such as encoding/decoding, interleaving/deinterleaving, modulation/demodulation, and scrambling/descrambling are performed in this block. Encoding for the channel serves as an error correcting code. Specifically, the encoded signal includes redundancy that is utilized by the receiver's decoder to re-construct the original signal from the corrupted received signal. Examples of error correcting codes include Convolutional Codes, Turbo Codes, and Low Density Parity Check (LDPC) [36]. The decoder constitutes the most computationally intensive part of the Signal Processing block, due to data transfer and memory schemes [37]. The second part that is regarded as highly complex and expensive, in terms of area and power, is the Fast Fourier Transform (FFT) and Inverse FFT (IFFT), as part of the modulation phase [38].

The signal processing block is commonly referred to as the *baseband processing block*. When discussing SDRs, the baseband block is at the heart of the discussion, since it makes up the bulk of the digital domain of the implementation. This implementation runs on top of a hardware circuitry that is capable of processing signals efficiently. Some examples include ASICs, FPGAs, DSPs, GPPs, and GPUs. The second part of the implementation is the software, which provides the functionality and high-level abstractions needed to execute the signal processing operations. In the next section, we examine the aforementioned hardware platforms and analyze in detail the various design approaches.

3. Design approaches

In this section, we discuss the classification of the various SDR design methodologies of the baseband processing block, namely GPP, GPU, DSP, FPGA, and co-design based methodologies. In this classification, we analyze and compare SDR platforms based on a set of performance metrics in the following criteria:

-*Flexibility and reconfigurability*. The capability for the modulation and air-interface algorithms and protocols to evolve by merely loading new software onto the platform [12].
-*Adaptability*. The SDR platform can adjust its capabilities based on network dynamics and user demands.
-*Computational power*. The processing rate of the SDR platform, namely Giga Operations per Second (GOPS).
-*Energy efficiency*. The total power consumption (typically within a few hundreds milliwatts), especially for mobile and IoT deployments [39,40].
-*Cost*. The total cost of the SDR platform, including time to market, development, and hardware costs.

3.1. GPP-based

One of the first approaches to realizing SDR platforms is using a General Purpose Processor (GPP), or the commonly known generic computer microprocessors such as x86/64 and ARM architectures. Some examples of SDR platforms that utilize GPPs are Sora [22], KUAR [41], and USRP [21].

3.1.1. Definition and uses

A GPP is a digital circuit that is clock-driven and register-based. It is capable of processing different functions and operates on data streams represented in the binary system [42]. These GPPs can be used for several purposes, making them extremely useful for an unlimited number of applications. This eliminates the need for building application-specific circuits, reducing the overall cost of running applications. GPPs are generally a preferable hardware platform by researchers in academia due to their flexibility, abundance, and ease of programmability, which is one of the main requirements in SDR platforms [43]. In addition, researchers prefer GPPs, since they are more familiar with them and their software frameworks when compared to DSPs and FPGAs. From the performance point of view, GPPs are being enhanced rapidly. This advancement is not only credited to technological advances in terms of Complementary Metal Oxide Semiconductor (CMOS) technology [44] but also to the increase of the average number of instructions processed per clock cycle. The latter is achieved through different means, and in particular, utilizes parallelism within and between processors. This has led to the evolution of multi-core GPPs [45].

3.1.2. Adoption and GPUs

Architecturally, the instruction set of GPPs includes instructions for different operations such as Arithmetic and Logic Unit (ALU), data transfer, and I/O. A GPP processes these instructions in a sequential order. Because of sequential processing, GPPs are not convenient for high-throughput computing with real-time requirements (i.e., high throughput and low latency) [46]. For example, using GNU Radio [47] to implement IEEE 802.11 standard, which requires a 20 MHz sampling rate, would be challenging, since GNU Radio is restricted by the limited processing capabilities of GPPs. This leads to the GPP cores (of the PC attached) to reach saturation and to frames becoming corrupted and discarded. Moreover, wireless protocols require predictable performance in order to guarantee that they meet the timing constraints. However, conditional branch instructions in the GPP's instruction sets lead to out-of-order execution, which makes it unfeasible to achieve predictability.

To overcome the limitation of GPPs, researchers have proposed multiple solutions, one of which is the addition of co-processors, such as the Graphic Processing Unit (GPU) [48]. GPUs are processors specifically designed to handle graphics-related tasks, and they efficiently process large blocks of streaming data in parallel. SDR platforms that are comprised of both GPPs and GPUs are flexible and have a higher level of processing power. However, this results in a lower level of power efficiency (e.g., GPP's power efficiency is $\approx$ 9GFLOPS/W for single-precision, compared to 20GFLOPS/W for GPU [49]). GPUs act as co-processors to GPPs because a GPP is required to act as the control unit and transfer data from external memory. After a transfer is completed, the GPU executes signal processing algorithms.

While GPUs are typically used for processing graphics, they are also used for signal processing algorithms. Over the past few years, the theoretical peak performance for GPUs and GPPs for single and double precision processing has been growing [50]. For example, when comparing Intel Haswell's 900 GFLOPs [51] with NVIDIA GTX TITAN's 4500 GFLOPS [52] for single precision, it is apparent that GPUs have a computational power that far exceeds their GPP counterparts [50]. Their multi-core architectures and parallel processors are the main attractive features, in addition to their relatively reasonable prices and small, credit card-like size. These features make them good candidates for co-processors in GPP-based SDRs, where they can play a vital role in accelerating computing-intensive blocks [53]. Another advantage is their power efficiency, which keeps improving with every new model (e.g., it went from 0.5 to 20GFLOPS/W for single-precision) [49]. To take full advantage of GPUs, it is a condition that the algorithms conform to their architecture. From an architectural perspective, GPUs have a number of advantages that make them preferable solutions to applications like video processing. In particular, GPUs employ a concept called Single Program Multiple Data (SPMD) that allows multiple instruction streams to execute the same program. In addition, due to their multi-threading scheme, data load instructions are more efficient. GPUs also present a high computational density, where the cache to ALU ratio is low [54].

In Table 2, the authors of [53] confirmed that the signal detection algorithm, which includes intensive FFT computations, shows a faster parallel processing in the case of GPU over GPP, while operating in real-time (by orders of magnitude). This is due to the availability of the compute unified Fast Fourier Transform (cuFFT) library which was developed for NVIDIA GPUs for more efficient FFT processing [55]. With regards to the architectural advantage of GPUs, several hundred CUDA cores can perform a single operation at the same time, as opposed to a few cores in the case of multi-core GPPs.

An example of using GPUs alongside GPPs to build SDR platforms is found in the work in [56], where the authors built a framework on a desktop PC in addition to using a GPU to implement an FM receiver. Additionally, the authors in [53] studied real-time signal detection using an SDR platform composed of a laptop computer and an NVIDIA Quadro M4000M [52]. Examples of GPUs available in the market can be found in Table 3. In this table, we show two examples of high performing GPUs (> 5500 GFLOPS), suitable for SDRs with strict timing and performance requirements. We also show two more examples of less powerful and less expensive GPUs, suitable for prototyping SDRs in academic environments.

Table 2
Performance of signal detection algorithm on GPP and GPU [53].

ADC Data length (ms)	Processing platform of signal detection algorithm		
	GPP Serial processing (ms)	GPP Parallel processing (ms)	GPU Parallel processing (ms)
1	13.487	1.254	0.278
10	135.852	12.842	2.846
100	1384.237	131.026	29.358
1000	13946.218	1324.346	321.254

Table 3
Comparison of GPUs.

	NVIDIA GeForce GTX 980 Ti [52]	AMD Radeon R9 390X [57]	NVIDIA GeForce GTX 680 [52]	AMD Radeon RX 560 [57]
GFLOPS	5632	5913	3090	2611
Power onsumption (W)	250	363	356	180
Frequency (MHz)	1000	1050	1006	1175
Cost (USD)	870	520	300	150

3.1.3. Shortcomings

State-of-the-art GPP and GPU-based platforms, such as Sora [22] and USRP [21], utilize desktop computers to realize the systems. However, these platforms consume a significant amount of power for a performance goal, and their form factor (i.e., shape and physical size) is large, making real-world deployment a challenging task. It is worth noting that GPPs and GPUs alike present scaling limitations while meeting Koomey's Law. This law states that the energy efficiency of computers doubles roughly every 18 months [58]. This limitation calls for alternatives that provide higher computing power while keeping the energy efficiency the same. One alternative is the *hybrid* or *co-design* approach, where software and hardware implementations are combined. This will be discussed in more details in Section 3.4.

When both GPPs and GPUs are used for a SDR design, data transfer operations between the GPP and GPU can create bottlenecks and cause performance loss, especially when trying to meet real-time requirements [59]. However, there are continuous efforts to reduce or eliminate the time overhead of data transfers by introducing multi-stream scheduling for pipelining of the memory copy tasks. This would ensure that there are no stalls in the pipeline, which would enhance processing parallelism [60,61]. Finally, although the processing power of microprocessors is constantly being improved, the balance between sufficient computing power and meeting a specific goal for energy consumption and cost will remain a very difficult task, both in the present day and in the future. This is true especially with the growing need for more data to be processed and blocks that can handle data processing in parallel.

3.2. DSP-based

The DSP-based solution can be considered as a special case of GPP-based solutions, but due to its popularity and unique processing features, it deserves a separate discussion. An example of DSP-based SDR is the Atomix platform [23] which utilizes TI TMS320C6670 DSP [62].

3.2.1. Definition and uses

DSP is a particular type of microprocessor that is optimized to process digital signals [63]. To help understand how DSPs are distinguished from GPPs, we should first note that both are capable of implementing and processing complex arithmetic tasks [64]. Tasks like modulation/demodulation, filtering, and encoding/decoding are commonly and frequently used in applications that include speech recognition, image processing, and communication systems. DSPs, however, implement them more quickly and efficiently due to their architecture (e.g., RISC-like architecture, parallel processing), which is specifically optimized to handle arithmetic operations, especially multiplications. Since DSPs are capable of delivering high performance with lower power, they are better candidates for SDR deployment [65] compared to GPPs. Examples of DSPs that are specifically designed for SDR platforms are TI TMS320C6657 and TMS320C6655. These DSPs are both equipped with hardware accelerators for complex functions like the Viterbi and Turbo Decoders [66].

3.2.2. Adoption

As discussed in the previous section, GPPs provide an average performance for a wide range of applications. Needless to say, this performance level might be sufficient for research and academia, but if the system is to be deployed commercially, certain performance requirements must be met. To this end, compared to GPPs, DSPs are tailored for processing digital signals efficiently, utilizing features like combined multiply-accumulate operations (MAC units) and parallelism [67]. DSP manufacturers usually sell these products in two categories: optimized for performance and optimized for energy. Therefore, when used in SDRs, high performance and energy efficient products can be employed in BSs and edge devices, respectively.

In terms of the instruction set, DSPs can be categorized into two groups: (i) Single Instruction Multiple Data (SIMD) architecture, and (ii) Multiple Instruction Multiple Data (MIMD) architecture, as described by Michael J. Flynn in what is known as Flynn's Taxonomy [68,69]. This taxonomy is a method of classifying various architectures depending on the number of concurrent instructions and data streams, as follows:

– A SIMD-based DSP can execute an instruction on multiple data streams at the same time. This architecture can be very efficient in cases when there exists high data parallelism within the algorithm [70]. This indicates that there are similar operations that can be performed on different datasets at the same time. Examples of SIMD-based DSPs include the Cell processor presented in [71] which supports 256 GFLOPS. More examples of DSPs that are optimized for low power are NXP CoolFlux DSP [72] and Icera Livanto [73]. A SDR employing a SIMD DSP is the SODA architecture [74]. It has been a common practice to add more cores in order to achieve a better trade-off between performance and power. With each extra core utilizing Very Long Instruction Word (VLIW), a higher level of parallelism can be accomplished as well.
– On the other hand, MIMDs have the ability to operate on multiple data streams executing multiple instructions at any point in time. This is essentially an extension of the SIMD architecture, where different instructions or programs run on multiple cores concurrently. This is especially important and useful in cases where parallelism is not uniform across different blocks. However, the MIMD architecture allows for parallel execution, leading to speed improvements. Examples of MIMD-based DSPs include Texas Instruments SMJ320C80 and SM320C80 DSPs with 100 MFLOPS [66].

Since DSPs are customized to meet certain signal processing-related needs, it is crucial to clarify these customizations in order to understand how DSPs stand out and how they are successful at not only meeting the requirements but also in how they are becoming a vital player in the wireless communication field. These customizations, which are mostly architecture-related, are as follows.

In [75], the authors discuss the energy efficiency of DSPs. In general, DSPs consume more power than ASICs, however, there are DSPs that are optimized for low power wireless implementations, such as TI C674x DSP [66]. One of the methods to lower power consumption is to use multiple data memory buses (e.g., one for write, and two for reads). This paves the way for higher memory bandwidth and allows for multiple operand instructions, resulting in fewer cycles. As discussed above, VLIW architectures, along with specialized instructions, can provide a higher level of efficiency, which lowers energy consumption. These improvements can be seen in DSPs like TI TMS320C6x [66] and ADI TigerSHARC [76]. These techniques, coupled with proven power-saving techniques, such as clock gating and putting either parts of or the entire system in sleep mode, further reduce power consumption. Examples of DSPs available in the market can be found in Table 4. In this table, we present three examples of DSPs that do not include co-processors, and three DSP-based SoCs that, in addition to DSP cores, include extra soft cores as control processors.

Table 4
Comparison of DSPs and DSP-based SoCs.

	DSP only			SoC		
	TI C66x (TMS320C6652) [66]	CEVA (XC-4500) [77]	Analog Devices (ADSP-21369) [76]	TI Keystone II (66AK2G02) [66]	Analog Devices (ADSP-SC573) [76]	Qualcomm Snapdragon 820 (Hexagon 680) [78]
GFLOPS	9.6	40	2.4	28.8	5.4	No Floating Point
Memory (Kb)	1088	No Info	2000	1024	768	No Info
Frequency (MHz)	600	1300	400	600	450	2000
Cost (USD)	≈ 25	No Info	≈ 20	≈ 20	≈ 20	≈ 70
Soft Core	N/A	N/A	N/A	ARM Cortex-A15	ARM Cortex-A5	Qualcomm Kyro 385 (CPU) Adreno 530 (GPU)

3.2.3. Shortcomings

Despite the ubiquity of DSPs in SDR implementations for the past two decades [79], they do present some shortcomings. First, as more applications call for increasing parallelism and reconfigurability in order to handle computationally intensive tasks, DSPs can be insufficient. Second, programming DSPs to achieve higher levels of parallelism predictability can be challenging. This opened the door for parallel architectures, such as FPGAs, multi-core GPPs, or even a hybrid of both, to be adopted for SDRs. Third, power consumption of DSPs is generally higher than FPGAs due to them operating at high frequencies.

3.3. FPGA-based

Another approach towards realizing SDRs is to use a programmable hardware such as FPGAs. Example of FPGA-based SDR platforms are Airblue [24], Xilinx Zynq-based implementation of IEEE 802.11ah [80], and the work found in [81] that used the same FPGA board to implement a complete communication system with channel coding.

3.3.1. Definition and uses

An FPGA is an array of programmable logic blocks, such as general logic, memory, and multiplier blocks, that are surrounded by a routing fabric, which is also programmable [82]. This circuit has the capability of implementing any design or function and is able to be easily updated. Although FPGAs consume more power and occupy more area than ASICs, the programmability feature is the reason behind their increasing adoption in a wide range of applications. Furthermore, when the reconfiguration delay is in the order of milliseconds, the SDR can switch between different modes and protocols seamlessly [83]. Another major difference is that, ASIC fabrication is expensive (at least a few tens of thousands of dollars), and the process requires a few months. In contrast, FPGAs can be quickly reprogrammed, and their cost is within a few tens to a few thousands of dollars, at most. The low-end product cycle, along with attractive hardware processing advantages, like high-speed performance, low power consumption and portability, when compared to processors such as GPPs and DSPs, present FPGAs as contenders that offer the best of both worlds [82].

