EURASIP Journal on Advances in Signal Processing
a SpringerOpen Journal

RESEARCH Open Access

A frame-based domain-specific language for rapid prototyping of FPGA-based software-defined radios

Ganda Stephane Ouedraogo, Matthieu Gautier* and Olivier Sentieys

Abstract

The field-programmable gate array (FPGA) technology is expected to play a key role in the development of software-defined radio (SDR) platforms. As this technology evolves, low-level designing methods for prototyping FPGA-based applications did not change throughout the decades. In the outstanding context of SDR, it is important to rapidly implement new waveforms to fulfill such a stringent flexibility paradigm. At the current time, different proposals have defined, through software-based approaches, some efficient methods to prototype SDR waveforms in a processor-based running environment. This paper describes a novel design flow for FPGA-based SDR applications. This flow relies upon high-level synthesis (HLS) principles and leverages the nascent HLS tools. Its entry point is a domain-specific language (DSL) which handles the complexity of programming an FPGA and integrates some SDR features so as to enable automatic waveform control generation from a data frame model. Two waveforms (IEEE 802.15.4 and IEEE 802.11a) have been designed and explored via this new methodology, and the results are highlighted in this paper.

Keywords: Software-defined radio (SDR); Field-programmable gate array (FPGA); Domain-specific language (DSL); High-level synthesis (HLS)

1 Introduction

Software-defined radio (SDR) is a flexible signal processing architecture with very high reconfiguration capabilities to adapt itself to various air interfaces. It was first introduced by Joseph Mitola and turned out to be a sustainable underlying structure for cognitive radio (CR) [1]. An important research and development work has been done to release SDR solutions, and a survey of SDR platforms is given in [2,3].

The mainstream approach to specify an SDR application has consisted in implementing the signal processing blocks on digital signal processors (DSP) coupled with hardware accelerators such as field-programmable gate array (FPGA) or application-specific integrated circuit (ASIC) fabrics [4]. As a matter of fact, the reason why such heterogeneous DSP-centric SDR platforms have been popularized is mainly due to the fact that current DSPs offer some important reconfiguration capabilities while being programmed with software design flows. However, DSPs suffer from high power consumption as compared to hardware fabrics. Thus, FPGA turned out to be an interesting alternative by trading-off between energy consumption and high computation performances. Dynamic reconfiguration capabilities are also available on recent FPGAs which offer some reconfiguration delays of up to a few microseconds. Furthermore, FPGA programming model enables to leverage an important dataflow parallelism through its native parallel computational model in contrast to the sequential nature of DSPs. Thus, FPGA-based SDR is a quite old paradigm [5,6], and most of the prototypes rely on intellectual properties (IP) described at the register-transfer level (RTL), therefore getting the SDR concept far away from its initial idea that is to say a software-only platform. Indeed, one of the issues related to FPGA-based SDR is the design flow, essentially based on hardware description languages (HDLs), which are used to program the applications. These languages, namely Verilog or VHSIC hardware description language (VHDL), represent an important burden and

*Correspondence: matthieu.gautier@irisa.fr
University of Rennes 1, IRISA, INRIA, 6 Rue de Kerampont, Lannion 22300, France

limit dramatically the programmability of the SDR platforms. In order tackle this issue, abstraction has been raised through software languages which were proposed to target FPGAs.

High-level languages (HLLs) are software languages that generate hardware description (RTL) from abstracted software specifications [7] for FPGA or ASIC targets. They raise the level of abstraction and bridge the gap between high-level algorithm designers and low-level circuit architecture designers. Their associated compiling frameworks generate RTL descriptions that fit the best to the specified application. We experienced with some of those high-level synthesis (HLS) tools and the flow depicted in this paper is essentially based on them.

In this paper, we are addressing the FPGA-based SDR and propose a methodology for rapid prototyping of the SDR applications. The main idea is to provide the SDR community with a design flow for specifying and implementing SDR waveforms fully running on FPGA-based platforms.

The major contribution of this paper is a design methodology consisting in a domain-specific language (DSL), which combines data frame information with dataflow computational model to synthesize an FPGA-SDR waveform. The aim is to hide the complexity of specifying an SDR waveform while automating all the control requirements from a high level data frame description. Furthermore, the HLS tools are employed to generate efficient signal processing blocks, at the RTL-level, while the compiling framework consistently builds the datapath and associates the control logic.

The rest of the paper is organized as follows. A discussion on the related work is given in Section 2. Subsequently, Section 2.3 states the problems which have motivated for this work. Section 3 details the methodology by first introducing the proposed DSL then describing each step of the language. Section 4 discusses the associated compiling framework, and Section 5 outlines some results. These results have been obtained on a Nutaq Perseus 6010 development board (Nutaq, Quebec, Canada) by prototyping both the PHY IEEE 802.15.4 and the PHY IEEE 802.11a. Finally, conclusions are drawn in Section 6.

2 Related work

2.1 Software-defined radio languages and middlewares

The main requirements in the SDR domain are essentially the reconfigurability of the underlying platform, the programmability of the platform, and the portability of the application over different platforms [1]. Therefore, several proposals attempted to meet those requirements by using software-based approaches [8-14]. Indeed, software gives an abstraction level that enables more control over the hardware design flow. To this end, two complementary approaches have been proposed, namely the SDR-specific languages to design the waveform and the SDR middleware to provide the building environment. They both take advantage of the abstraction level given by the software.

The first approach consists in defining specific languages, (i.e., DSLs), which purpose is to simplify the prototyping process of an SDR waveform, that is to say the physical layer (PHY). Most of these languages proposed in the literature essentially target DSP-centric architectures through specific design flows [8-10]. The waveform description language (WDL) [8] enables implementing the overall PHY from a hierarchical decomposition. It is mixing graphical and state machine concepts to provide the user with facilities to specify a complete SDR-PHY. Processing elements are blocks within which a state machine locally handles both scheduling, thanks to handshake protocols, as well as the communication with the other blocks. SPEX [9] is another language developed to specify SDR PHYs on single instruction multiple data (SIMD) processor using vector data type such as Matlab and also data type borrowed from SystemC. It is declined into three sub-languages, namely the Kernel SPEX defining the processing algorithm, the Stream SPEX handling the dataflow and modules interconnections, and finally the Synchronous SPEX for real-time constraints consideration. DiplodocusDF [10] is a modeling language that was proposed for implementing SDR PHYs on software-based platforms. It leverages a unified modeling language (UML)-like representation to model the SDR PHY and generates an executable to run on a software-based platform.