In a study by the authors in [84], they compared the performance of Xilinx FPGAs [85] against 16-core GPPs. The calculation of peak performance for GPPs was performed by multiplying the number of floating point function units on each core by the number of cores and by the clock frequency. For FPGAs, performance is calculated by picking a configuration, adding up the Lookup Tables (LUTs), flip-flops and DSP slices needed, and then multiplying them by the appropriate clock frequency. The authors calculated the theoretical peaks for 64-bit floating point arithmetic and showed that Xilinx Virtex-7 FPGA is about 4.2 times faster than a 16-core GPP. This is shown in Fig. 2. Even with a one-to-one adder/multiplier configuration, the V7-2000T achieved 345.35GFLOPS, which is better than a 16-core GPP. From Intel [51], Stratix 10 FPGAs can achieve a 10 Tera FLOPS peak floating point performance [86]. This is due to the fixed architecture of the GPP where not all functional units can be fully utilized, the inherent

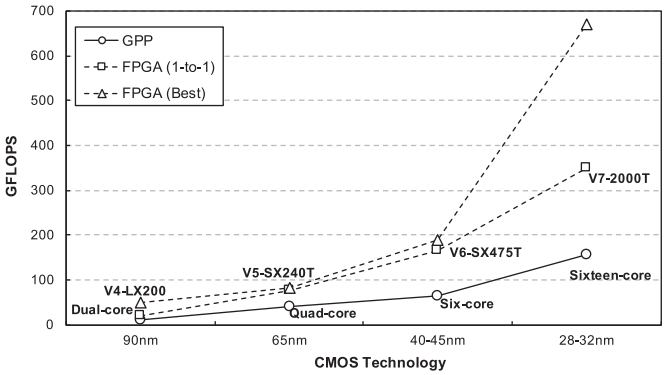

Fig. 2. Peak performance of GPPs versus FPGAs when performing 64-bit floating point operations [84]. It can be observed that FPGAs increased their floating point performance by an order of magnitude compared to GPPs.

parallelism of FPGAs, and their dynamic architecture. In addition, despite having lower clock frequencies (up to 300 MHz), FPGAs can achieve better performances due to their architectures which allows for higher levels of parallelism through custom design [87]. Furthermore, the authors in [88] compared the performance and power efficiency of FPGAs to that of GPPs and GPUs using double-precision floating point matrix-vector multiplication.The results show that FPGAs are capable of outperforming the other platforms while maintaining their flexibility. In addition, the authors in [54] thoroughly analyzed and compared FPGAs against GPUs via the implementations of various algorithms. The authors concluded that although both architectures support a high level of parallelism, which is crucial to signal processing applications, FPGAs offer a larger increase in parallelism, while GPUs have a fixed parallelism due to their data path and memory system.

3.3.2. Adoption

Over the past decade, FPGAs have significantly advanced and become more powerful computationally. They now exist in many different versions such as Xilinx Kintex UltraScale [85] and Intel Arria 10 [51] [89,90]. In addition, the availability of various toolsets gave FPGAs an advantage by making them more accessible. This is supported by the availability of compilers that have the capability of generating Register-transfer Level (RTL) code, such as Verilog and Very high speed integrated circuits Hardware Description Language (VHDL), that is needed to run on FPGAs, from high-level programming languages. This process is typically referred to as *High Level Synthesis* (HLS). Examples of such compilers include HDL Coder [91] for MATLAB code [92] and Xilinx HLS [93] or Altera Nios II C2H compiler [94] for C, C++, and SystemC. We will explain some of these tools in Section 4.

HLS allows software engineers to design and implement applications, such as SDRs, on FPGAs using a familiar programming language to code, namely C, C++, SystemC, and MATLAB, without the need to posses a prior rich knowledge about the target hardware architecture (refer to Section 4.1). These compilers can also be used to speed up or accelerate parts of the software code running on a GPP or DSP that are

Table 5
Comparison of FPGAs and FPGA-based SoCs.

	FPGA only			SoC		
	Xilinx Kintex-7 (XC7K70T) [85]	Intel Cyclone V GX (C5) [51]	Lattice ECP3-70 (LFE3-70EA) [97]	Xilinx Zynq-700 (Z-7020 XC7Z020) [85]	Intel Cyclone V SE SoC (A5) [51]	Microsemi SmartFuion2 (M2S090) [98]
Logic Cells (K)	65.6	77	67	85	85	86.31
Memory (Mb)	4.86	4.46	4.42	4.9	3.97	4.488
DSP Slices	240	150	128	220	87	84
Cost (USD)	130	185	80	110	110	155
Soft Core	N/A	N/A	N/A	Dual-core ARM Cortex-A9	Dual-core ARM Cortex-A9	ARM Cortex-M3

causing slowdowns or setbacks to the overall performance. This will be further discussed in Section 3.4. Further, FPGAs can achieve high performance while still consuming less energy than the previously discussed processors [95] (e.g., Intel Stratix 10 FPGA can achieve up to 100 GFLOPS/W [96], compared to 23 GFLOPS/W for NVIDIA GeForce GTX 980 Ti [52]). In addition, power dissipation can be further lowered through the implementation of several techniques at a system, device, and/or architecture level like clock gating and glitch reduction [83]. Table 5 presents a summary of the widely-used FPGA platforms.

3.3.3. Shortcomings

One of the challenges of using FPGAs, however, is the prior knowledge about the target hardware architecture and resources that a developer needs to possess in order to design an application efficiently for FPGAs. In the SDR domain, designing the platform has typically been the job of software engineers, and thus the process can be time-consuming and less trivial to incorporate this experience into hardware design. However, as it will be discussed in Section 4.1, the adoption of FPGA solutions can be made more feasible through HLS tools.

3.4. Hybrid design (a.k.a., co-design)

The fourth approach towards realizing SDRs is the hybrid approach, where both hardware and software-based techniques are combined into one platform. This is commonly referred to as the *co-design* or *hybrid* approach. Examples of SDRs that adopted the co-design approach include WARP [25] and CODIPHY [99].

3.4.1. Definition

Hardware/software co-design as a concept has been around for over a decade, and it has evolved at a faster rate in the past few years due to an increasing interest in solving integrated circuit design problems with a new and different approach. Even with GPPs becoming more powerful than ever, and with multi-core designs, it is clear that in order to achieve higher performance and create applications that demand real-time processing, designers had to shift attention to new design schemes that utilize hardware solutions, namely, FPGAs and ASICs [100,101]. Co-design indicates the use of hardware design methodology, represented by the FPGA fabric, and software methodology, represented by processors.

As more applications in the automotive, communication, and medical fields grow in complexity and size, it has become a common practice to design systems that integrate both software (like firmware and operating system) and hardware [102]. This has been made feasible in the recent years thanks to the advances in high-level synthesis and in developing tools that not only have the capability to produce efficient RTL from software codes, but also define the interface between both sides. The industry has identified the huge market for co-design and has provided various SoC boards that, in addition to the FPGA fabric, contain multiple processors. For example, the Xilinx Zynq board [85] includes an FPGA fabric as well as two ARM Cortex-A9 processors [103]. In addition to the aforementioned advantages, there are other reasons that make co-design even more interesting, such as faster time to market, lower power consumption (when optimized for this), flexibility, and higher processing speeds, as the hardware in these systems is typically used as an acceleration to software bottlenecks [104].

Adopting the co-design methodology in essence is a matter of partitioning the system into synthesizable hardware and executable software blocks. This process depends on a strict criteria that is developed by the designer [105,106]. The authors in [107] and [108] discuss their partitioning methodologies and present the process of making the proper architectural decisions. Common methods typically provide useful information to the designer in order to make the best decision of what to implement in hardware and what to keep in software. This information can include possible speedups, communication overheads, data dependencies, and the locality and regularity of computations [108]. Examples of SoC boards available in the market can be found in Table 5.

3.4.2. Adoption

As mentioned in Section 2, SDRs can be considered inherently hybrid or heterogeneous systems, implying the need for both hardware and software blocks. This is due to the fact that the control part is usually taken care of by a general processor. Other functions, such as signal processing, are taken care of by a specialized processor like DSPs, and they are sometimes accelerated using dedicated hardware like FPGAs [109]. This design approach fits well with SDRs and can be fully utilized to meet certain requirements that pertain to their attractive features. For example, accelerating portions of a block or moving it entirely to the FPGA fabric can help to push the processing time to the limit in order to achieve a real-time performance for real-life deployment. In addition, through careful implementation of RTL optimization techniques, the development of power efficient systems for mobile and IoT applications would be possible. On the other hand, running most of the MAC layer operations on a processor, or multi-processors, can be advantageous for easy reconfiguration. Therefore, different partitioning schemes can be adopted to meet the requirements of the application at hand.

It is worth noting that FPGA vendors, namely Intel [51] and Xilinx [85], are widening their product base with more SoCs and Multi-processor SoCs (MPSoCs) [110], due to the growing demand for such devices. An example of an SDR realized on an MPSoC is the work by [111]. In a white paper, National Instruments, the company that owns USRP [21], predicts that the future of SDRs is essentially a co-design implementation [112], especially due to the introduction of FPGAs that are equipped with a large number of DSP slices that are used for handling intensive signal processing tasks, as depicted in Fig. 3. This also can be seen from USRP E310 model, which incorporates a Xilinx Zynq SoC [85].

3.4.3. Shortcomings

A downside of adopting SoCs for co-design is that their prices are generally higher, compared to the previously mentioned design approaches, because they have multiple components on the same board,

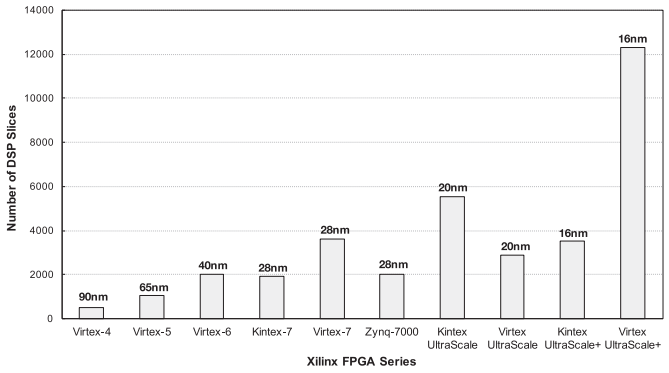

Fig. 3. Number of DSP Slices in Xilinx FPGAs [85]. The values on top of the bars refer to the CMOS technology used.

i.e., processor and FPGA fabric. Other factors that contribute to this are extra memory and sophisticated interfaces. Another challenge of co-design is the shared memory access, e.g., external DDR memory, between the processor and FPGA fabric. The study of [113] shows that the number of memory read and write operations performed by a GPP is higher than that of FPGAs. This is due to the fact that processors perform operation on registers, while FPGAs operate on buffers. Since memory accesses add up to the overall latency, this can cause a bottleneck to the overall performance. In addition, the authors have developed a methodology for predicting shared memory bandwidth by using a functionally-equivalent software. This enables the designers to be aware of any bottlenecks before implementing the entire co-design.

3.5. Comparison

When we covered different design methodologies and hardware platforms for a wide selection of SDRs, we intended to compare them analytically one-on-one using a cross-platform implementation of one of the wireless communication protocols, which means the software can be implemented on multiple hardware platforms. However, the literature showed a series of abstract comparisons using a set of benchmarks that targeted High Performance computing but not necessarily SDR applications. It is somewhat difficult to draw conclusions from these numbers alone, since a performance comparison in the SDR field requires real-world testing.

In Table 6, we provide a high-level comparison between three major design approaches as a guideline for designers towards choosing the method that best meets their application requirements.To help us compile the information mentioned in the table, we use prominent examples from corresponding vendors. These examples include Intel Core i5 [51] for GPPs, TI C66x [66] for DSPs, and Xilinx Virtex [85] for FPGAs. In this comparison, we used the criteria that was introduced at the beginning of Section 3. However, we do not make assumptions on what the best approach is and believe it is the developer's responsibility to make the best judgment depending on the application area. Please note that in this table we did not include GPUs, as they typically act as co-processors to GPPs, and their addition generally improves performance. We also did not include co-design, since it combines GPPs with FPGAs.

As Table 6 shows, while GPPs are easy to program and extremely flexible, they lack the power to meet specifications in real-time and are very inefficient in terms of power. To increase their performance, multiple cores with similar instruction sets are included in the same GPP platform to exploit parallelism and perform more operations per clock cycle. However, hardware replication (i.e., adding more cores to GPPs) may not necessarily translate to a higher performance. GPUs tackle this by offering the same control logic for several functional units. The sequential portion of the code runs on the GPP, which can be optimized on multi-core GPPs, while the computationally intensive portion runs on a several-hundred-core GPU, where the cores operate in parallel. Another example of a customized processor is the DSP. It performs significantly better than GPPs, while at the same time maintains the ease-of-use feature that GPPs possess, making them very attractive options. They are also more power efficient and better fit for signal processing applications. On the other hand, they are more expensive, which is the main trade-off. Finally, FPGAs combine the flexibility of processors and efficiency of hardware. FPGAs can achieve a high level of parallelism through dynamic reconfiguration, while yielding better power efficiency [49]. FPGAs are typically more suitable for fixed-point arithmetic, like in signal processing tasks, but in the recent years their floating-point performance has increased significantly [88,114]. However, the designers are expected to know a lot more about the hardware, which is sometimes a deterring feature.

In a comparative analysis by [115], the authors studied the performance and energy efficiency of GPUs and FPGAs using a number of benchmarks in terms of targeted applications, complexity, and data type. The authors concluded that GPUs perform better for streaming applications, while FPGAs are more suitable for applications that employ intensive FFT computations, due to their ability to handle non-sequential memory accesses in a faster and more energy efficient manner. Similarly, in [49], the authors review and report the sustainable performance and energy efficiency for different applications. One of their findings related to SDRs is that FPGAs should be used for signal processing without floating point, which confirms the aforementioned results. In addition, the authors in [116] report that GPUs are ten times faster than FPGAs with regards to FFT processing, while the authors in [88] demonstrate that the power efficiency of FPGAs is always better than GPUs for matrix operations. Finally, the authors in [117] compare GPPs, GPUs, and FPGAs through the implementation of LDPC decoders, and their results led to the conclusion that GPUs and FPGAs perform better than GPPs. It is obvious from the above studies that trade-offs are to be expected when a particular design methodology is adopted, hence careful analysis should be carried out beforehand. Other comparative studies include [118–120] with similar results and conclusions.

Table 6
Comparison of SDR design approaches.

	GPP [51]	DSP [66]	FPGA [85]
Computation	Fixed Arithmetic Engines	Fixed Arithmetic Engines	User Configurable Logic
Execution	Sequential	Partially Parallel	Highly Parallel
Throughput	Low	Medium	High
Data Rate	Low	Medium	High
Data Width	Limited by Bus Width	Limited by Bus Width	High
Programmability	Easy	Easy	Moderate
Complex Algorithms	Easy	Easy	Moderate
I/O	Dedicated Ports	Dedicated Ports	User Configurable Ports
Cost	Moderate	Low	Moderate
Power Efficiency	Low	Moderate	High
Form Factor	Large	Medium	Small

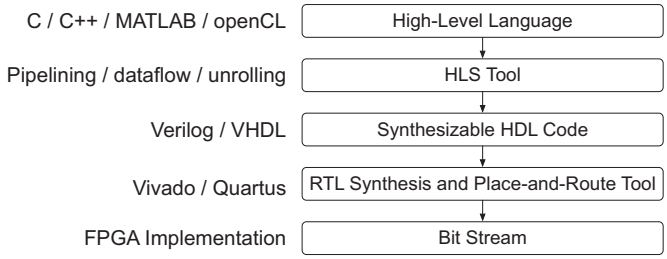

Fig. 4. HLS design flow commonly adopted by Xilinx [93], Intel [134], and MATLAB [91].

4. Development tools

As we mentioned in Section 3.3, HLS is an abstract method of designing hardware using a high-level programming language. Developers of FPGA and co-design based SDRs can benefit from HLS since it requires no prior experience with hardware design. Unlike the rest of the development tools, HLS tools share a common theme and offer similar features. Thus, we first discuss HLS in this section. Next, we review the common development tools that are typically used in the process of SDR design and implementation for different design approaches.

4.1. High Level Synthesis (HLS)

HLS has been a hot research topic for over a decade, with both academia and industry trying to make hardware design more accessible to every developer [121]. HLS is the process of converting an algorithmic specification of the design that is described by a high-level programming language to an RTL implementation. HLS provides a new level of design abstraction through exploring the micro-architecture and any hardware constraints. The resulting RTL is highly optimized, in terms of power, throughput and latency, and it is reasonably comparable to a hand-tuned code. Fig. 4 depicts this process. The major difference between RTL and C is the absence of the timing description in the high-level model, which is merely a behavioral description of the system with no details about the underlying hardware. The second difference is the processing architecture. While GPP architecture is fixed, the best possible processing architecture is built by the compiler for FPGA [122]. In addition, HLS can speed up the development cycle (time to market), going from several months down to several weeks [123]. This is because the task of producing an optimized RTL is handled by the HLS tool, and the developer's efforts are focused on describing the system's algorithmic description.

In [124] the authors presented LegUP, an open-source HLS tool. This tool is capable of profiling code to identify frequently executed sections of the code for hardware acceleration (i.e., moving them to the FPGA fabric). The authors in [125] survey HLS compilers and their capability to provide an accurate estimation of functional area and timing, and they compare them with the results from hand-tuned hardware designs. In an effort to help the developer make the right decision in picking an HLS tool that yields the best results for their application, the authors in [123] present a study where they compared three of the industry tools, namely Vivado HLS [93], Intel FPGA SDK for OpenCL [126], and MaxCompiler [127], through developing LDPC decoders, which are often used as error correcting blocks in SDRs. All three tools successfully synthesized LDPC decoders and implemented them on Intel [51] and Xilinx [85] FPGA boards. The difference, however, was in the logic utilization and performance. Similarly, the authors in [128], compare the same aforementioned list of compilers quantitatively and qualitatively using several financial engineering problems (e.g., Monte Carlo-based Option Pricing) and compare the performance of several FPGA boards. Their results show that both Intel FPGA SDK for OpenCL and MaxCompiler performed better than Vivado HLS due to their ability to extract parallelism more effectively. In [129], the authors comprehensively review recent HLS tools and provide a methodology based on C benchmarks to compare some of these tools and their optimization features. The various benchmarks implemented demonstrate that some tools are better suited for certain applications than others, with no specific tool dominating the HLS field. The authors also show that open-source HLS tools, such as LegUP [130], can be as effective as their commercial counterparts. Other surveys and analyses include [104,131,132] which focused on open-source tools, and [133], which studied some of the trade-offs of HLS-generated designs and their degree of reliability when errors are injected. All of the studies above prove the feasibility and reliability of HLS tools to generate RTL codes, despite having different development and optimization solutions.

Examples of HLS tools include Xilinx Vivado HLS [93] and SDSoC [135]; Intel HLS Compiler [134] and FPGA SDK for OpenCL [126]; Cadence Stratus High-level Synthesis, which combines Cadence C-to-Silicon and Forte Cynthesizer [136]; Synopsys Synphony C Compiler [137]; Maxeler MaxCompiler [127]; MATLAB HDL Coder [91]; and LegUP [130], which unlike the rest of the tools is vendor-independent (works with all types of FPGA boards like Xilinx [85], Intel [51], Lattice [97], and Microsemi [98]).

Table 7 presents a summary of the commercial HLS tools. While some of them are vendor-specific, other tools work with a variety of FPGA boards. The examples mentioned in the table all provide a set of area and timing optimizations such as resource sharing, scheduling, and pipelining. However, not all of them are capable of generating testbenches for the design.

4.2. Tools

In this section we review the existing software tools for SDR development. For each design methodology, we discuss a compatible development tool and list its features. We also provide an overall comparison between them to highlight the differences. This review is particularly important in order to make the right decision of picking the most fitting tool for the intended application. Learning about the features offered by each tool helps the developers fully utilize the available tools. Table 8 presents an overview of these tools.

4.2.1. MATLAB and Simulink

Most designers start with modeling and simulating the system using Mathworks MATLAB [92] and Simulink [140]. With the availability of a wide range of built-in functions and toolboxes, especially for signal processing and communication, developing and testing applications became very common and widely adopted. However, in order to use

Table 7
HLS tools.

	Xilinx Vivado HLS [93]	Intel FPGA SDK for OpenCL [126]	Cadence Stratus High-level Synthesis [136]	Synopsys Synphony C Compiler [137]	Maxeler MaxCompiler [127]
Input	C/C++/SystemC	C/C++/SystemC	C/C++/SystemC	C/C++	MaxJ
Output	VHDL/Verilog/SystemC	VHDL/Verilog	VHDL/Verilog	VHDL/Verilog/SystemC	VHDL
Testbench	Yes	No	Yes	Yes	No
Optimizations	Yes	Yes	Yes	Yes	Yes
Compatibility	Xilinx FPGA	Intel FPGA	All	All	All

Table 8
Development tools and platforms.

	MATLAB & Simulink [138]	Vivado HLS & SDSoC [85]	LegUP [130]	GNU Radio [47]	LabView [139]	CUDA [55]
Input	MATLAB/Graphical	C/C++/SystemC	C	Graphical/Python/C++	Graphical	C/C++/FORTRAN/Python
Output	MATLAB/C++/RTL	C/RTL	C/RTL	C/RTL	C/RTL	Machine Code
Platform	GPP/GPU/DSP/FPGA	GPP/FPGA	GPP/FPGA	GPP/GPU/DSP/FPGA	GPP/GPU/DSP/FPGA	GPU
Licence	commercial	commercial	open-source	open-source	commercial	commercial

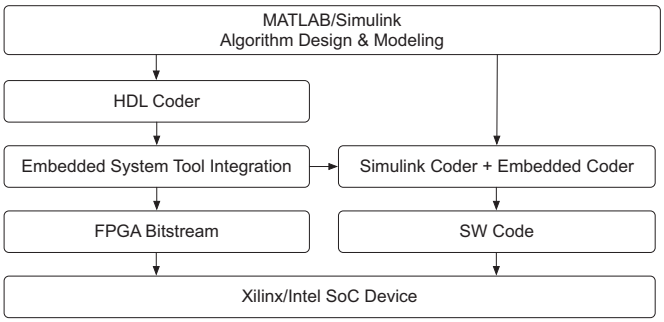

Fig. 5. Mathworks SoC design flow [138].

these models for different platforms, developers would need to use MATLAB Coder [141] and Simulink Coder [142] to generate C/C++ codes. The generated codes can be used with Embedded Coder [143] to optimize them and generate software interfaces with AXI drivers for the sake of running on embedded processors and microprocessors, like the dual ARM cortex A9 MPcore [103] on the ZedBoard [144]. Alternatively, developers can use the HDL Coder [91] to generate synthesizable RTL (Verilog or VHDL) code to be implemented on FPGAs or ASICs. It also has support for Xilinx [85] and Intel [51] SoC devices by providing some information and optimizations that pertain to resource utilization and distributed pipelining. Fig. 5 shows the design flow for SoC platforms that the aforementioned tools offer and how they are connected. Examples of using MATLAB and Simulink to develop an SDR are found in the works by [145] and [146], where the authors used the RTL-SDR very low-cost SDR dongle [147] (≈ $20) with a desktop computer (GPP) to design an academic curriculum for teaching DSP and communications theory.

4.2.2. Vivado HLS and SDSoC

Xilinx Vivado HLS [93] is a design environment for high-level synthesis. This tool offers a variety of features to tweak and improve the RTL netlist output that is compatible and optimized for Xilinx FPGA boards. It accepts input specifications described in several languages (e.g., C, C++, SystemC, and OpenCL) and generates hardware modules in Verilog or VHDL. Developers are provided with several options to optimize the solution in terms of area and timing through the use of directives, which are guidelines for the optimization process, and pragmas for RTL optimization. These optimizations include loop unrolling, loop pipelining, and operation chaining. SDSoC [135] is another tool by Xilinx [85]. The major difference between the two tools is that the latter has the capability to provide solutions for SoCs. SDSoC is built on top of Vivado HLS and has the same C-to-RTL conversion capability. The main advantage of using SDSoC is that it automatically generates data movers, which are responsible for transferring data between the software on the processor and the hardware on the FPGA.

A similar tool to SDSoC that is open-source is LegUP [130]. It was developed at the University of Toronto, as part of an academic research effort to design an HLS tool that is capable of taking in C code as an input and providing three possible outputs: a synthesizable RTL code for an FPGA, a pure software executable, and a hardware/software co-design solution for a SoC.

4.2.3. GNU Radio

It is an open-source software development toolkit that provides signal processing blocks to implement SDRs [47,148]. It runs on desktop or laptop computers and can build a basic SDR, with the addition of simple hardware such as USRP B200 [21]. It is often used by academia and the research community for simulation, as well as to quickly set up SDR platforms. Similar to the System Generator tool [149] and Simulink [140], it includes different kinds of blocks such as decoders, demodulators, and filters. It is also capable of connecting these blocks and managing data transfer in a reliable fashion. In addition, it supports the popular USRP systems [21]. One of the attractive features of GNU Radio is the ability to define and add new blocks through programming in C++ or Python. An example of using GNU Radio is in the work by [150], where the author uses it with a USRP to realize different types of transceivers such as Time Division Multiplexed Access (TDMA) and Carrier Sense Multiple Access (CSMA). Similarly, the authors in [151] successfully achieve real-time communication between two computers using USRP [21] and RTL-SDR [147].

4.2.4. LabVIEW

A widely used tool from National Instruments [139] that offers a visual programming environment for test, automation and control applications used by both industry and academia. It is similar to GNU Radio and Simulink, where the design can be constructed schematically by connecting a chain of various blocks together, each of which performs a certain function. It also offers complete support for USRP [21] to enable rapid prototyping of communications systems. Designing different blocks of the system can be achieved using high-level languages, such as C or MATLAB, or using a graphical dataflow. An SDR platform development using LabVIEW is found in the work by [152], where the author describes a wireless communication course design that incorporates USRP and LabVIEW, due to their ease of use, in order to help teach students basic concepts. Similarly, in [153] the authors designed an SDR platform, namely FRAMED-SOFT, that includes two types of USRPs and is intended for an academic environment.

4.2.5. CUDA

Developed by NVIDIA, it issues and manages computing platforms and programming models for data-parallel computing on GPUs [55]. Developers typically use CUDA when GPUs are part of the processing architecture as co-processors; they want to take full advantage of their power by speeding up applications. As discussed in Section 3.1.2, in order to identify application components that should be run on a GPP and the parts that should be accelerated by the GPU, one needs to look at the tasks at hand. Programming languages that can be used in CUDA include C, C++, Python, FORTRAN, and MATLAB [92]. In addition to the rich library full of GPU-related acceleration functions, the toolkit includes a compiler, development tools, and a CUDA runtime library. It is used to develop applications and optimize them for systems that incorporate GPUs.

5. Platforms

In this section, we list the different types of SDRs from the architecture and design point of view. We analyze them, examine their strengths and shortcomings, and discuss their impact on SDR

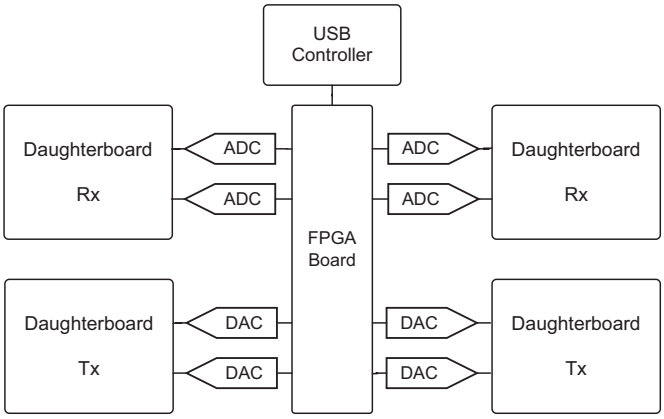

Fig. 6. USRP board architecture [21]. RF daughterboard selection depends on the application specifications in terms of frequency coverage.

development. The depth and elaboration level of each platform depends on its novelty, complexity, popularity in the SDR developers' community, and whether it played a vital role in developing new implementations.

5.1. GPP-based

USRP N-Series. Universal Software Radio Peripheral (USRP) is the most common SDR platform known to the developers' community [21]. The cost of this platform is around $4000–5000. It provides a hardware platform for the GNU Radio Project [148]. There are two generations available: USRP1 and USRP2. USRP1, released in 2004, is connected to a generic computer through USB, with the addition of a small FPGA. The FPGA board has two roles: routing information, and limited signal processing. This generation was capable of supporting a ≈ 3 MHz bandwidth due to USB 2.0 limitation. The second generation, USRP2, was released in 2008, and it supports 25MHz bandwidth by utilizing gigabit Ethernet. It includes a Xilinx Spartan 3 FPGA [85] for local processing operations.

USRP, in general, is a board with ADC and DAC, an RF front end, a PC host interface, and an FPGA. This board consists of a motherboard and, typically, four daughterboards (two transmitters Tx and two receivers Rx), as depicted in Fig. 6. The daughterboards process analog operations like filtering and up/down conversions. They are modular, so they can deal with applications operating up to 6 GHz. Depending on the USRP series, the FPGA board handles a few signal processing operations, and the majority of operations are offloaded to the connected host system. USRP platforms can be easily set up to use. However, while their performance is suitable for research experiments and quick prototyping, these platforms do not necessarily meet the requirements of communication standards. In fact, the minimum bandwidth of the RF, PC host, or FPGA component used affects the throughput and timing characteristics of the platform.

KUAR. Another platform that is similar to USRP is the Kansas University Agile Radio (KUAR) platform [41]. The basic architecture is composed of a generic computer and a Xilinx Virtex II Pro-P30 FPGA, which has two PowerPC 405 cores [85]. The main advantage of this platform is the degree of flexibility that it offers. Developers have the option of implementing communication standards in three different methods: (i) They can fit the entire design onto the FPGA and assign few tasks to the host's GPP (full hardware); (ii) They can build a full software implementation, where the design is implemented entirely on the GPP with minimal FPGA involvement; and (iii) They can create a hybrid hardware/software co-design implementation, where the developer can partition the design in any way that fits their criteria.

LimeSDR. Resembling the general basic architecture of USRP (e.g., USRP B200 [21]), Lime Microsystems [154] developed a series of SDRs that is based on Lime Microsystems' latest generation of field programmable RF transceiver (FPRF) technology, in addition to an Intel FPGA [51] and a microcontroller. It is then connected to a computer via USB 3.0. The SDR platform has the responsibility of delivering the wireless data, while the GPP (computer) has the task of processing the incoming signals and generating the data to be transmitted by the SDR. The GPP in this case is the source of computing power for baseband. LimeSDR supports signal bandwidth $\approx$ 30–60 MHz. Developers of LimeSDR also developed LimeSuite software, which is used to model SDRs. This tool, source code, firmware, and schematics are open-source.