The second approach can be defined as a set of proposals for SDR standardization. They consist essentially in defining middlewares as the interfaces between the hardware and the application [11-14]. The core idea is to provide an environment, based on application programmable interfaces (APIs), to specify an SDR application while giving an emphasis to both the portability and the programmability of the application. The Software Communication Architecture (SCA) [11], initiated by the Joint Tactical Radio System (JTRS), is a major contribution in the software radio domain. It is defined as both a framework and a common standard for software radio specifications, and it is based on three major elements. The Core Framework handles via a hardware abstraction the installation, the configuration, the control, and the management of the waveforms. The ORB middleware serves to ensure the communication between the entities through client/server-like architecture. Both the Core Framework and the ORB leverage a real-time operating system to get access to the hardware resources. The Prismtech Spectra Core Framework (PrismTech Group Ltd., Stirlingshire, UK) [14] is a SCA-compliant framework that supports the deployment of waveform components on any mix of

general purpose processor (GPP), DSP microprocessors, and FPGAs. FPGA functions are essentially programmed in VHDL-RTL. In the same way, the Platform and Hardware Abstraction Layer (P-HAL) [12] aims at designing specific radio applications independently of the hardware context. The underlying approach consists in abstracting the hardware platform by software functional units. Thus, it manages radio process real-time constraints, processing elements and communication issues, and enables the software functional units to be configured. The P-HAL defines four services, namely the *BRIDGE* that handles real-time constraints, the *SYNC* that synchronizes the concurrent processes, the *KERNEL* that schedules the software functional units, and the *STATS* that analyses the statistics of the functional units.

The GNU Radio [13] provides both the signal processing primitives and the environment to implement software radio applications running exclusively on a host PC. The interconnections within the waveform are written in Python, and the signal processing blocks together with some of the critical datapath are implemented in C++. It is usually combined with an external RF chip to generate radio waveforms or simply used for simulation purpose.

In summary, it is important to note that managing an SDR from a higher level of specification is a key element since it would provide a complete overview over the application at the early stages of the design process. Most of the proposals that we have discussed in this section target either software-based platforms or heterogeneous platforms composed of mix of GPPs, DSPs, and FPGAs. FPGA uniprocessor platforms are a promising alternative for SDR. Indeed, some research work [15] have already addressed this issue by developing FPGA-based uniprocessor SDR platform where the signal processing is entirely done on an FPGA. However, such platforms lack of programmability since FPGAs are programmed with low-level languages. In this context, HLS turns out to be a good candidate to achieve such a high level of abstraction when FPGA-based SDR is addressed.

2.2 High-level synthesis tools and flow

A traditional implementation of a waveform intended to run on hardware processors such as FPGA or ASIC fabrics often requires a manual HDL description. Those HDLs appeared to be relatively fastidious, error prone, and hard to maintain when it comes to specify huge and complex applications. In the early 1980s, some new approaches/languages, most of them inspired from the C language, suggested a more abstracted way to specify and implement the hardware circuit architecture. Known as HLL, this trend is still on the mainstream, and an interesting survey was proposed in [7]. One could make an analogy with software programming flows which employ the C language as entry point instead of pure assembly code. However, employing such high-level design flow still requires a good knowledge in hardware circuit architecture so as to achieve good design performance.

The majority of HLLs are academic research works, but commercial examples of such languages/tools have also been released, namely Catapult from Calypto (Calypto Design Systems, Inc., San Jose, CA, USA) and Vivado HLS from Xilinx (Xilinx Inc., San Jose, CA, USA). Figure 1 illustrates the Catapult design flow that we have experienced within the work presented in [16]. The first stage requires specifying the waveform from an algorithmic point of view. This specification allows a functional simulation of the application and it is more about deciding what the system does and how the computation is done. At this stage, some relevant aspects such as data sizing and communication protocols can also be explored. Then, those realistic specifications are fed to the synthesis tool together with a set of architectural and resource constraints. The compiler parses the specifications and the constraints to decide both a scheduling and resource allocation for the final design. In a successful case, it generates the RTL description of the specification which can be tested and synthesized for a specific FPGA target or ASIC fabric. Recently, OpenCL (Khronos Group, Beaverton, OR, USA) [17-19] has also been proposed as an abstraction to program FPGAs. It is argued that OpenCL has a native approach to express application parallelism;

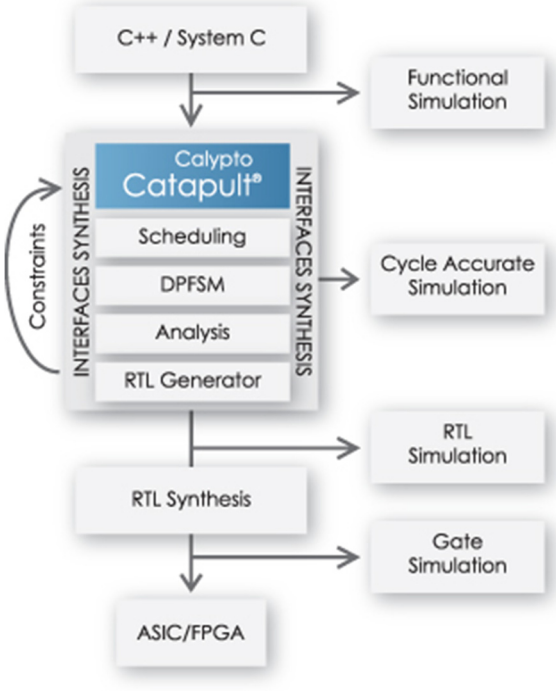

Figure 1 Catapult-C conception flow [16].

hence, it is a good candidate for designing parallel signal processing applications for FPGA fabrics.

These high-level languages and their compilers give an emphasis to the timing, area, or throughput constraints while making it easier to explore a set of solutions throughout design space exploration (DSE). Thus, HLS enables skipping several manual steps and gives an error-free path from abstract specification down to RTL description. In effect, by bypassing those steps, an important workload is being transferred to writing good specifications. A set of design optimization techniques such as loop pipelining or loop unrolling, which can be triggered from the specification, is also proposed by the HLS tools to achieve better design performance.

2.3 Problem statement and motivations

The FPGA platforms have not extensively been considered for the implementation of SDR because of the programming model which is offered by this technology. Table 1 summarizes the SDR languages that have been proposed to achieve the high-level programmability of a target platform, its reconfiguration, and the portability of the application over it. Most the proposals that we have introduced in Section 2.1 use FPGA as hardware accelerators using in most of the case RTL description. They do not address the implementation of the SDR waveform in a uniprocessor FPGA platform.

As mentioned previously, HLS is a powerful means to prototype and synthesize waveform specifications down to circuit architecture. However, HLS has been thus far specialized in datapath designing rather than control. Indeed, complex control structures might require to be written entirely in HDL. Actually, HLS has been employed as a processor generator intended to be used as hardware accelerators. For instance, it does not properly address the specification and the implementation of state machines that are the mainstream approach for specifying complex control system. Thus, in the context of full FPGA-based SDR, HLS can be leveraged to increase the programmability of each signal processing block composing the SDR PHY dataflow graph whereas control requirements could be handled separately. In addition to this, employing different HLS tools so as to achieve various performance in a target design is a situation in which the control structure could be handled separately, as well. In the following sections, the authors propose an SDR PHY design methodology combining the model-driven engineering (MDE) for high-level description and the HLS for datapath designing.

3 DSL-based SDR waveform specification

3.1 Model-driven engineering for FPGA-SDR

The growth of the platform complexity exhibits the limitations of current programming languages. Furthermore, these platforms evolve rapidly while the application codes are still written and maintained manually. A mainstream approach to handle such platform evolution is the MDE [20]. It comprises both a DSL, which formalizes the application structure, behavior, and requirements in a declarative way, and a set of transformation engines and generators to generate multiple artifacts such as source code. The MDE approach ensures a 'correct-by-construction' development of an application.

The spearhead of the proposed methodology is the definition of a DSL to implement SDR waveforms running on FPGA uniprocessor platforms. A DSL, as opposed to the general-purpose languages (GPLs), is a computer programming language of limited expressiveness focused on a particular domain [21,22]. In [21], they are declined into two variants, namely the internal DSL and the external DSL. Internal DSLs are languages that depend on a host language (generally a GPL). Their syntax is derived from the host language syntax and they benefit from the existing GPL compiling frameworks. An external DSL denotes a language with a custom syntax not depending on any GPLs. They are generally developed from scratch and require defining a specific compiling framework. Finally, the advantages of implementing a DSL are the improvement of the development productivity, the fact of facilitating the communication between domain expert through an explicit syntax, and above all, the usage of an alternative computational model.

Table 1 Summary of state-of-the-art SDR languages

Proposals	Programming language	Flexibility	Portability
WDL [8]	UML-based representation	Constrained specifications	n/a
SPEX [9]	Subset of the C++	n/a	DSP (VLIW and SIMD)
DiplodocusDF [10]	UML-based representation	Constraint profile	GPP and DSP
P-HAL [12]	Object-oriented C++	Real-time adaptation	GPP and DSP and FPGA
GNU Radio [13]	C++ and Python	Compile-time flexibility	GPP
Prismtech Spectra Core [14]	Model-based design and RTL IP cores	n/a	GPP and DSP and FPGA

n/a, not applicable.

The proposed methodology relies upon the MDE concept and deduces its computational model from a common feature of most of the telecommunication standards, that is, the data frame structure. The design flow synthesizes an efficient FPGA-based SDR waveform from both a specification of the data frame and the dataflow representation of the SDR waveform. These two features of the waveform are described in a novel external DSL. Figure 2 gives the generic DSL-based framework for an SDR-PHY description, which is divided into three related parts. The *Header* specifies different information that are later used both in the design flow and at compile time. Libraries of the required HLS-based functional blocks (FBs) are included together with the platform-specific information.

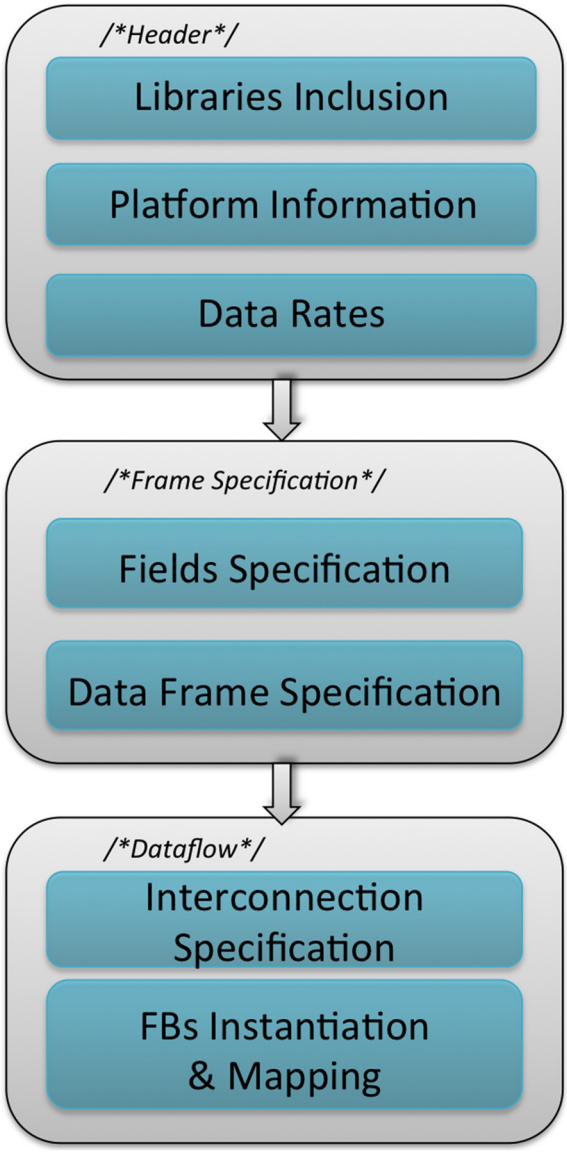

Figure 2 Framework of the PHY DSL-based description.

Clocks and data rates are also highlighted in this part. The *Frame specification* part highlights the data frame structure for a given PHY. Thus, each field is described and the complete frame is specified afterwards. In the *Dataflow* part, both the interconnections and FB mapping are specified. It provides a natural dataflow representation (graph) of the PHY. Further details of the DSL are provided in the following parts by taking the PHY IEEE 802.11a as a use case.

3.2 DSL-based frame specification

In most of the radio communication standards [23-25], the transmitted data are organized in data frames or packets. This structure ensures the interoperability of the solutions that are released by different vendors. A frame is composed out of a set of fields or subframes that carry either synchronization information or upper layer such as medium access control (MAC) data payload or frame-specific information.

Moreover, the information nested in a given field can vary or remain unchanged in all the transmitted frames. For instance, synchronization data should always respect a regular pattern, whereas data payload can vary in terms of content or size. Thus, in the proposed DSL, each field is characterized depending on its fix or variable nature. As a result, two types of fields are identified, namely the constant fields and the variable fields. Figure 3 illustrates a field-based data frame structure of a PHY IEEE 802.11a standard where the SHORTPB and LONGPB are constant fields, whereas HEADER and DATA fields are variable.

3.2.1 Field specification

In the DSL, the declarations of constant and variable fields are done with the key words *#fieldC* and *#fieldV*, respectively. They are followed by an arbitrary identifier that represents the field in the rest of the DSL source code. Following that, field-specific information such as data redundancy, size, and duration are defined within curly braces. These definitions enable to adapt the control when each field is transmitted or received.

Figure 4 shows a DSL-based source code description of the PHY IEEE 802.11a data frame previously shown in Figure 3. The PHY IEEE 802.11a frame is comprised of a short preamble (*SHORTPB*) appended for coarse synchronization. It is a known time domain data sequence used for signal detection, automatic gain control, and channel diversity selection. The short preamble is followed by a long preamble (*LONGPB*) field consisting of two orthogonal frequency-division multiplexing (OFDM) symbols used for channel and fine frequency estimation. The *HEADER* field carries frame-specific information such as the current modulation scheme and coding rate. The *DATA* field corresponds to the data from the upper layers to be transmitted.