Ziria. It is a programming platform that uses a Domain-Specific Language (DSL) named Ziria and an optimizing compiler [155]. The application of Ziria is the implementation of the PHY layer of wireless protocols. Ziria has a 2-layer design:

- A lower layer that is an imperative language, which is a mixture of C and MATLAB [92] code, with the features of the two languages carefully chosen to guarantee efficient compilation.
- A higher layer, which is the language used to specify and stage stream processors.

The Ziria optimizing compiler consists of two parts: the frontend and the backend. The frontend parses the Ziria code, puts it in an abstract representation, and then applies several optimizations on it. The backend compiles the resulting optimized code into an optimized, low-level execution model.

The particular benefits of this platform are as follows. The first benefit is easy and dynamic reconfiguration due to the dynamic staging of the control graph. This is unlike the GNU Radio [47] C++ templates that only allow limited reconfigurability. In addition, its code optimization can operate on data-flow components and can often yield a faster execution on GPPs (e.g., through the use of LUTs). In general, Ziria code is very concise and easy to use. For example, an implementation of a WiFi scrambler in Ziria only needs thirteen lines. Ziria is interesting research that is based on data-flow languages, which are typically used in embedded systems. It also builds upon popular functional reactive programming framework.

Sora. Sora is a fully programmable software radio platform on PC architecture. It requires C programming on multi-core GPP and yields high performance that includes high processing speed and low latency. The Sora platform uses the Ziria language discussed above to write high-level SDR descriptions and is tested for real-time operation [22]. Unlike WARP [25] (which we will explain in Section 5.5), Sora can accommodate various RF front ends.

Since PC hardware is not intended for signal processing of wireless protocols, their performance can be limited. We discussed some of the limitations of GPPs in the context of SDRs in Section 3.1. These limitations were the motivation behind the development of Sora. The overall setup of Sora includes a soft-radio stack that combines a multi-core GPP and a radio control board, which consists of a Xilinx Virtex-5 FPGA [85], PCIe-x8 interface, and Double Data Rate 2 (DDR2) Synchronous Dynamic Random Access Memory (SDRAM). Sora uses both hardware and software techniques to address the challenges of using PC architectures for high speed SDR. It is the first SDR platform that enables users to develop high speed wireless implementations entirely in software on a standard PC architecture.

In Sora, new techniques are proposed for efficient PHY implementation. Some of these techniques include: (i) exploiting large high-speed cache memory to minimize memory accesses, (ii) extensive use of LUTs, where they would trade memory for calculation and still fits well into L2 cache, (iii) exploiting data parallelism in PHY, and (iv) utilizing wide-vector SIMD extension in a GPP. One of the main novelties of Sora is its capability of efficiently partitioning and scheduling the PHY processing across cores in a GPP. The second innovation is core dedication for real-time support. This was accomplished by exclusively allocating enough cores for SDR processing in multi-core systems. They

showcased its effectiveness through SoftWiFi, which is a full implementation of IEEE 802.11a/b/g PHY and CSMA MAC, using 9000 lines of C code and real-time performance. They also successfully implemented a 3GPP LTE Physical Uplink Shared Channel (PUSCH), or Soft-LTE, with 5000 lines of C code and a peak rate of 43.8 Mbps with 20 MHz.

Some of the critiques to Sora is that its FPGA is not programmable, and its capabilities are not fully utilized. In addition, the authors do not share the details of its internal routines. Lastly, Sora only works on GPPs, and there is no clear mapping to DSPs.

Iris. It is a cross-platform SDR, developed to support highly reconfigurable radio networks [156]. This is due to its plugin architecture, namely *components*, that helps to achieve modularity. These components process data streams and perform different operations on them. An engine is used to run, load, and maintain the components. Similar to the System Generator [149] and Simulink [140] tools, the Iris engine can be used to link components together to build a complete radio system. XML is used to specify the components used in the program, their parameters, and how they should be linked. The main features of the Iris architecture include: (i) runtime configurability, (ii) support for the entire network stack (all layers), not just the PHY layer, and (iii) support for advanced processing platforms including FPGAs.

There are multiple studies on using Iris. An example is the work in [157], where the authors discuss the process through which they were able to implement Iris on Xilinx Zynq SoC [85]. The motivation behind this work is the fact that communication systems are in a constant state of development, and they increase in complexity and sophistication. This calls for higher computational performance and a higher level of flexibility. In order to implement Iris on the Zynq platform, the components are first translated into C++ using Cmake tools, and then they are ported to the platform. HLS tools, like Xilinx HLS [93], can be used to accelerate parts of the design that are considered to be the bottlenecks. This is done by running system profiles, like Linux Perf, on the software components. Acceleration, in particular, refers to running parts of the software (after re-writing it in RTL) on the FPGA fabric in order to achieve higher performance.

5.2. GPU-based

WiMAX SDR. The authors in [158] built a GPU-based platform to construct a WiMAX system. In their study, they also compared the performance of GeForce 9800GTX GPU [52] against a TMS320C6416 DSP [66] via implementing the Viterbi decoder algorithm. The results indicate that the throughput of the GPU is 181.6 Mbps, which is a considerable difference compared to the DSP's 2.07 Mbps.

OFDM for WiFi Uplink SDR. In [59], the authors used the WARP framework [25] as a basis for implementing their NVIDIA GPU-based SDR platform. They utilized the inherent parallelism of GPUs and, with the help of CUDA [55], they were able to achieve real-time performance on WARP. They used this platform to design and implement a Single Input Single Output (SISO) OFDM system for WiFi uplink communication. Fig. 7 depicts the architecture of this enhanced (accelerated)

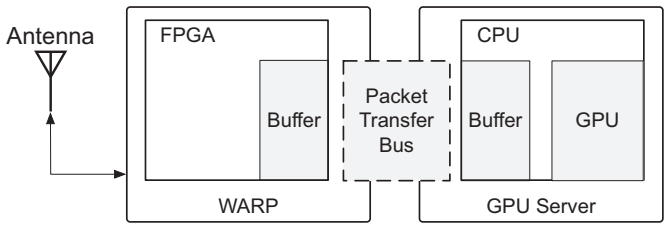

Fig. 7. GPU-accelerated SDR Platform using WARP [59]. The GPU server is used for baseband processing, while WARP is used for radio control and interface. Offloading signal processing tasks to the GPU has significantly improved the overall performance.

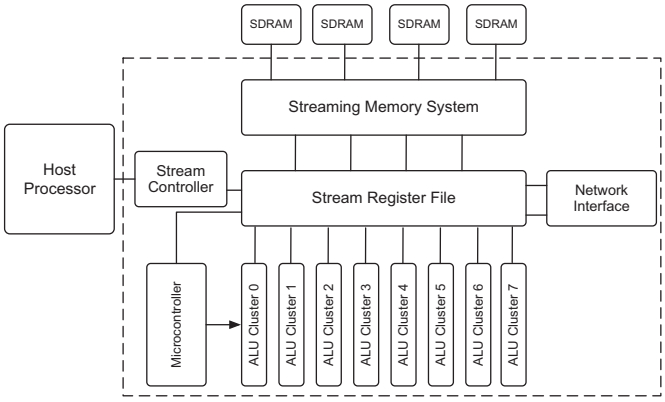

Fig. 8. Imagine Processor Architecture [160]. This platform employs a real-time stream processor for baseband processing.

WARP SDR. For this platform, they used: (i) a WARP version 3, which consists of a Xilinx Virtex-6 FPGA [85] for radio control and interface, and (ii) a GPU server, which consists of an Intel i7-3930K six-core 3.2GHz CPU [51] for transceiver configuration, and four NVIDIA GTX TITAN graphic cards [52] for baseband processing. Each TITAN is comprised of 2880 core Kepler GPU running at 889 MHz. The accelerated WARP achieves less than 3 ms latency and higher than 50Mbps Over-the-Air throughput.

Signal Detection SDR. In [53] the authors designed and studied real-time signal detection using an SDR platform comprised of a laptop computer with an Intel Xeon E3-1535M processor [51] and an NVIDIA Quadro M4000M GPU [52]. For 1000 ms long samples, this design reduces the parallel processing time by 75%, compared to GPPs.

5.3. DSP-based

Imagine Processor-based SDR. Authors of [159] proposed one of the earliest SDR solutions that is fully based on a DSP. This SDR employs the Imagine stream processor, developed at Stanford University in 2001 [160]. The Stanford Imagine project aimed at providing a signal and image processor that was C programmable and was able to match the high performance and density of an ASIC. It is based on stream processing [161–163], which is similar to dataflow programming in exploiting data parallelism and is suitable for signal processing applications. This work paved the way to the development of GPUs.

As Fig. 8 shows, the Imagine processor uses VLIW-based ALU clusters that are arranged in a SIMD fashion to handle data streams. At the middle of the architecture, there is the Stream Register File, which stores data from other components, thereby minimizing memory accesses. The performance of this platform has been evaluated by implementing complex algorithms relevant to W-CDMA cellular system. The results show a higher performance compared to TI C67 DSP [66], where channel estimation and detection are improved by 48x and 42x, respectively.

SODA. In [74], the authors introduce the Signal-processing On-Demand Architecture (SODA), which is an SDR platform based on multi-core DSPs. It offers full programmability and targets various radio standards. The SODA design achieves high performance, energy efficiency, and programmability. This is attributed to a combination of features that include SIMD parallelism and hardware support for 16-bit computations, since most algorithms operate on small values. The basic processing element is an asymmetric processor, consisting of a scalar and SIMD pipeline, and a set of distributed scratchpad memories that are fully managed in software. SODA is a multi-core architecture, with one ARM Cortex-M3 processor [103] for control purposes and multiple processing elements for DSP operations. By using four processing elements, it is able to meet the computational requirements of 802.11a and W-CDMA. Compared to WARP and Sora, as a single-chip

implementation, SODA is more appropriate for embedded scenarios. As with WARP, the developers must learn the architecture in order to implement SDRs.

ARM Ardbeg. In [164], a commercial prototype based on the SODA architecture was presented. The main enhancements of Ardbeg compared to SODA are optimized SIMD design, VLIW support, and a few special ASIC accelerators, which are dedicated to certain algorithms such as Turbo encoder/decoder, block floating point and arithmetic operations.

Atomix. Typically, programmers need to do one of three tasks to the software workflow of DSPs: tap into a signal processing chain, tweak a block, or insert/delete a block. To simplify these tasks, modularity is crucial. However, designing a modular software for DSPs is challenging considering the particular requirements that must be supported, such as latency sensitivity and high throughput. The main challenge is the need for programmers to define and manage everything manually and explicitly. In other words, it is necessary to use bare metal features, like moving data across cores, managing SRAMs, and parallelizing software.

In order to address these concerns, Atomix [23] describes the software in blocks, named atoms. An atom can be used to implement any operation. This can be signal processing or system handling. Atoms can be used for constructing blocks, flowgraphs, and states in wireless stacks. In addition, the simple control flow makes atoms composable. The easy modification feature mentioned above is due to declarative language. It is important to note that an Atomix signal processing block implements a fixed algorithmic function, operates on fixed data lengths, is associated with a specific processor type, and uses only the memory buffers passed to it during invocation. The blocks will run fixed sets of instructions executing uninterrupted on fixed resources using fixed memories. This results in having fixed execution times. Atoms can also be combined to build larger atoms. Using Atomix, radio software can be built entirely out of atoms and is easily modifiable. Atomix-based radio also meets throughput and latency requirements.

Developers define the atoms using the C language. Then, the blocks are composed into flowgraphs and states using the Atomix interface. The next step involves developing a parallelized schedule and resource assignment in order to meet latency and throughput requirements. The software is then compiled by Atomix into low-level C. The compiled code, along with Atomix libraries, is compiled into binary. Atomix was only used to build the IEEE 802.11a receiver, namely Atomix11a. Evaluation of Atomix11a shows that it exceeds receiver sensitivity requirements, operates in indoor environments robustly, and has low processing latency. Additionally, the atoms have low timing variability. Although the power consumption reported is 7 W, it does not include the power consumed by the front end, USRP2, which is about 14 W. A shortcoming of Atomix is that it is intended only for synthesis on a variety of DSPs, but not for GPPs, GPUs, or FPGAs.

BeagleBoard-X15. A collaborative project between Texas Instruments [66], Digi-Key [165], and Newark element14 [166], BeagleBoard is an open-source SoC computer [167]. It features TI Sitara AM5728 [66], which includes two C66x DSPs [66], two ARM Cortex-A15, two ARM M4 cores [103], and two PowerVR SGX544 GPUs [168]. With its relatively low price (≈ $270), the DSPs along with the co-processors make a powerful platform for implementing standalone SDRs. An example of using the BeagleBoard (an older model but with the same general architecture) is the work by [169], where it was used to implement the Public Safety Cognitive Radio (PSCR) [170] through GNU Radio [47]. PSCR is FM radio-based and was developed by the Center for Wireless Telecommunications (CWT) at Virginia Tech.

5.4. FPGA-based

Airblue. This work [24] introduces an FPGA-based SDR platform for PHY and MAC layers. Airblue is a method to implement radios on a FPGA to achieve configurability. This is done using an HDL language called Bluespec, through which all hardware blocks of a radio transceiver are written. In Bluespec, a developer describes the execution semantics of the design through Term Rewriting Systems (TRS). TRS is a computational paradigm based on the repeated application of simplification rules [171]. The next step is compiling the TRS into RTL codes. TRS has the capability of generating efficient hardware designs. The main difference between a Verilog interface and a Bluespec interface is that the former is merely a collection of wires with no semantic meaning, while the latter includes handshake signals for block communication. Therefore, Bluespec facilitates latency-insensitive designs, which are essential to system construction via modular composition. Using Airblue, developers may find the need to modify the building blocks, or modules, to add new features, make algorithmic modifications, tune the performance to meet throughput or timing requirements, or make FPGA-specific optimizations.

In order to reach modular refinements, the design of a configurable radio must have two design properties, latency-insensitivity and data-driven control. In addition to flexibility, Airblue meets tight latency requirements by implementing both PHY and the MAC on FPGA and connecting them with streaming interfaces, which allows data to be processed as soon as it is ready. Another advantage of Airblue is the implementation of highly reusable designs through parameterizations (i.e., same RTL block can be instantiated several times using different parameter values). Also, several techniques that permit designers to reuse existing designs to implement run-time parameterized designs are developed.

Airblue essentially is a HLS platform that offers modular refinement, where modifying one module does not affect the rest of the system, similar to the approach adopted by Atomix. This is why, when compared to WARP, Airblue is more flexible, since WARP was not designed with the aforementioned principles in mind. The authors in [24] also studied HLS tools and compared them to Bluespec. They found them to be more effective than Bluespec in early stages of the design; nevertheless, Bluespec is capable of yielding a more optimized final RTL. They argued that HLS will be of limited use in final FPGA implementations, especially for the high-performance blocks required by future wireless protocols. Therefore, Bluespec is the language of choice for Airblue. For performance evaluation, the authors have implemented 802.11 and experimented with a set of protocol changes. Airblue demonstrated that it was easily modifiable and still meets timing requirements. Airblue achieves a higher speed than Sora for cross-layer communication (between MAC and PHY layers), which typically has the latency requirement of a few microseconds.

5.5. Hybrid design

USRP Embedded (E) Series. This is the embedded version of USRP, where they incorporate Xilinx SoCs [85] to develop standalone SDRs. USRP E310, for example, is based on Xilinx Zynq 7020, which yields high performance and is energy efficient. This USRP is stand-alone and suitable for mobile applications. It supports a frequency range from 70 MHz to 6 GHz and features a 2 × 2 Multiple Input Multiple Output (MIMO) RF Front End.