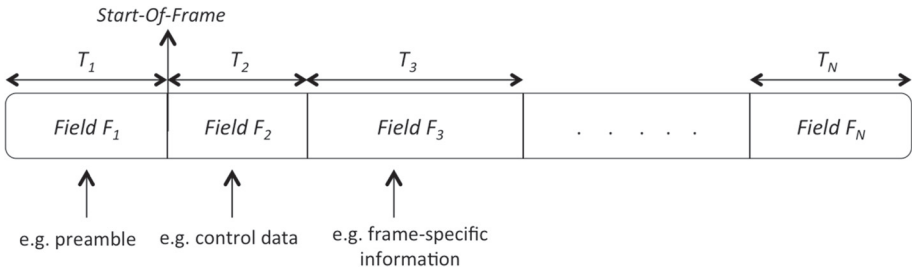

Figure 3 PHY IEEE 802.11a data frame and equivalent DSL declaration.

3.2.2 Data frame specification

A data frame is specified out of the instantiated fields with the key word *frame* followed by a frame identifier. Within curly braces, the set of fields composing the frame are listed. A frame can be either complex or real. Complex frames imply both an in-phase and a quadrature phase projection of the signal. Once a frame is specified, a *Start-Of-Frame* (*sof*) is designated among the constitutive fields. It denotes essentially a synchronization moment at the receiver. Thus, it enables to sketch the control of the waveform at the receiver.

The purpose of a frame declaration is twofold. On one hand, computation resources are optimized away in regard with the nature of the field. For instance, constant fields are one time computed and mapped to memory. The actual frame is built by consistently multiplexing those fields to the rest of the frame during transmission. On the other hand, the attributes of each field enable to dimension the appropriate control logic required in the waveform. This logic aims at sketching the datapath with energy efficiency considerations.

```
1   /*Short training field specification*/
2   #fieldC SHORTPB{
3       constant    ShTrainingSb: ./shpb.dat;
4       redundancy 10;
5       duration    8 us;
6       }
7   /*Long training field specification*/
8   #fieldC LONGPB{
9       constant    LgTrainingSb: ./lgpb.dat;
10      redundancy 2;
11      duration    8 us;
12      }
13  /*Header(SIGNAL) field definition*/
14  #fieldV HEADER{
15      data       hdpayloal;
16      size       3 bytes;
18      duration 4 us;
19      }
20  /*DATA field specification*/
21  #fieldV DATA{
22      data       dtpayload;
23      maxsize    2312 bytes;
24      minsize    0     bytes;
25      }
26  /*OFDM PPDU specification*/
28  complex frame PPDU{
29      SHORTPB, LONGPB, HEADER, DATA
30      } sof after SHORTPB
```

Figure 4 DSL-based IEEE 802.11a PHY frame specification.

3.3 DSL-based dataflow

3.3.1 The underlying model of computation

Addressing the dataflow computation of an SDR waveform requires to define an appropriate model of computation (MoC). Thus, an SDR waveform can be modeled as a synchronous dataflow (SDF) graph [26] as implemented in the Ptolemy project [27]. SDF models dataflow applications in a graph-like representation, and the usage of SDF as the underlying MoC makes our approach more formal and high-level than some of the existing approaches such as GNU Radio [13] where no MoC has been defined. SDF defines actors, interconnected by channels that are transporting tokens (data). The channels are buffer memories that model infinite first-in-first-outs (FIFOs). An example of SDF signal flow graph is given in Figure 5, where *FB1*, *FB2*, *FB3*, and *FB4* denote FBs communicating through channels. An SDF implementation requires a control unit, and a typical SDF-based dataflow architecture is given in Figure 6, where the FBs depict the computing cores. Figure 6 can be interpreted as an implementation of the graph given in Figure 5. At the inputs and outputs of the FBs, FIFOs are interfaced to control the streaming flow through the graph. An important element that is making the waveform structure consistent is the control logic which purpose is to orchestrate the data routing and computing in the graph. Indeed, it provides the system with specific signals which are used to change the state of the system. This generic overview of the dataflow architecture clearly shows that for efficiency purposes, enhancements can be performed at three distinct levels, namely the FBs, the communication infrastructure, and finally the control unit. In this work, the control unit is automatically inferred and generated from the DSL-based specification of the waveform by means of a DSL compiler.

3.3.2 DSL-based dataflow description

A relevant step in the proposed design methodology is the assembly of the FBs into a waveform architecture. This step is automatically performed by the DSL compiler that

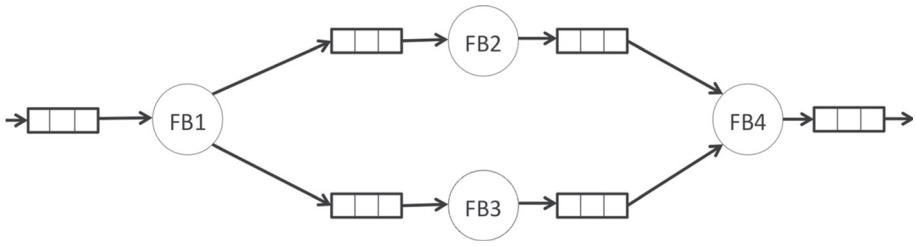

Figure 5 SDF signal flow graph example.

is further detailed in the next section. The resulting waveform is intended to process the data frame specified in the DSL source code. Moreover, each block has to be synthesized according to some set of throughput and latency constraints. To this end, the dataflow structure of the final waveform is inferred from the DSL-based description of its datapath. In Figure 7, a DSL-based description of an IEEE 802.11a PHY is depicted, where the FBs are sourced from HLS-based libraries written in C++. As it can be noted, the description of each FB is made by specifying the set of fields, with the key word *processing*, that the block is intended to process. In addition to mentioning the fields of interest, their respective source (analog-to-digital converter (ADC) or another FB) is also highlighted in that description. Inside braces, input, and output are connected to the external world by making explicit or not the rate at which they work. Inputs and outputs are declared with the key words *read* and *write*, respectively. Further details are made explicit for synthesis purpose. Thus, the expected synthesis tool and various constraints are given in the description.

The proposed design flow is here to synthesize SDR waveforms. To this end, aiming at anticipating on the flexibility requirements, when adaptive coding modulation is implemented for instance, a flexible FB is declared *adaptive* within the DSL source code. In Figure 7, both the fast Fourier transform (FFT) and the Demapper blocks are declared adaptive. Indeed, these two blocks work in multi-mode, and it is often required to reconfigure them on the fly in order to work in a given mode. Thus, block-level reconfiguration is addressed early in the design process. For a complete waveform to be synthesized, both the transmitter (TX) and the receiver (RX) have to be described.

A DSL compiler has been implemented to parse the entire waveform description and synthesize all the artifacts required to implement the waveform. This compiler is introduced and detailed in the next section.