WARP. The Wireless open-Access Research Platform is another example of a co-design that is specifically developed to prototype wireless protocols [25]. It is programmable and scalable, however, parts of the device are implemented in ASIC, which makes it less flexible. WARP v3 contains a Xilinx Virtex-6 FPGA [85], which includes two MicroBlaze processors and a Gigabit Ethernet peripheral. It requires the use of the Xilinx Embedded Development Kit (EDK) [85] to design SDRs. It is also open-source, with the 802.11 reference design made available to the research community. WARP has become widely used in the research community due to its effectiveness in implementing various wireless protocols (e.g., 802.11 g/n PHY and MAC) and achieving real-time performance. It also supports MIMO since it has multiple RF interfaces.

PSoC 5LP. Developed by Cypress Semiconductor [172], this SoC is composed of an ARM Cortex-M3 GPP [103], an analog system, and a

Table 9
Comparison of Existing SDR Platforms.

	Programmability	Flexability	Portability	Modularity	Computing power	Energy efficiency	Soft core	FPGA	Cost (USD)
Imagine-based [159]	✓	×	×	×	Medium	Low	Imagine Stream Processor	N/A	N/A
USRP X300 [21]	✓	✓	×	✓	High	Low	PC	Xilinx Kintex-7	≈ 4 – 5K Total
USRP E310 [21]	✓	✓	✓	✓	High	High	Dual-core ARM Cortex-A9	Xilinx Artix-4	≈ 3K Total
KUAR [41]	✓	×	×	×	Medium	Low	Pc + 2 × PowerPC cores	Xilinx Virtex II Pro-	N/A
LimeSDR [154]	✓	✓	×	✓	High	Low	PC	Intel Cyclone IV	≈ 300 Board Only
Ziria [155]	✓	✓	×	×	High	Low	PC	Depends on App	N/A
Sora [22]	✓	✓	×	×	High	Low	PC	Xilinx Virtex-5	≈ 900 Board Only
SODA [74]	✓	✓	✓	×	High	High	ARM Cortex-M3 + Processing Elements	N/A	N/A
Iris [156]	✓	✓	✓	✓	High	High	Dual-core ARM Cortex-A9	Xilinx Kintex-4	≈ 1.2K Total
Atomix [23]	✓	✓	✓	✓	High	Medium	TI 6670 DSP	N/A	≈ 200 DSP Only
BeagleBoard-X15 [167]	✓	✓	✓	✓	High	Medium	2 × TI C66x DSPs + 2 × ARM Cortex-A15 & 2 × M4	N/A	≈ 270 Board Only
Airblue [24]	✓	✓	✓	✓	High	High	N/A	Intel Cyclone IV	≈ 1.3K Board Only
WARP v3 [25]	✓	×	✓	✓	High	High	2 × Xilinx MicroBlaze cores	Xilinx Virtex-6	≈ 7K Total
PSoC 5LP [172]	✓	×	✓	✓	Low	High	ARM Cortex-M3	N/A	10 Board Only
Zynq-based [174]	✓	✓	✓	✓	High	High	Dual-core ARM Cortex-A9	Xilinx Kintex-4	≈ 1.2K Total

digital system that uses Universal Digital Blocks (UDBs). A UDB is a programmable digital block based on Programmable Logic Devices (PLDs) for realizing synchronous state machines, i.e., they resemble an FPGA but are smaller. All parts are reconfigurable and programmable by using the PSoC Creator software IDE, which includes a graphical design editor. A few developers have used it to build standalone SDRs due to its simplicity [173]. In addition to its low price ($15), PSoC is a good candidate for quickly prototyping and getting familiar with SDRs.

Zynq-based SDR. The authors in [174] developed an SDR using the Xilinx Zynq ZC706 and ZedBoard SoCs to implement IEEE 802.11a. For their RF Front End, they used Analog Devices FMComm3 AD9361 [76]. They used two Zynq boards to compare their performances. They both include dual-core ARM Cortex-A9 [103], with the ZC706 containing Kintex-7 and the ZedBoard containing Artix-7 FPGAs. In addition to the HDL Coder [91] and Embedded Coder [143] to generate RTL and software executable, they used Mathworks Simulink [140] to generate the model. They reported an average of 2W power consumption for Tx and Rx, compared to the 5–8 W reported by Atomix [23].

5.6. Comparison

Table 9 compares the SDR platforms discussed above in an effort to provide a reference guide for developers. SDR platforms are compared according to the following criteria:

- *Programmablity*: As protocols evolve, a platform is re-programmed by simply adding or exchanging parts of the design. For example, when 3GPP issues an update for the LTE standard, instead of replacing the entire radio, an SDR is reprogrammed to accommodate the upgrade.
- *Flexibility*: A platform is capable of handling future wireless protocols as requirements become more demanding. This means an SDR should be able to support tighter timing requirements for next generation of protocols.
- *Portability*: A platform is standalone and readily deployed. This is generally a requirement for mobile and IoT applications.
- *Modularity*: A design's components are separated and recombined without internal module changes. For example, a developer may need to exchange a Viterbi decoder with a Turbo decoder without having to worry about the rest of the design.
- *Computing power*: Since performance depends on the protocol used, we merely evaluate the capability of platforms to implement a given protocol. We are not limited to a subset of the protocol (e.g., implementing the Viterbi decoder only).
- *Energy efficiency*: Similar to computing power, we evaluate the capability to implement protocols efficiently, while keeping power consumption at a minimum.
- *Cost*: The cost of the hardware equipment. If a platform requires a PC, its cost is not included. Also, when the price of the entire setup is unknown, the price of the basic platform (not including RF Front End or interfaces) is shown.

6. Related research and potential solutions

Although in the previous sections we have highlighted some of the challenges of building SDRs, in this section, we present additional research areas that are still being faced by the research and development community. These challenges are technical or practical.

6.1. Remote system update

One of the main features and motivations of SDRs is their reconfigurability and flexibility. In order to take full advantage of this, the process of updating a SDR platform should be quick and easy. Remote stand alone SDRs are usually FPGA and DSP-based, with more FPGAs being used. Hence, most of the research has been focused on

remote updates of FPGAs. Since FPGAs are volatile, which means that they are configured during system power-on through their flash memories, an update is traditionally done using Joint Test Action Group (JTAG) method [175]. However, with more SDRs deployed and adopted in wireless and cellular networks, a remote update becomes necessary and a challenge. With remote Over the Air (OTA) updates, it becomes possible to send patches to current designs, or even upload an entirely new and improved design to mobile networks and BSs [176,177]. Some of the challenges faced by the research community include speed, reliability, cost, and security [178].

In [175], the authors introduce a method based on RS-422 serial communication and High Level Data Link Control (HDLC) protocol to update DSP and FPGA systems. Their method is fast, stable, and easy to implement. The authors in [179] show a new method for remotely updating Xilinx FPGAs [85] by storing the new design or code in Serial Peripheral Interface (SPI) flash memories. They utilized the Xilinx Quick Boot application, along with the KC705 Evaluation Kit from Xilinx, in order to develop their method. However, it is not always practical or feasible to use a download cable in order to update the system. The authors in [178] tackle this issue and the problem of downtime during an update or in the case of a failure. With this method, there are two images, namely a factory mode configuration image, which acts as the baseline, and an application mode configuration image, which is used for some specific functions. They show the capability of updating and running a new application mode (design) image, in addition to rolling back into the factory mode image when no application mode image is available. Others have focused their efforts on improving the security of remote updates, such as [180] and [181]. While there exists a few solutions to the remote update process, a few challenges, including partial reconfiguration of FPGAs, are not yet resolved.

6.2. Centralized algorithms and network slicing

In order to simplify the design and provisioning of large-scale networks, Software-Defined Networking (SDN) has been proposed to centralized network control. In this architecture, a controller (or multiple controllers) communicates with network devices (data plane) to collect their status information and configure their operation. Recent studies show that wireless networks can significantly benefit from the central network view established in the controller in order to design more efficient network control algorithms such as channel assignment and mobility control.

Centralized control of network resources is the enabler of network slicing, which refers to the abstraction, slicing, and sharing of network resources. Three levels of slicing are applicable to wireless networks: (i) spectrum (a.k.a., link virtualization): requires frequency, time or space multiplexing; (ii) infrastructure: the slicing of physical devices such as BSs; (iii) network: this refers to the slicing of the network infrastructure. Compared to wired networks, slicing the resources of wireless networks is significantly more challenging due to the variable nature of wireless channel. Meanwhile, since SDRs enable the slicing of resources both at the spectrum and infrastructure level, they can be used to augment SDNs towards a fine-grain allocation of resources. For example, compared to ASIC-based transceivers, SDRs provide a significantly higher level of control over the parameters of PHY and MAC layer.

It should be noted that when centralized network control is employed, the delay of communication between the controller and SDR platforms as well as the delay of programming and applying new configurations should be bounded within the specification requirements. For example, in a dense and mobile environment, the controller may employ a centralized channel and power control algorithm to instruct the nodes to adjust their channels based on the decisions made centrally. In this case, it is essential to ensure that the delay of sending configuration messages to multiple SDR platforms meets the requirements of the central algorithm. Furthermore, it is essential to ensure that all of the SDRs apply the configuration at the same time, otherwise serious interference and collision might happen. Although protocols such as OpenFlow [182] and Netconf [183] have been designed for interactions between the controller and data plane, their implications on the performance of wireless networks have not been investigated. Specifically, it is essential to evaluate the effect of hardware platforms (i.e., GPP, DSP, FPGA) on the delay of applying configurations. These challenges have not been addressed yet.

6.3. Network Functions Virtualization (NFV)

One of the new topics is the concept of Network Functions Virtualization (NFV), which offers an alternative way of designing and managing networking functions. The concept is very similar to SDRs, in the sense that various network functions can run in software on top of different hardware platforms. These platforms are typically high-volume servers, storage devices, and cloud-computing data centers [184]. Some of the functions that are virtualized via this method are load balancing, firewalls, intrusion detection devices, and WAN accelerators. In addition to cost reductions, this flexibility is what makes NFV very attractive for network operators, carriers, and manufacturers [185]. From the SDR point of view, instead of performing signal processing operations on the platform, these operations are offloaded to a general computing platform. Such an architecture reduces the load of edge devices, and BSs can benefit from powerful processors to implement complex signal processing operations. For example, when multiple SDRs are connected to a computing platform, sophisticated signal processing algorithms could be developed to cope with challenges such as interference.

To leverage NFV for SDRs, developers have been attempting to tie them together to achieve complete flexibility across the platform [186,187]. Although several wireless SDN architectures have been proposed to address the challenges of wireless communication [188–192], most of them do not benefit from the features of SDRs.

6.4. Energy efficiency

Battery-powered devices in an IoT network face the challenge of minimizing power consumption in order to extend battery-life before they are due for a replacement, which is a costly operation. Ready-to-deploy SDR systems may use high-performing platforms, such as FPGAs, without providing solutions or alternatives to batteries [193]. Even in the case of BSs with on-grid power sources, it has become crucial to lower power consumption in order to reduce CO_2 emissions [194]. This is particularly important for cellular network operators, where BSs consume more than 50% of the total energy consumed in the network [195]. To address these concerns, energy harvesting mechanisms have been introduced. Energy harvesting or scavenging is the process of deriving power from external sources, such as solar and wind energies (also, referred to as green energy), and stored for consumption alongside internal sources (e.g., battery or electrical grid sources) [196]. As this green energy is a viable option for powering BSs [197], Ericsson (a major telecommunication company) has invested into and designed green-energy-powered BSs motivated by environmental and financial reasons [198]. Therefore, a hybrid power operation has been accepted as a solution to lower electrical grid energy consumption and cost [195].

In [196], the authors present a survey on energy harvesting technologies with regards to transducers, such as antennas and solar cells, that can utilize renewable energy sources, and cover some of the applications in the IoT and M2M world. The authors in [197] overview the cellular network operators that have adopted the hybrid solution and started using green energies to power their BSs. The authors in [199] discuss the issue of implementing an energy harvesting transmitter in a cognitive radio. They also derive the optimal spectrum sensing policy that maximizes the expected total throughput under

energy causality and collision constraints. Energy causality indicates that the total consumed energy should be equal to or less than the total harvested energy. The collision constraint states that the probability of accessing the occupied spectrum is equal to or less than the target probability of a collision. The authors concluded that a battery-operated radio can operate for a long time by optimizing both the energy and spectral efficiency. Energy harvesting is often associated with what is known as "Green IoT", which is the new trend for IoT networks and devices, where the main focus is making them more energy efficient. Another relevant work is [200,201], where the authors present an overview of the challenges and existing solutions of energy-efficient IoT systems.

6.5. Co-design

By definition, co-design is the process of realizing system-level goals by exploiting the trade-offs between software and hardware in an integrated design. This yields a higher design quality, and optimizes the cost and design cycle time, which in turn shortens the time to market. As co-design is adopted for more applications, developers typically face the problem of partitioning and scheduling. Finding the optimal design partition is not trivial. While system profilers can assist in providing insight into the system's behavior and can help identify the parts of the code that can be accelerated on hardware, partitioning should be an automatic process and requires no external involvement. There are several algorithms proposed to address the challenges of partitioning, such as PSO, FCM, and FCMPSO [202]. These are optimization algorithms used for mapping embedded applications to Directed Acyclic Graphs (DAGs) for multi-core architectures. However, in SDRs, the problem is even more complicated due to having two layers of operation, namely the PHY and MAC layer. Partitioning needs to take into account strict real-time requirements. It is a delicate equilibrium between performance, cost, and correct operation. Even with the process being more challenging than other design approaches, the benefits of co-design for complex systems outweigh the initial cost.

6.6. Security

SDRs simplify security provisioning. For example, when a new security mechanism does not require hardware replacement (e.g., 802.11i's WPA), it can be implemented by reprogramming SDRs. The reprogrammability feature of SDRs exposes security threats, whether they are standalone or part of a SDN architecture. Assume an 802.11 network employs SDR-based BSs (i.e., access points). In this case, a Denial of Service (DoS) attack can be implemented by instructing a large number of clients to associate with a BS. If the controller is compromised, then all of the SDRs might be reprogrammed to be nonfunctional. Therefore, it is important to identify security threats and take them into account when designing SDR-based wireless networks.

Offloading SDR processing to general processing platforms through NFV enables the deployment of sophisticated central algorithms for detecting abnormal activities and network breaches. For example, by analyzing the signal strength received from a client at one or multiple BSs, a denial of service attack that is caused by generating excessive interference could be detected. Proposing security mechanisms that benefit from the features of SDRs is an important future direction, especially for large-scale and public networks.

7. Existing surveys

Joseph Mitola III was the first to use the term "Software Radio" in 1993, when he published an important work that introduced and explained the concept of using software rather than traditionally-used hardware for designing radio systems [203]. In an early survey in 1999 [204] that was on the "then" emerging technology of SDR, the authors made the case that SDRs have a great potential to advance and facilitate the development of communication standards like WiFi and cellular networks. In a review that was published a few years later [26], the author paid close attention to the hardware component of SDRs, such as programmable RF modules and high-performance DACs and ADCs, as more technological advancement had been made. Around the same time, the authors in [205] put forward a survey of the advances that had been made in the SDR field, exclusively in Japan. They also discussed several prototypes that were developed by both academia and industry. They studied the current status of communication systems in Japan and concluded that SDRs were viable solutions for the challenges they faced.

There are very few notable works that survey and review different SDR platforms and testbeds. One such survey is the work by [45], where the author discusses the challenges faced by SDR developers. These challenges include size, weight, power, software architecture, security, certification and business opportunities. While this work is important for compiling information about these challenges and presenting them in one place, it abstains from enumerating and discussing the different SDR platforms, implementation approaches and their applications in the communication standards world.

In that regard, the authors of [206] compare the SDR platforms developed by the year 2012 and give a brief description of what each platform entails. It lacks, however, any detailed comparison based on the different categories of computational power, energy efficiency, flexibility, adaptability, and cost. It is through these comparisons that a designer is able to make an informed decision on what platform to adopt for their specific application. The authors of [207] attempt to list and review several DSP-based SDR platforms from both academia and industry, with more focus on commercial solutions, and then they provide a simple comparison between them in terms of programmability, power, and flexibility. However, what this work lacks, in addition to being outdated, is a more comprehensive discussion of FPGAs and hardware/software co-design solutions, as well as a methodical analysis of the different design approaches based on a set of performance metrics.

The authors of [208] presented a survey of a few SDR platforms that had been developed more than a decade ago. In [209], the authors presented a paper that laid out the development and evolution of SDRs over the past several years and discussed the motivation behind why it has been recently gaining more attention. Another work is the comparison conducted by the authors in [210] between the Imec Bear platform [211] and a few multicore-based SDR platforms. Other attempts include the work by [212], where the authors focused on discussing the analog end of the SDR concerning signal sampling, processing, as well as the hardware/software responsible for handling these tasks. The work by [213] compares several SDR platforms to prove the feasibility and reliability of using SDRs in education, industry, and government. Another outdated survey is [214].

Another survey that is relevant to SDR platforms is the work by [215], where the author discusses and reviews the state-of-the-art in microwave technology in transceivers. The paper explores the development of SDRs using different technologies in radio frequency engineering. It is a comprehensive study of several research topics, such as the design of tunable radio frequency components, linear and power efficient amplifiers, linear mixers, and interference rejection techniques. Similarly, the authors in [216] present a compilation of several studies that discuss research topics, ranging from SCA to spectrum sharing and new signal processing techniques to embedded systems. The topics are carefully selected in order to update prospective SDR developers on the latest technology than can help them build more efficient and powerful SDRs. Whereas, the authors in [217] provide a very thorough study on the several security threats and challenges in SDRs and go over their certification process. As SDR platforms grow in popularity and more communication protocols are realized using them, it becomes essential that developers are prepared for security issues and well-equipped with tools that protect their systems.

From the aforementioned surveys, it is apparent that there is a need for an up-to-date, comprehensive, and in-depth survey of the most prominent SDR platforms and development tools. Previous surveys are outdated, have a limited scope, did not study and analyze the development process, and, most importantly, did not offer a complete guide to future developers on SDR implementation. Furthermore, with more studies published on the capabilities of various processing devices, we think there is a significant need to analyze them and present an analysis of the different processing alternatives.

8. Conclusion

In this paper, we provided a comprehensive overview of the various design approaches and hardware platforms adopted for SDR solutions. This includes GPPs, GPUs, DSPs, FPGAs, and co-design. We explained the basic architectures and analyzed their advantages and disadvantages. Due to the different features of design approaches, it was important to compare them against each other in terms of computational power and power efficiency. We then reviewed the major current and past SDR platforms, whether they were developed by the industry or in academia. Finally, we discussed some of the research challenges and topics that are predicted to improve in the near future, helping to advance SDRs and their wide adoption.

We believe that SDR solutions are going to be mainstream and that their ability to implement different wireless communication standards with high levels of flexibility and re-programmability will be considered the norm. This paper poses as an exhaustive overview of this phenomenon–its enabling technologies, applications, and the current research that tackles this issue from different angles.

References

[1] WWRF Visions and Research Direction for the Wireless World, Technical Report, [Online]. Available: http://wwrf.ch/outlook.html.

[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash, Internet of things: A Survey on enabling technologies, protocols, and applications, IEEE Commun. Surv. Tutor. (2015), https://doi.org/10.1109/COMST.2015.2444095.

[3] B. Dezfouli, M. Radi, S.A. Razak, T. Hwee-Pink, K.A. Bakar, Modeling low-power wireless communications, J. Netw. Comput. Appl. 51 (2015) 102–126.

[4] B. Dezfouli, M. Radi, O. Chipara, Mobility-aware real-time scheduling for low-power wireless networks, The 35th Annual IEEE International Conference on Computer Communications (INFOCOM), IEEE, 2016, pp. 1–9.

[5] IEEE, IEEE 802.11, The Working Group Setting the Standards for Wireless LANs, [Online]. Available: http://www.ieee802.org/11/.

[6] 3GPP, 3GPP - Release 16. [Online]. Available: http://www.3gpp.org/release-16.

[7] M. Bansal, J. Mehlman, S. Katti, P. Levis, OpenRadio, Proceedings of the First Workshop on Hot Topics in Software Defined Networks - HotSDN '12, (2012), p. 109, https://doi.org/10.1145/2342441.2342464. New York, USA.

[8] Software Defined Radio Market by Application, Component, End User, Type - Global Forecast to 2021, Technical Report, ReportBuyer, 2016. [Online]. Available: https://www.reportbuyer.com/product/4364831/software-defined-radio-market-by-application-component-end-usertype-global-forecast-to-2021.html.

[9] Global Industry Analysts Inc, Software Defined Radio (SDR) - Global Strategic Business Report, Technical Report, (2017). [Online]. Available: https://www.researchandmarkets.com/research/pc9x7g/software_defined .

[10] B. Paillassa, C. Morlet, Flexible satellites: software radio in the sky, 10th International Conference on Telecommunications, ICT, 2 (2003), pp. 1596–1600.

[11] P. Angeletti, R.D. Gaudenzi, M. Lisi, I. Introduction, S. Division, C. Scientist, From bent pipes to software defined payloads: evolution and trends of satellite communications systems, System (June) (2008) 10–12, https://doi.org/10.2514/6.2008-5439.

[12] P. Angeletti, M. Lisi, P. Tognolatti, Software defined radio: a key technology for flexibility and reconfigurability in space applications, 2014 IEEE Metrology for Aerospace (MetroAeroSpace), (2014), pp. 399–403, https://doi.org/10.1109/MetroAeroSpace.2014.6865957.

[13] J. Seo, Y.-H. Chen, D.S. De Lorenzo, S. Lo, P. Enge, D. Akos, J. Lee, A real-time capable software-defined receiver using GPU for adaptive anti-jam GPS sensors. Sensors 11 (9) (2011) 8966–8991, https://doi.org/10.3390/s110908966.

[14] W. Xiang, F. Sotiropoulos, S. Liu, xRadio: an novel software defined radio (SDR) platform and its exemplar application to vehicle-to-vehicle communications, International Conference on Ad-Hoc Networks and Wireless, Springer, Cham, 2015, pp. 404–415.

[15] K.D. Singh, P. Rawat, J.-M. Bonnin, Cognitive radio for vehicular ad hoc networks (CR-VANETs): approaches and challenges, EURASIP J. Wireless Commun. Netw. 2014 (1) (2014) 49, https://doi.org/10.1186/1687-1499-2014-49.

[16] M. Kloc, R. Weigel, A. Koelpin, SDR implementation of an adaptive low-latency IEEE 802.11p transmitter system for real-time wireless applications, 2017 IEEE Radio and Wireless Symposium (RWS), (2017), pp. 207–210.

[17] Y. Chen, S. Lu, H.-S. Kim, D. Blaauw, R.G. Dreslinski, T. Mudge, A low power software-defined-radio baseband processor for the Internet of Things, 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA), (2016), pp. 40–51, https://doi.org/10.1109/HPCA.2016.7446052.

[18] Y. Park, S. Kuk, I. Kang, H. Kim, Overcoming IoT language barriers using smartphone SDRs, IEEE Trans. Mob. Comput. 16 (3) (2017) 816–828, https://doi.org/10.1109/TMC.2016.2570749.

[19] A. Devices, Software defined radio: past, present, and future, Analog Devices White Paper (2017).

[20] Anywave: Vanu. [Online]. Available: http://www.vanu.com/products/outdoor/anywave/.

[21] Ettus research - networked software defined radio (SDR). [Online]. Available: https://www.ettus.com/.

[22] K. Tan, H. Liu, J. Zhang, Y. Zhang, J. Fang, G.M. Voelker, Sora: high-performance software radio using general-purpose multi-core processors, Commun. ACM 54 (1) (2011) 99, https://doi.org/10.1145/1866739.1866760.

[23] M. Bansal, A. Schulman, S. Katti, Atomix: a framework for deploying signal processing applications on wireless infrastructure, 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), (2015), pp. 173–188.

[24] M.-C. Ng, K. Fleming, M. Vutukuru, S. Gross, Arvind, H. Balakrishnan, Airblue: a system for cross-layer wireless protocol development, Symp. Architectures for Networking & Communications Syst. (ANCS), (2010), pp. 1–11.

[25] WARP project. [Online]. Available: https://warpproject.org/trac.

[26] A. Haghighat, A review on essentials and technical challenges of software defined radio, MILCOM 2002. Proceedings, (2002), pp. 377–382, https://doi.org/10.1109/MILCOM.2002.1180471.

[27] U.L. Rohde, T.T.N. Bucher, Communications Receivers: Principles and Design, 4, McGraw-Hill Education, 1988.

[28] T.J. Rouphael, RF And Digital Signal Processing for Software-Defined Radio : A Multi-Standard Multi-Mode Approach, Newnes, 2009.

[29] J.J. Carr, The Technician's Radio Receiver Handbook : Wireless and Telecommunication Technology, Newnes, 2001.

[30] R. Walden, Analog-to-digital converter survey and analysis, IEEE J. Sel. Areas Commun. 17 (4) (1999) 539–550, https://doi.org/10.1109/49.761034.

[31] T. Hentschel, M. Henker, G. Fettweis, The digital front-end of software radio terminals, IEEE Pers. Commun. 6 (4) (1999) 40–46, https://doi.org/10.1109/98.788214.

[32] C. Bowick, J. Blyler, C.J. Ajluni, RF Circuit Design, Newnes/Elsevier, 2011.

[33] M.N.O. Sadiku, C.M. Akujuobi, Software-defined radio: a brief overview, IEEE Potentials 23 (4) (2004) 14–15, https://doi.org/10.1109/MP.2004.1343223.

[34] A. Collins, All programmable RF-sampling solutions, Xilinx White Paper (2017).

[35] A.Q. Nguyen, A.A. Kisomi, R. Landry, New architecture of direct RF sampling for avionic systems applied to VOR and ILS, IEEE Radar Conference (RadarConf), (2017), pp. 1622–1627.

[36] L.C. Choo, Z. Lei, CRC codes for short control frames in IEEE 802.11ah, IEEE 80th Vehicular Technology Conference (VTC2014-Fall), (2014), pp. 1–5.

[37] F. Berns, G. Kreiselmaier, N. Wehn, Channel decoder architecture for 3G mobile wireless terminals, in: Proceedings Design, Automation and Test in Europe Conference and Exhibition, pp. 192–197. 10.1109/DATE.2004.1269229.

[38] P.-Y. ChiuehTzi-Dar and Tsai, OFDM Baseband Receiver Design for Wireless Communications, John Wiley & Sons, 2008.

[39] B. Dezfouli, M. Radi, O. Chipara, REWIMO: a real-time and reliable low-Power wireless mobile network, ACM Trans. Sens. Netw. 13 (3) (2017) 17.

[40] B. Dezfouli, I. Amirtharaj, C.-C. Li, EMPIOT: an energy measurement platform for wireless IoT devices, arXiv:/1804.04794 (2018).

[41] G.J. Minden, J.B. Evans, L. Searl, D. DePardo, V.R. Petty, R. Rajbanshi, T. Newman, Q. Chen, F. Weidling, J. Guffey, D. Datla, B. Barker, M. Peck, B. Cordill, A.M. Wyglinski, A. Agah, KUAR: a flexible software-defined radio development platform, 2007 2nd IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, (2007), pp. 428–439, https://doi.org/10.1109/DYSPAN.2007.62.

[42] K. Kant, Microprocessors and Microcontrollers. Prentice-Hall Of India, 2014.

[43] T. Kazaz, C. Van Praet, M. Kulin, P. Willemen, I. Moerman, Hardware accelerated SDR platform for adaptive air interfaces, arXiv:/1705.00115 (2017).

[44] C.-H. Lin, B. Greene, S. Narasimha, J. Cai, High performance 14nm SOI FinFET CMOS technology with 0.0174m2 embedded DRAM and 15 levels of Cu metallization, 2014 IEEE International Electron Devices Meeting, (2014), pp. 3.8.1–3.8.3, https://doi.org/10.1109/IEDM.2014.7046977.

[45] T. Ulversoy, Software defined radio: challenges and opportunities, IEEE Commun. Surv. Tutor. 12 (4) (2010) 531–550, https://doi.org/10.1109/SURV.2010.032910.00019.

[46] R. Kamal, Embedded Systems : Architecture, Programming and Design, Tata MaGraw-Hill, New Delhi, 2003.

[47] GNU radio, [Online]. Available: https://www.gnuradio.org/.

[48] K. Vachhani, Multiresolution analysis: an unified approach using discrete wavelet transform on GNU radio, International Conference on Green Computing and Internet of Things (ICGCIoT), (2015), pp. 887–892, https://doi.org/10.1109/ICGCIoT.2015.7380588.

[49] M. Vestias, H. Neto, Trends of CPU, GPU and FPGA for high-performance computing, 24th International Conference on Field Programmable Logic and Applications (FPL), (2014), pp. 1–6.

[50] CPU vs GPU performance - michaelgalloy.com, [Online]. Available: http://michaelgalloy.com/2013/06/11/cpu-vs-gpu-performance.html.