4 DSL compiler

4.1 Compiling framework

The proposed DSL enables to capture, for a given waveform, several extra information like the PHY data frame structure or the adaptive nature of some of the FBs. This description is parsed and synthesized into the actual waveform intended to run on FPGA fabrics. The main interface language that is employed in this flow is the *.tcl* scripting language. Indeed, the *.tcl* language is proposed by most of the electronic design automation (EDA) tools as entry point. This scripting language is leveraged in the proposed design methodology to automate several steps throughout the synthesis process. HLS tools, for instance,

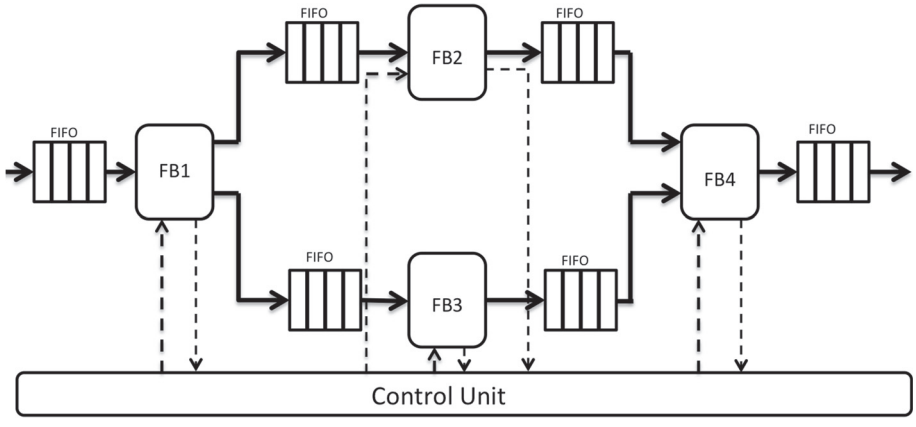

Figure 6 Typical SDF dataflow RTL architecture.

```
1   /*Cyclic Prefix Removal*/
2   cpremoval_i: ip CpRemoval processing LONGPB
3       HEADER DATA from ADC
4       {
5         read adc_data on port frame_in at fe;
6         read sync0    on port sync_data0;
7         read sync1    on port sync_data1;
8         write cpRemov on port frame_out at fe;
9         synthesis     #catapultc;
10        constraint    latency 1;
11      }
12  /*Coarse Time-Frequency Synchronization*/
13  ctfs_i: ip CoarseTimeFreqSync processing
14      SHORTPB from ADC
15      {
16        read adc_data on port frame_in at fe;
17        write sync0   on port sync_data0;
18        synthesis     #catapultc;
19      }
20  /*FFT instantiation*/
21  fft_i: adaptive ip FFT processing LONGPB
22      HEADER DATA from cpremoval_i
23      {
24        read cpRemov   on port fft_in  at fe;
25        write cplx_sbl on port fft_out at fs;
26        synthesis      #catapultc;
27        constraint     latency N;
28      }
29  /*Fine Time synchronization*/
30   fts_i: ip FineTimeSync processing LONGPB
31       from fft_i
32       {
33        read cplx_sb  on port cplx_fts at fe;
34        write sync1   on port sync_data1;
35        synthesis      #catapultc;
36       }
37  /*LongPreamble-based Channel Estimation*/
38  equa_i: ip ChannelPhaseTrack processing
39        LONGPB from fft_i
40        {
41        read  cplx_sb   on port fft_in_eq at fs;
42        write cplx_sb_eq on port eq_out   at fs;
43        synthesis         #catapultc;
44        }
45  /*De-mapping*/
46  demapper_i: adaptive ip DeMapper processing
47       HEADER DATA from fft_i
48       {
49        read cplx_sb_eq  on port cplx_in at fs;
50        write coded_bit  on port bit_out at fc;
51        synthesis         #catapultc;
52       }
```

Figure 7 DSL source code for a PHY IEEE 802.11a receiver.

are interfaced with *.tcl* scripts that are generated by the DSL compiler. Indeed, for each FB instantiated in the DSL, a *.tcl* script will be generated by the compiler. Afterwards, those scripts are used to synthesize the desired RTL description of the block. In addition, after generating the appropriate *.tcl* scripts, the DSL compiler collects the FB RTL descriptions and implements the datapath of the desired SDR waveform as a top level design. A controller is also inferred from the DSL-based description. It is implemented as a hierarchical finite state machine (HFSM) that is automatically generated after parsing the source code. The structure and the working modes of the finite state machine (FSM) are deduced from both an algorithm and the description provided by the DSL. Tcl scripting is also used to interface the Xilinx synthesis tools when it comes to synthesize the bitstream of the waveform. Figure 8 illustrates the compiling framework that is made up of four parts. The first part consists in modeling the waveform in the proposed DSL. Then, this description is parsed to generate *.tcl* scripts for each HLS-based FB. A heterogeneous approach is also envisioned in this methodology so as to interconnect FBs issued from different HLS tools. Thus, some key words such as *#catapult*, *#vivadohls*, and *#rtl* enable to target Catapult-C, Vivado HLS, and native RTL blocks, respectively. The second part is the core compiler where each script is used to synthesize (at the RTL