[51] Intel Xeon Processors, [Online]. Available: https://www.intel.com/content/www/us/en/products/processors/xeon.html.
[52] NVIDIA - Visual Computing Technologies, [Online]. Available: http://www.nvidia.com/content/global/global.php.
[53] A. Fisne, A. Ozsoy, Grafik processor accelerated real time software defined radio applications, 25th Signal Processing and Communications Applications Conference (SIU), (2017), pp. 1–4.
[54] B. Cope, P.Y. Cheung, W. Luk, L. Howes, Performance comparison of graphics processors to reconfigurable logic: a case study, IEEE Trans. Comput. 59 (4) (2010) 433–448, https://doi.org/10.1109/TC.2009.179.
[55] CUDA toolkit documentation, [Online]. Available: http://docs.nvidia.com/cuda/.
[56] P. Szegvari, C. Hentschel, Scalable software defined FM-radio receiver running on desktop computers, 2009 IEEE 13th International Symposium on Consumer Electronics, (2009), pp. 535–539, https://doi.org/10.1109/ISCE.2009.5156861.
[57] Amd | processors | graphics and technology.
[58] J. Koomey, S. Berard, M. Sanchez, H. Wong, Implications of historical trends in the electrical efficiency of computing, IEEE Ann. Hist. Comput. 33 (3) (2011) 46–54, https://doi.org/10.1109/MAHC.2010.28.
[59] K. Li, M. Wu, G. Wang, J.R. Cavallaro, A high performance GPU-based software-defined basestation, 48th Asilomar Conference on Signals, Systems and Computers, (2014), pp. 2060–2064.
[60] K. Li, B. Yin, M. Wu, J.R. Cavallaro, C. Studer, Accelerating massive MIMO uplink detection on GPU for SDR systems, IEEE Dallas Circuits and Systems Conference (DCAS), (2015), pp. 1–4.
[61] J.G. Millage, GPU Integration into a Software Defined Radio Framework, Iowa State University, 2010 Ph.D. thesis.
[62] TMS320C6670 Multicore Fixed and Floating-Point System-on-Chip | TI.com. [Online]. Available: https://www.ti.com/product/tms320c6670.
[63] L.R. Rabiner, B. Gold, Theory and Application of Digital Signal Processing, Englewood Cliffs, N.J., Prentice-Hall, Inc., 1975, p. 777. 1975
[64] S.W. others Smith, The Scientist and Engineer's Guide to Digital Signal Processing, California Technical Pub. San Diego, 1997.
[65] S.A. Dyer, B.K. Harms, Digital Signal Processing, Advances in Computers 37 (1993) 104–117, https://doi.org/10.1016/S0065-2458(08)60403-9.
[66] SMJ320C80 Digital Signal Processor | TI.com. [Online]. Available: http://www.ti.com/product/SMJ320C80.
[67] A. Antoniou, Digital Signal Processing, Mcgraw-Hill, 2016.
[68] D.A. Patterson, J.L. Hennessy, Computer Organization and Design, Morgan Kaufmann (2007) 474–476.
[69] M.J. Flynn, Some computer organizations and their effectiveness, IEEE Trans. Comput. C-21 (9) (1972) 948–960, https://doi.org/10.1109/TC.1972.5009071.
[70] R. Duncan, A survey of parallel computer architectures, Computer 23 (2) (1990) 5–16, https://doi.org/10.1109/2.44900.
[71] M. Gschwind, H. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, T. Yamazaki, Synergistic processing in Cell's multicore architecture, IEEE Micro 26 (2) (2006) 10–24, https://doi.org/10.1109/MM.2006.41.
[72] Nxp | Microcontrollers and Processors. [Online]. Available: http://www.nxp.com/products/microcontrollers-and-processors:MICROCONTROLLERS-AND-PROCESSORS.
[73] Livanto ICE8060,. [Online]. Available: http://www.icerasemi.com/products/livanto_chipsets/livanto_soft_baseband/Livanto_ICE8060/index.html.
[74] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, K. Flautner, SODA: a low-power architecture for software radio, Proceedings - International Symposium on Computer Architecture, 2006 (2006), pp. 89–100.
[75] A. Gatherer, T. Stetzler, M. McMahan, E. Auslander, DSP-based architectures for mobile communications: past, present and future, IEEE Commun. Mag. 38 (1) (2000) 84–90, https://doi.org/10.1109/35.815456.
[76] TigerSHARC Processors | Analog Devices.
[77] CEVA- Leading Licensor of Signal Processing IP, [Online]. Available: https://www.ceva-dsp.com/.
[78] Qualcomm | Wireless Technology and Innovation. [Online]. Available: https://www.qualcomm.com/.
[79] G. Frantz, Digital signal processor trends, IEEE Micro 20 (6) (2000) 52–59, https://doi.org/10.1109/40.888703.
[80] R. Akeela, Y. El Ziq, Design and verification of IEEE 802.11ah for IoT and M2M applications, IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), (2017), pp. 491–496.
[81] X. Cai, M. Zhou, X. Huang, Model-Based design for software defined radio on an FPGA, IEEE Access 5 (2017) 8276–8283, https://doi.org/10.1109/ACCESS.2017.2692764.
[82] I. Kuon, R. Tessier, J. Rose, FPGA architecture: survey and challenges, Found. Trends Electron. Des. Autom. 2 (2) (2007) 135–253, https://doi.org/10.1561/1000000005.
[83] N. Grover, M.K. Soni, Reduction of power consumption in FPGAs -An overview, Inf. Eng. Electron. Bus. 5 (5) (2012) 50–69.
[84] D. Strenski, C. Kulkarni, J. Cappello, O. Design, P. Sundararajan, F. Programmable, G. Arrays, B. High, T. Graphical, Latest FPGAs Show Big Gains in Floating Point Performance, 2014. [Online]. Available: https://www.hpcwire.com/2012/04/16/latest_fpgas_show_big_gains_in_floating_point_performance/.
[85] Xilinx - All Programmable, [Online]. Available: https://www.xilinx.com/.
[86] D. Lewis, G. Chiu, J. Chromczak, D. Galloway, B. Gamsa, V. Manohararajah, I. Milton, T. Vanderhoek, J. Van Dyken, The Stratix 10 highly pipelined FPGA architecture, Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '16, New York, USA, (2016), pp. 159–168.
[87] K. Sano, S. Yamamoto, FPGA-based scalable and power-efficient fluid simulation using floating-point DSP blocks, IEEE Trans. Parallel Distrib. Syst. 28 (10) (2017) 2823–2837.
[88] S. Kestur, J.D. Davis, O. Williams, BLAS Comparison on FPGA, CPU and GPU, 2010 IEEE Computer Society Annual Symposium on VLSI, (2010), pp. 288–293.
[89] R. Woods, Wiley InterScience, FPGA-Based Implementation of Signal Processing Systems, John Wiley & Sons, 2008.
[90] U. Meyer-Baese, Digital Signal Processing with Field Programmable Gate Arrays, Signals and Communication Technology, Berlin, Heidelberg, 2014, https://doi.org/10.1007/978-3-642-45309-0.
[91] HDL Coder, [Online]. Available: https://www.mathworks.com/products/hdl-coder.html.
[92] MATLAB - MathWorks, [Online]. Available: https://www.mathworks.com/products/matlab.html.
[93] Vivado High-Level Synthesis, [Online]. Available: https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html.
[94] Intel FPGA and SoC, [Online]. Available: https://www.altera.com/.
[95] S. Choi, R. Scrofano, V.K. Prasanna, J.-W. Jang, Energy-efficient signal processing using FPGAs, Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on Field Programmable Gate arrays - FPGA '03, New York, USA, (2003), p. 225, https://doi.org/10.1145/611817.611850.
[96] Altera, Achieving One TeraFLOPS with 28-nm FPGAs. Altera White Paper (2010).
[97] Lattice Semiconductor, [Online]. Available: http://www.latticesemi.com/.
[98] Microsemi | Semiconductor & System Solutions | Power Matters, [Online]. Available: https://www.microsemi.com/.
[99] A. Dutta, D. Saha, D. Grunwald, D. Sicker, CODIPHY, Proceedings of the Second Workshop on Software Radio Implementation Forum - SRIF '13, New York, USA, (2013), p. 1, https://doi.org/10.1145/2491246.2491247.
[100] W. Wolf, A decade of hardware/software codesign, Computer 36 (4) (2003) 38–43, https://doi.org/10.1109/MC.2003.1193227.
[101] G. De Micheli, R. Ernst, M. Wolf, Readings in Hardware/Software Co-Design, Morgan Kaufmann Publishers, 2002.
[102] J. Teich, Hardware/software codesign: the past, the present, and predicting the future, Proc. IEEE 100 (Special Centennial Issue) (2012) 1411–1430, https://doi.org/10.1109/JPROC.2011.2182009.
[103] Architecting a Smarter World Arm, [Online]. Available: https://www.arm.com/.
[104] S. Windh, X. Ma, R.J. Halstead, P. Budhkar, Z. Luna, O. Hussaini, W.A. Najjar, High-level language tools for reconfigurable computing, Proc. IEEE 103 (3) (2015), https://doi.org/10.1109/JPROC.2015.2399275.
[105] K.B. Chehida, M. Auguin, HW / SW partitioning approach for reconfigurable system design, Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, New York, USA, (2002), p. 247, https://doi.org/10.1145/581630.581670.
[106] M. López-Vallejo, J.C. López, On the hardware-software partitioning problem, ACM Trans. Des. Autom. Electron. Syst. 8 (3) (2003) 269–297.
[107] L. Zhuo, V.K. Prasanna, Hardware/software co-design for matrix computations on reconfigurable computing systems, 2007 IEEE International Parallel and Distributed Processing Symposium, (2007), pp. 1–10, https://doi.org/10.1109/IPDPS.2007.370268.
[108] G. Sapienza, I. Crnkovic, P. Potena, Architectural decisions for HW/SW partitioning based on multiple extra-functional properties, IEEE/IFIP Conference on Software Architecture, (2014), pp. 175–184, https://doi.org/10.1109/WICSA.2014.19.
[109] I. Bolsens, H. De Man, B. Lin, K. Van Rompaey, S. Vercauteren, D. Verkest, Hardware/software co-design of digital telecommunication systems, Proc. IEEE 85 (3) (1997) 391–418, https://doi.org/10.1109/5.558713.
[110] C. Zhang, Y. Ma, W. Luk, HW/SW partitioning algorithm targeting MPSOC with dynamic partial reconfigurable fabric, 14th International Conference on Computer-Aided Design and Computer Graphics (CAD/Graphics), (2015), pp. 240–241.
[111] D.K. Halim, S.W. Lee, M.S. Ng, Z.N. Lim, C.M. Tang, Exploring software-defined radio on multi-processor system-on-chip, 3rd International Conference on New Media (CONMEDIA), (2015), pp. 1–4, https://doi.org/10.1109/CONMEDIA.2015.7449149.
[112] G. Snider, P. Kuekes, R.S. Williams, CMOS-like logic in defective, nanoscale crossbars, Nanotechnology 15 (8) (2004) 881–891.
[113] M. Gobel, A. Elhossini, B. Juurlink, A methodology for predicting application-specific achievable memory bandwidth for HW/SW-Codesign, Euromicro Conference on Digital System Design (DSD), (2017), pp. 533–537.
[114] K. Underwood, Keith, FPGAs vs. CPUs: trends in peak floating-point performance, Proceeding of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays - FPGA '04, New York, USA, (2004), p. 171, https://doi.org/10.1145/968280.968305.
[115] B. Betkaoui, D.B. Thomas, W. Luk, Comparing performance and energy efficiency of FPGAs and GPUs for high productivity computing, International Conference on Field-Programmable Technology, (2010), pp. 94–101.
[116] B. Duan, W. Wang, X. Li, C. Zhang, P. Zhang, N. Sun, Floating-point mixed-radix FFT core generation for FPGA and comparison with GPU and CPU, International Conference on Field-Programmable Technology, (2011), pp. 1–6.
[117] M. Owaida, P. Ienne, G. Falcao, J. Andrade, C. Antonopoulos, N. Bellas, M. Purnaprajna, D. Novo, G. Karakonstantis, A. Burg, Enhancing design space exploration by extending CPU/GPU specifications onto FPGAs, ACM Trans. Embedded Comput. Syst. 14 (2) (2015) 1–23, https://doi.org/10.1145/2656207.
[118] X. Tian, K. Benkrid, High-performance quasi-Monte Carlo financial simulation: FPGA vs. GPP vs. GPU, ACM Trans. Reconfigurable Technol. Syst. 3 (4) (2010) 1–22, https://doi.org/10.1145/1862648.1862656.
[119] Q. Wu, Y. Ha, A. Kumar, S. Luo, A. Li, S. Mohamed, A heterogeneous platform with