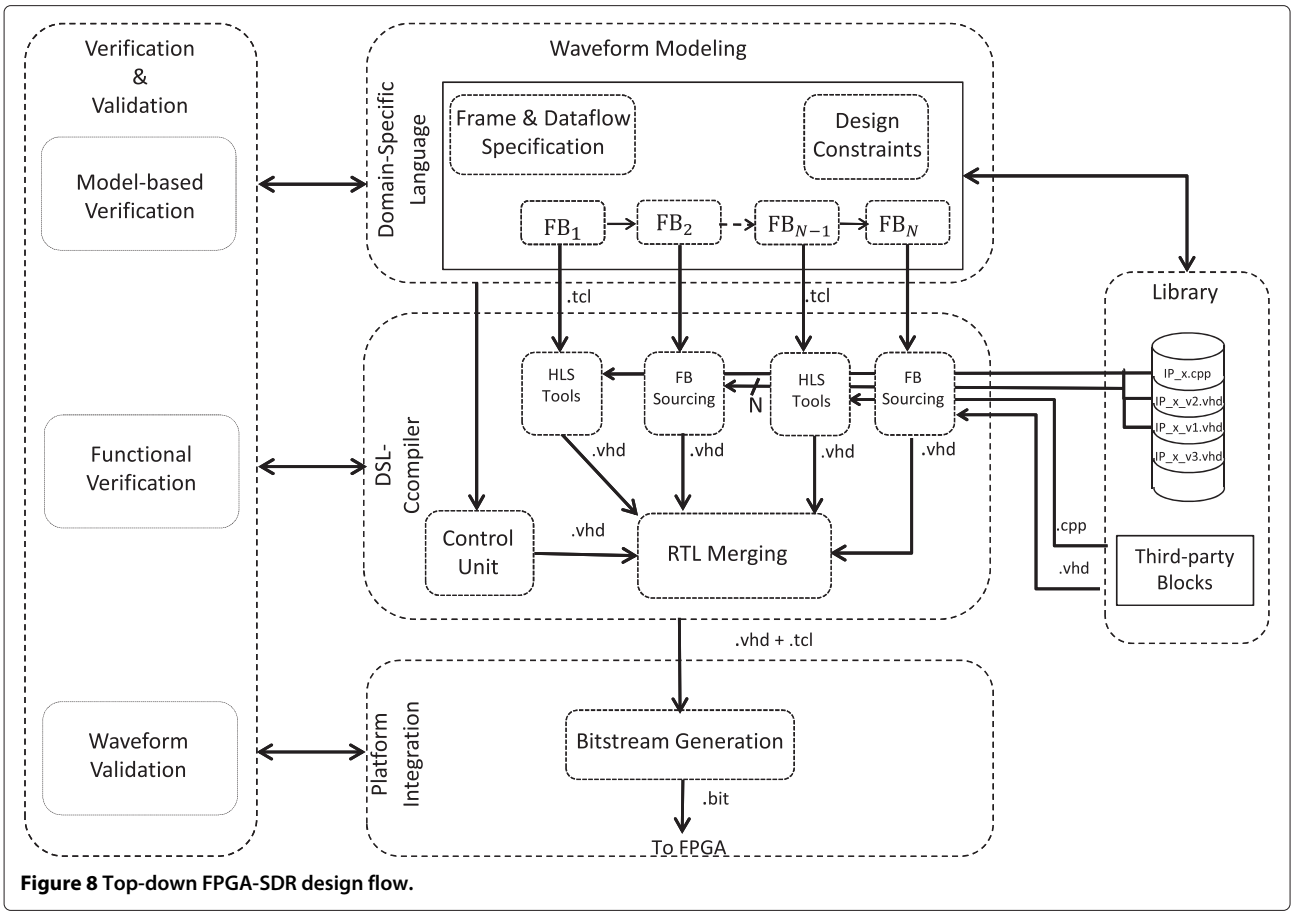

Figure 8 Top-down FPGA-SDR design flow.

level) the desired blocks while a control unit is automatically inferred and generated. The connection of all the FBs into a datapath is also performed automatically in this step together with the assembly of waveform which consists of the dataflow unit (datapath) and the control unit. The third part consists in the integration of the waveform into the platform with the help of the generated scripts. However, this step is not fully automated and requires the user to deal with the synthesis tools. The fourth stage operates in parallel with the others and deals essentially with verification and validation. In the following section, we discuss the validation aspects of the proposed design flow.

4.2 Verification and validation framework

Verification and validation (V&V) processes are very important in a design flow because they ensure that the final design meets with the requirements. They can be time consuming to be performed, but they enable to prove correctness and reliability in the various steps of design and implementation. In the proposed design as illustrated in Figure 8, the V&V is declined into three steps, namely the *Model-Based Verification*, the *Functional Verification*, and the *Waveform Validation*. The *Model-Based Verification* relies on the MDE concept and includes formal verification, dataflow analysis, and model checking. The *Functional Verification* relies first on the verification provided by the HLS tools which enables to verify each generated FB from a C/C++ test bench. The data frame specification is used to generate the stimuli for the test bench. The final waveform is verified from a generated VHDL test bench whose outputs are compared with predetermined *golden* outputs. Finally, the *Waveform Validation* takes place after bitstream synthesis. It consists essentially in testing the waveform on the platform so as to analyze its performance. Contrary to the first two steps that are performed automatically, this step is by now performed manually, and the authors are investigating on how to automate such validation process from the high-level specifications.

4.3 Frame-based SDR waveform controller

A frame is generally considered at distinct levels, namely the bit level, the symbol level, and optionally at the sample level. It is mainly characterized by its duration, its source (e.g., a FB), and its composition (a set of fields). This structure gathers exploitable information that are leveraged to achieve better waveform control performance. The duration of each field for instance can

help generate the read and write clock signals during the appropriate slot of time. In addition, each block within the flow graph is meant to perform a given action on a specific set of fields. Once this action terminates, the block may no longer be required, then disabled. For instance, some FBs may address only synchronization and some others address only data decoding. It is then convenient to control the activation and deactivation of each FB.

The automatically generated controller is a hierarchical FSM working in both TX and RX modes. Its overall structure is given in Figure 9, where the dash lines denote parallel states. In the TX mode, the controller consists of two major states called *super-states*. First is the *IDLE* super-state corresponding to the inactive state of the transmitter. After detecting a start signal from the MAC layer for example, it switches from the *IDLE* to the *FRAMING* super-state where data coding is sketched. Generally speaking, dataflow transmitters are feed-forward architectures then less complex to implement as compared to their associated receiver. The *FRAMING* super-state is declined into three parallel sub-states, namely the *CODING* state, the *INSERT* state, and finally the *BL-RECONF* state. In the *CODING*, the dataflow computation described in the DSL is performed. The output samples are fed to the digital-to-analog converters (DACs) prior to RF stage. In parallel to the *CODING* state, an *INSERT* state is active. This state manages the insertion of specific data in the frame, both at the time and frequency domains. Considering the standards using

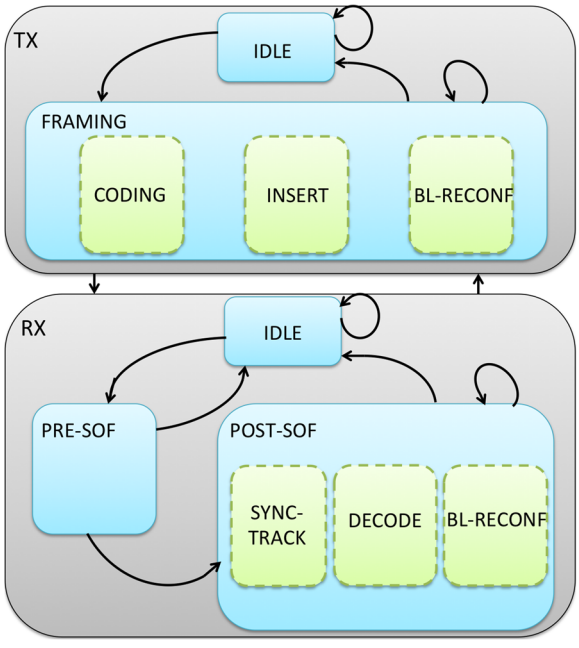

Figure 9 Transceiver hierarchical FSM.

an OFDM modulation, they require to inject pilot symbols for coherent detection. Moreover, a data frame often includes constant fields (e.g., preamble) that remain the same in all the transmitted frames. For the purpose of reducing the overall computation, such fields are one-time computed and inserted (at run-time) at the sample level in the frame before DACs. The *BL-RECONF* state handles the block-level (fine-grained) reconfiguration of the transmitter. Modern standards require certain blocks to be adaptive (ACM), i.e., changing their properties on-the-fly. One approach consists in hard coding all the configuration of the block once, then using software controlled switch to select the desired configuration at run-time. A second approach is to reconfigure the block when a given configuration is desired. It is a suitable approach which fits the best to the paradigm of SDR and would require partial reconfiguration capabilities in the context of FPGA.