GPU and FPGA for power efficient high performance computing, 2014 International Symposium on Integrated Circuits (ISIC), (2014), pp. 220–223, https://doi.org/10.1109/ISICIR.2014.7029447.
[120] H. Giefers, P. Staar, C. Bekas, C. Hagleitner, Analyzing the energy-efficiency of sparse matrix multiplication on heterogeneous systems: a comparative study of GPU, Xeon Phi and FPGA, 2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), (2016), pp. 46–56, https://doi.org/10.1109/ISPASS.2016.7482073.
[121] J.M.P. Cardoso, M. Weinhardt, High-level synthesis, FPGAs for Software Programmers, Springer International Publishing, Cham, 2016, pp. 23–47, https://doi.org/10.1007/978-3-319-26408-0_2.
[122] R. Tessier, K. Pocek, A. DeHon, Reconfigurable computing architectures, Proc. IEEE 103 (3) (2015) 332–354, https://doi.org/10.1109/JPROC.2014.2386883.
[123] J. Andrade, N. George, K. Karras, D. Novo, F. Pratas, L. Sousa, P. Ienne, G. Falcao, V. Silva, Design space exploration of LDPC decoders using high-level synthesis, IEEE Access (2017) 1, https://doi.org/10.1109/ACCESS.2017.2727221.
[124] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, T. Czajkowski, S.D. Brown, J.H. Anderson, LegUp: : an open-source high-level synthesis tool for FPGA-based processor/accelerator systems, ACM Trans. Embedded Comput. Syst. 13 (2) (2013) 1–27, https://doi.org/10.1145/2514740.
[125] J. Cong, Bin Liu, S. Neuendorffer, J. Noguera, K. Vissers, Zhiru Zhang, High-Level synthesis for FPGAs: from prototyping to deployment, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 30 (4) (2011) 473–491, https://doi.org/10.1109/TCAD.2011.2110592.
[126] Intel FPGA SDK for OpenCL. [Online]. Available: https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html.
[127] MaxCompiler | Maxeler Technologies. [Online]. Available: https://www.maxeler.com/products/software/maxcompiler/.
[128] G. Inggs, S. Fleming, D. Thomas, W. Luk, Is high level synthesis ready for business? A computational finance case study, International Conference on Field-Programmable Technology (FPT), (2014), pp. 12–19.
[129] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y.T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, K. Bertels, A survey and evaluation of FPGA high-Level synthesis tools, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 35 (10) (2016) 1591–1604.
[130] High-Level Synthesis For Any FPGA | LegUp Computing,. [Online]. Available: https://www.legupcomputing.com/.
[131] W. Meeus, K. Van Beeck, T. Goedemé, J. Meel, D. Stroobandt, An overview of today's high-level synthesis tools, Des. Autom. Embedded Syst. 16 (3) (2012) 31–51, https://doi.org/10.1007/s10617-012-9096-8.
[132] S. Ravi, M. Joseph, Open source HLS tools: a stepping stone for modern electronic CAD, IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), (2016), pp. 1–8.
[133] L.A. Tambara, J. Tonfat, A. Santos, F. Lima Kastensmidt, N.H. Medina, N. Added, V.A.P. Aguiar, F. Aguirre, M.A.G. Silveira, Analyzing reliability and performance trade-offs of HLS-Based designs in SRAM-Based FPGAs under soft errors, IEEE Trans. Nucl. Sci. 64 (2) (2017) 874–881, https://doi.org/10.1109/TNS.2017.2648978.
[134] Intel HLS Compiler, [Online]. Available: https://www.altera.com/products/design-software/high-level-design/intel-hls-compiler/overview.html.
[135] SDSoC Development Environment. [Online]. Available: https://www.xilinx.com/products/design-tools/software-zone/sdsoc.html.
[136] Stratus High-Level Synthesis - Cadence Design Systems. [Online]. Available: https://www.cadence.com/content/cadence-www/global/en_US/home/tools/digital-design-and-signoff/synthesis/stratus-high-level-synthesis.html.
[137] Synphony C Compiler - Synopsys. [Online]. Available: https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/synphony-c-compiler.html.
[138] MathWorks - Makers of MATLAB and Simulink. [Online]. Available: https://www.mathworks.com/.
[139] LabVIEW - National Instruments. [Online]. Available: http://www.ni.com/en-us/shop/labview.html.
[140] Simulink - Simulation and Model-Based Design. [Online]. Available: https://www.mathworks.com/products/simulink.html.
[141] MATLAB Coder. [Online]. Available: https://www.mathworks.com/products/matlab-coder.html.
[142] Simulink Coder - MATLAB & Simulink. [Online]. Available: https://www.mathworks.com/products/simulink-coder.html.
[143] Embedded Coder - MATLAB & Simulink. [Online]. Available: https://www.mathworks.com/products/embedded-coder.html.
[144] ZedBoard Zynq-7000 ARM/FPGA SoC Development Board. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/1-elhabt.html.
[145] R.W. Stewart, L. Crockett, D. Atkinson, K. Barlee, D. Crawford, I. Chalmers, M. Mclernon, E. Sozer, A low-cost desktop software defined radio design environment using MATLAB, simulink, and the RTL-SDR, IEEE Commun. Mag. 53 (9) (2015) 64–71, https://doi.org/10.1109/MCOM.2015.7263347.
[146] R.W. Stewart, K.W. Barlee, D.S.W. Atkinson, L.H. Crockett, Software Defined Radio using MATLAB & Simulink and the RTL-SDR, Strathclyde Academic Media, 2015.
[147] RTL-SDR (RTL2832U). [Online]. Available: https://www.rtl-sdr.com/.
[148] Blossom, Eric, GNU radio: tools for exploring the radio frequency spectrum, Linux J. 2004 (122) (2004) 4.
[149] Vivado System Generator for DSP. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado/integration/sysgen.html.
[150] J. Malsbury, Modular, open-source software transceiver for PHY/MAC research, Proceedings of the Second Workshop on Software Radio Implementation Forum - SRIF '13, (2013), https://doi.org/10.1145/2491246.2491255.
[151] M. Abirami, V. Hariharan, M.B. Sruthi, R. Gandhiraj, K.P. Soman, Exploiting GNU radio and USRP: an economical test bed for real time communication systems, 4th International Conference on Computing, Communications and Networking Technologies, ICCCNT 2013, (2013).
[152] K. Küçük, RTWiFi-Lab: a real-time wi-Fi laboratory platform on USRP and labview for wireless communications education and research, Computer Applications in Engineering Education (2017), https://doi.org/10.1002/cae.21865.
[153] V.P.G. Jimenez, A.L. Serrano, B.G. Guzman, A.G. Armada, Learning mobile communications standards through flexible software defined radio base stations, IEEE Commun. Mag. 55 (5) (2017) 116–123, https://doi.org/10.1109/MCOM.2017.1601219.
[154] Software Defined Radio - Lime Micro, [Online]. Available: http://www.limemicro.com/products/software-defined-radio/.
[155] G. Stewart, M. Gowda, G. Mainland, B. Radunovic, D. Vytiniotis, C.L. Agullo, Ziria, Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS, 50 (2015), pp. 415–428, https://doi.org/10.1145/2694344.2694368. New York, USA.
[156] P.D. Sutton, J. Lotze, H. Lahlou, S.A. Fahmy, K.E. Nolan, B. Özgül, T.W. Rondeau, J. Noguera, L.E. Doyle, Iris: an architecture for cognitive radio networking testbeds, IEEE Commun. Mag. 48 (9) (2010) 114–122, https://doi.org/10.1109/MCOM.2010.5560595.
[157] J. van de Belt, P.D. Sutton, L.E. Doyle, Accelerating software radio: Iris on the Zynq SoC, IFIP/IEEE 21st International Conference on Very Large Scale Integration (VLSI-SoC), (2013), pp. 294–295, https://doi.org/10.1109/VLSI-SoC.2013.6673295.
[158] J. Kim, S. Hyeon, S. Choi, Implementation of an SDR system using graphics processing unit, IEEE Commun. Mag. 48 (3) (2010) 156–162, https://doi.org/10.1109/MCOM.2010.5434388.
[159] S. Rajagopal, S. Rixner, J.R. Cavallaro, A programmable baseband processor design for software defined radios, The 2002 45th Midwest Symposium on Circuits and Systems, 2002. MWSCAS-2002. 3 (2002), pp. 413–416, https://doi.org/10.1109/MWSCAS.2002.1187061.
[160] B. Khailany, W.J. Dally, U.J. Kapasi, P. Mattson, J. Namkoong, J.D. Owens, B. Towles, A. Chang, S. Rixner, Imagine: media processing with streams, IEEE Micro 21 (2) (2001) 35–46, https://doi.org/10.1109/40.918001.
[161] B. Khailany, W.J. Dally, S. Rixner, U.J. Kapasi, J.D. Owens, B. Towles, Exploring the VLSI scalability of stream processors, The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings. (2003), pp. 153–164, https://doi.org/10.1109/HPCA.2003.1183534.
[162] J. Gummaraju, M. Rosenblum, Stream programming on general-purpose processors, 38th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'05), (2005), pp. 343–354, https://doi.org/10.1109/MICRO.2005.32.
[163] B.K. Khailany, T. Williams, J. Lin, E.P. Long, M. Rygh, D.W. Tovey, W.J. Dally, A programmable 512 GOPS stream processor for signal, image, and video processing, IEEE J. Solid-State Circuits 43 (1) (2008) 202–213, https://doi.org/10.1109/JSSC.2007.909331.
[164] M. Woh, Y. Lin, S. Seo, S. Mahlke, T. Mudge, C. Chakrabarti, R. Bruce, D. Kershaw, A. Reid, M. Wilder, K. Flautner, From SODA to scotch: the evolution of a wireless baseband processor, 2008 41st IEEE/ACM International Symposium on Microarchitecture, (2008), pp. 152–163, https://doi.org/10.1109/MICRO.2008.4771787.
[165] DigiKey Electronics - Electronic Components Distributor, [Online]. Available: https://www.digikey.com/.
[166] Newark element14 Electronics | Electronic Components Distributor, [Online]. Available: https://www.newark.com.
[167] BeagleBoard.org - community supported open hardware computers for making, [Online]. Available: http://beagleboard.org/.
[168] Imagination Technologies - Developing and Licensing IP cores, [Online]. Available: https://www.imgtec.com/.
[169] A.S. Fayez, Designing a Software Defined Radio to Run on a Heterogeneous Processor, Virginia Tech, 2011 Ph.D. thesis.
[170] B. Le, F.A. Rodriguez, Q. Chen, B.P. Li, F. Ge, M. ElNainay, T.W. Rondeau, C.W. Bostian, A public safety cognitive radio node, Proceedings of the 2007 SDR Forum Technical Conference, SDRF Google Scholar, (2007).
[171] M.M. Bezem, J.W. Klop, R. de. Vrijer, Terese (Group), Term Rewriting Systems, Cambridge University Press, 2003.
[172] 32-bit ARM Cortex-M3 PSoC 5LP | Cypress Semiconductor, [Online]. Available: http://www.cypress.com/products/32-bit-arm-cortex-m3-psoc-5lp.
[173] PSoC SDR PAøRWE, [Online]. Available: http://pa0rwe.nl/?page_id=32.
[174] B. Drozdenko, M. Zimmermann, T. Dao, K. Chowdhury, M. Leeser, Hardware-Software codesign of wireless transceivers on Zynq heterogeneous systems, IEEE Trans. Emerg. Top Comput. (2017) 1, https://doi.org/10.1109/TETC.2017.2651054.
[175] Z.X. Zhang Xin, T.L.-b. Tang Lin-bo, J.M.-p. Ji Mei-ping, Remote updating for DSP and FPGA programs, IET International Radar Conference 2015, Institution of Engineering and Technology, 2015, pp. 4.–4 ., https://doi.org/10.1049/cp.2015.1280.
[176] J. Hoffmeyer, Il-Pyung Park, M. Majmundar, S. Blust, Radio software download for commercial wireless reconfigurable devices, IEEE Commun. Mag. 42 (3) (2004) S26–S32, https://doi.org/10.1109/MCOM.2004.1273771.
[177] B. Bing, A fast and secure framework for over-the-air wireless software download using reconfigurable mobile devices, IEEE Commun. Mag. 44 (6) (2006) 58–63, https://doi.org/10.1109/MCOM.2006.1668420.
[178] L. Zhou, Q. Liu, B. Wang, P. Yang, X. Li, J. Zhang, Remote system update for system on programmable chip based on controller area network, Electronics 6 (2) (2017) 45, https://doi.org/10.3390/electronics6020045.

[179] A. Fernandes, R.C. Pereira, J. Sousa, P.F. Carvalho, M. Correia, A.P. Rodrigues, B.B. Carvalho, C.M.B.A. Correia, B. Goncalves, FPGA remote update for nuclear environments, IEEE Trans. Nucl. Sci. 63 (3) (2016) 1645–1649, https://doi.org/10.1109/TNS.2016.2559478.
[180] J. Vliegen, N. Mentens, I. Verbauwhede, Secure, remote, dynamic reconfiguration of FPGAs, ACM Trans. Reconfigurable Technol. Syst. 7 (4) (2014) 1–19, https://doi.org/10.1145/2629423.
[181] H. Kashyap, R. Chaves, Compact and on-the-Fly secure dynamic reconfiguration for volatile FPGAs, ACM Trans. Reconfigurable Technol. Syst. 9 (2) (2016) 1–22, https://doi.org/10.1145/2816822.
[182] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: enabling innovation in campus networks, ACM SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74.
[183] R. Enns, M. Bjorklund, J. Schoenwaelder, A. Bierman, Network configuration protocol (NETCONF), IETF RFC (2011).
[184] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, R. Boutaba, Network function virtualization: state-of-the-art and research challenges, IEEE Commun. Surv. Tutor. 18 (1) (2016) 236–262, https://doi.org/10.1109/COMST.2015.2477041.
[185] J. Gil Herrera, J.F. Botero, Resource allocation in NFV: a comprehensive survey, IEEE Trans. Netw. Serv. Manage. 13 (3) (2016) 518–532, https://doi.org/10.1109/TNSM.2016.2598420.
[186] S. Sun, M. Kadoch, L. Gong, B. Rong, Integrating network function virtualization with SDR and SDN for 4G/5G networks, IEEE Netw. 29 (3) (2015) 54–59.
[187] P. Shome, Muxi Yan, S.M. Najafabad, N. Mastronarde, A. Sprintson, CrossFlow: a cross-layer architecture for SDR using SDN principles, IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), (2015), pp. 37–39.
[188] J. Schulz-Zander, C. Mayer, B. Ciobotaru, S. Schmid, A. Feldmann, OpenSDWN: programmatic control over home and enterprise Wifi, Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, ACM, 2015, p. 16.
[189] A. Zubow, S. Zehl, A. Wolisz, BIGAP - seamless handover in high performance enterprise IEEE 802.11 networks, in IEEE/IFIP Network Operations and Management Symposium (NOMS), (2016), pp. 445–453.
[190] R. Riggio, M.K. Marina, J. Schulz-Zander, S. Kuklinski, T. Rasheed, Programming abstractions for software-defined wireless networks, IEEE Trans. Netw. Serv. Manage. 12 (2) (2015) 146–162.
[191] J. Vestin, A. Kassler, QoS enabled Wifi mac layer processing as an example of a NFV service, 1st IEEE Conference on Network Softwarization (NetSoft), (2015), pp. 1–9.
[192] W. Wang, Y. Chen, Q. Zhang, T. Jiang, A software-defined wireless networking enabled spectrum management architecture, IEEE Commun. Mag. 54 (1) (2016) 33–39.
[193] L.-m. Ang, K.P. Seng, L.W. Chew, L.S. Yeong, W.C. Chia, Wireless Multimedia Sensor Networks on Reconfigurable Hardware, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, https://doi.org/10.1007/978-3-642-38203-1.
[194] T. Han, N. Ansari, A traffic load balancing framework for software-Defined radio access networks powered by hybrid energy sources, IEEE/ACM Trans. Network. 24 (2) (2016) 1038–1051.
[195] T. Han, N. Ansari, On optimizing green energy utilization for cellular networks with hybrid energy supplies, IEEE Trans. Wirel. Commun. 12 (8) (2013) 3872–3882, https://doi.org/10.1109/TCOMM.2013.051313.121249.
[196] M.M. Tentzeris, A. Georgiadis, L. Roselli, Energy harvesting and scavenging [Scanning the Issue], Proc. IEEE 102 (11) (2014) 1644–1648, https://doi.org/10.1109/JPROC.2014.2361599.
[197] T. Han, N. Ansari, Powering mobile networks with green energy, IEEE Wirel. Commun. 21 (1) (2014) 90–96, https://doi.org/10.1109/MWC.2014.6757901.
[198] Ericsson Inc., Sustainable Energy Use in Mobile Communications Abstract, Technical Report, TechOnline, 2017.
[199] S. Park, H. Kim, D. Hong, Cognitive radio networks with energy harvesting, IEEE Trans. Wirel. Commun. 12 (3) (2013) 1386–1397, https://doi.org/10.1109/TWC.2013.012413.121009.
[200] F.K. Shaikh, S. Zeadally, E. Exposito, Enabling technologies for green internet of things, IEEE Syst. J. (2017), https://doi.org/10.1109/JSYST.2015.2415194.
[201] R. Arshad, S. Zahoor, M.A. Shah, A. Wahid, H. Yu, Green IoT: an investigation on energy saving practices for 2020 and beyond, IEEE Access 5 (2017) 15667–15681, https://doi.org/10.1109/ACCESS.2017.2686092.
[202] I. Mhadhbi, S. Ben Othman, S. Ben Saoud, An efficient technique for hardware/software partitioning process in codesign, Sci Program (2016) 1–11.
[203] J.M. Mitola, Software radios survey, critical evaluation and future directions, IEEE Aerosp. Electron. Syst. Mag. 8 (4) (1993) 25–36.
[204] W.H.W. Tuttlebee, Software-defined radio: facets of a developing technology, IEEE Pers. Commun. 6 (2) (1999) 38–44, https://doi.org/10.1109/98.760422.
[205] N. Nakajima, R. Kohno, S. Kubota, Research and developments of software-defined radio technologies in japan, IEEE Commun. Mag. 39 (8) (2001) 146–155.
[206] M. Dardaillon, K. Marquet, T. Risset, A. Scherrer, Software defined radio architecture survey for cognitive testbeds, 8th International Wireless Communications and Mobile Computing Conference, (2012), pp. 189–194.
[207] O. Anjum, T. Ahonen, F. Garzia, J. Nurmi, C. Brunelli, H. Berg, State of the art baseband DSP platforms for software defined radio: a survey, EURASIP J. Wirel. Commun. Netw. 2011 (1) (2011) 5, https://doi.org/10.1186/1687-1499-2011-5.
[208] M. Cummings, T. Cooklev, Tutorial: Software-defined radio technology, 25th International Conference on Computer Design, (2007), pp. 103–104, https://doi.org/10.1109/ICCD.2007.4601887.
[209] C. Moy, J. Palicot, Software radio: a catalyst for wireless innovation, IEEE Commun. Mag. 53 (9) (2015) 24–30, https://doi.org/10.1109/MCOM.2015.7263342.
[210] M. Palkovic, P. Raghavan, M. Li, A. Dejonghe, L. Van der Perre, F. Catthoor, Future software-defined radio platforms and mapping flows, IEEE Signal Process. Mag. 27 (2) (2010) 22–33, https://doi.org/10.1109/MSP.2009.935386.
[211] V. Derudder, B. Bougard, A. Couvreur, A. Dewilde, S. Dupont, L. Folens, L. Hollevoet, F. Naessens, D. Novo, P. Raghavan, T. Schuster, K. Stinkens, J.W. Weijers, L. Van der Perre, A 200 Mbps+ 2.14nJ/b digital baseband multi processor system-on-chip for SDRs, 2009 Symposium on VLSI Circuits, (2009), pp. 292–293.
[212] R.G. Machado, A.M. Wyglinski, Software-defined radio: bridging the analog-digital divide, Proc. IEEE 103 (3) (2015) 409–423.
[213] A.M. Wyglinski, D.P. Orofino, M.N. Ettus, T.W. Rondeau, Revolutionizing software defined radio: case studies in hardware, software, and education, IEEE Commun. Mag. 54 (1) (2016) 68–75, https://doi.org/10.1109/MCOM.2016.7378428.
[214] R. Farrell, M. Sanchez, G. Corley, Software-defined radio demonstrators: an example and future trends, Int. J. Digit. Multim. Broadcast. (2009) 1–12, https://doi.org/10.1155/2009/547650.
[215] T. Nesimoglu, A review of software defined radio enabling technologies, 2010 10th Mediterranean Microwave Symposium, (2010), pp. 87–90, https://doi.org/10.1109/MMW.2010.5605145.
[216] S. Singh, M. Adrat, G. Ulbricht, Special issue on increasing flexibility in wireless software defined radio systems, J. Signal Process. Syst. 89 (1) (2017) 81–83.
[217] G. Baldini, T. Sturman, A.R. Biswas, R. Leschhorn, G. Godor, M. Street, Security aspects in software defined radio and cognitive radio networks: A Survey and a way ahead, IEEE Commun. Surv. Tutor. 14 (2) (2012) 355–379, https://doi.org/10.1109/SURV.2011.032511.00097.