In the RX mode, the FSM is composed out of the three super-states. An *IDLE* state, as in TX mode, denotes the inactive state of the receiver. In this state, the receiver monitors the environment seeking for an incoming signal. Once a signal is detected, through an received signal strength indicator (RSSI) for instance, the receiver switches from the *IDLE* state to the *PRE-SOF* state. The *PRE-SOF* state consists essentially of synchronization tasks as imposed by most of the standards. Once the system enters the *PRE-SOF* state, a set of synchronization elements has to be detected and computed within a certain delay. If not, the system returns in the *IDLE* state. These synchronization element detection and computation are associated to the *sof* event which is defined in the DSL-based frame specification that was lately introduced. An *sof* detection makes the system switch from *PRE-SOF* to *POST-SOF* where a coherent data decoding is sketched. The *POST-SOF* state is declined into three parallel sub-states, namely the *DECODING* state where most of the signal processing is required, the *SYNC-TRACK* state in which the system keeps on tracking synchronization elements, and finally the *BL-RECONF* state to handle the run-time block-level reconfiguration like in TX mode.

In each of the states (TX and RX), a set of dataflow computation is intended. In the context of FPGA, the datapath is one time mapped to dedicated resources and ready to operate as soon as the FPGA is powered on. One of the roles of the controller is to distribute the clock signal to activate or deactivate the FBs when required. We leverage the properties of the data frame together with the intrinsic structure of the datapath (dataflow) to build the control unit. First, a data frame F is as a collection of fields, i.e.,

$$F = \cup_{i=1}^{N} F_i, \tag{1}$$

where F_i denotes the ith field. Each field F_i is characterized by its duration T_i, its *constant*, or *variable* nature *State* of its transported data *Payload*:

$$F_i = \{T_i, \text{State}, \text{Payload}\}. \tag{2}$$

The duration T_F of the overall frame F is computed as,

$$T_F = \sum_{i=1}^{N} T_i. \tag{3}$$

The datapath, on both the transmitter and receiver side, is a set of interconnected FBs as illustrated in Figure 6. They are characterized by their latency (L), throughput (TP), and input and output data rates (f_{in} and f_{out}). For each block, when the inputs and outputs rate are known, they can be directly managed using enable signals. Moreover, each block is activated or deactivated on a per-field basis since computation happens to be specific to a given field. Thus, let FB_j be the jth FB within the dataflow graph:

$$\text{FB}_j = \left\{f_{\text{in}_j}, f_{\text{out}_j}, L_j, \text{TP}_j\right\}. \tag{4}$$

Assuming that FB_j processes the field F_i of duration T_i at an input rate of f_{in} and output rate of f_{out}, the block requires being enabled a number of clock cycles equal to

$$k_{i,j} = T_i f_{\text{in}}. \tag{5}$$

Thus, each block is affected a time slot to process a field when this field is traversing the graph. They are then activated depending on the ongoing field. To achieve this, the controller decides a starting moment for each block in the graph. This starting time is computed by considering both the graph structure and the properties (latency and throughput) of each block composing it. Indeed, each state is associated to a datapath, and once the system enters a state, the processing starts with a specific block that is tagged as a reference block. The activation moment of the remaining blocks in the datapath is then estimated by computing their distance to the reference block based on the latency and the throughput of the blocks preceding them. This algorithm has been integrated into the presented SDR PHYs design flow.

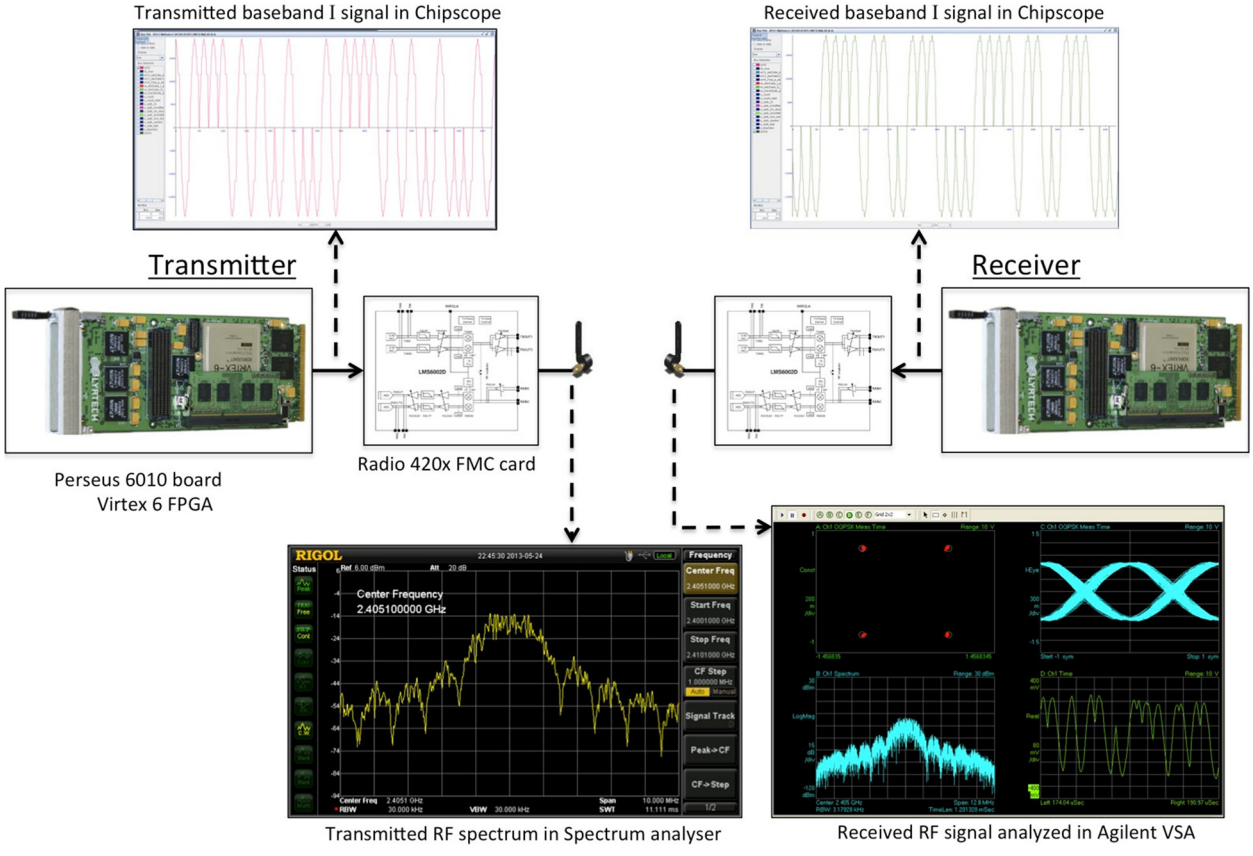

Figure 10 **Nutaq boards and signal analysis snapshots.**

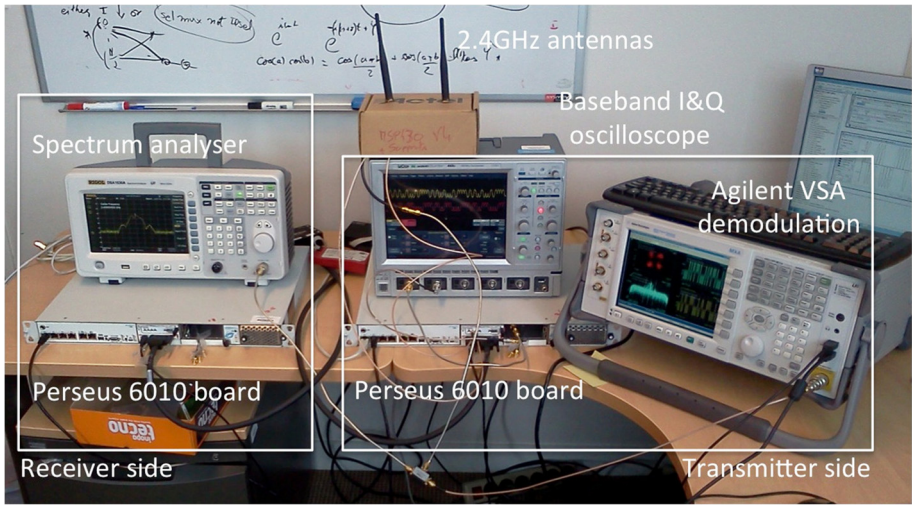

Figure 11 Experimental testbed.

5 Experimentation and validation

5.1 Testbed description and waveform implementation results

The experimentation of the proposed DSL-based design methodology was carried out on an FPGA-based platform. It consisted in describing and automatically implementing both IEEE 802.15.4 and IEEE 802.11a transceivers.

The testbed is composed of a Nutaq (ex-Lyrtech) Perseus 6010 development board. Perseus 6010 is an advanced mezzanine card designed around a Virtex-6 FPGA with fabrics flexibility and an external memory. It also benefits from multiple high-pin-count add-on FPGA mezzanine card (FMC)-based cards. An FMC-based Radio 420x daughter board is used as full duplex SDR agile RF front-end with 12 bits ADC and DAC at up to 40 mega-samples per second.

The integration of the applications is partly made by using generated *.tcl* scripts as entry point to the synthesis tools. Moreover, in order to demonstrate the feasibility of the concept, a first test has been made without the DSL on the same platform and the results were published in [28].

In Figure 10, the testbed is presented. Figure 11 on the other hand, illustrates the set of equipment utilized in this experimentation. It includes two Nutaq platforms (TX and RX) and some measurement tools such as ADS

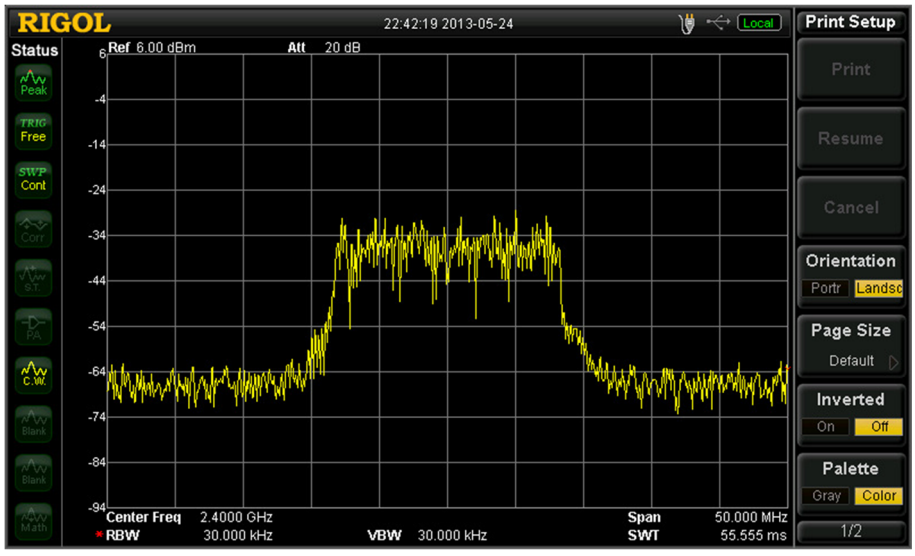

Figure 12 IEEE 802.11a transmitted spectrum.

Table 2 Resource estimation for the IEEE 802.15.4 and IEEE 802.11a receivers

	Slices	FF	LUT	DSP	BRAM
IEEE 802.15.4	543	1630	1058	1	0
IEEE 802.11a	961	803	2832	8	5

vector signal analyzer (VSA), spectrum analyzer, and timing scopes. The PHY IEEE 802.15.4 transceiver was first prototyped, and the corresponding baseband signals are also shown in Figure 10 through a Chipscope capture. The spectrum corresponding to transmitted signal is observed using a spectrum analyzer, and VSA Agilent (Agilent Technologies, Santa Clara, CA, USA) was also used to demodulate the transmitted signal, thus validating the IEEE 802.15.4 waveform. Moreover, we have developed an OFDM transceiver whose corresponding transmitted signal spectrum is given in Figure 12.

5.2 DSL complexity

A waveform specification with the proposed DSL is done with a few hundreds of lines of source code. In the examples depicted in this paper, less than 150 of lines of source code was used to model the PHY IEEE 802.11a and less than a hundred for the PHY IEEE 802.15.4. The flow also relies on a set of constantly evolving FB libraries, hence reducing considerably the development time. As previously explained, most of the steps in this flow are made automatic through the *.tcl* scripting language. The DSL compiler handles the implementation of the waveform by connecting the FBs and generating the waveform controller introduced previously in this paper. The resulting RTL source code for a given waveform consists of a few thousand of lines of source code entirely generated by both the DSL compiler and the HLS tools.

5.3 Synthesis results and design space exploration

Table 2 gives the synthesis results for the two waveform receivers. The results are collected after place and route and are intended to show that the proposed flow allows to complete the whole design process. Resource optimization can be performed from a higher level through the HLS tools. Indeed, this flow also enables automatic DSE for a given FB. The exploration aims essentially at meeting performance requirements and represents an important add value to the HLS tools. They make it possible within a few clicks to explore various solutions of the same design. In Figure 13, a DSE is performed on a *Decoder* block, part of the PHY IEEE 802.15.4, in order to trade off between the area and the throughput of the block. Each curve on this figure corresponds to a level of internal loop pipelining which represents how often, in clock cycles, a loop iteration is started. Thus, the less the initiation interval (II), is the deeper the pipeline is. One can see from these curves that low throughput in cycles (high in frequency) is achieved when the design is pipelined to a maximum (II $=$ 1). Loop unrolling (U) impacts considerably the design throughput. However, both loop unrolling and pipelining require more resources; hence, an increasing area as can be seen in Figure 14 shows a DSE for a 256-point FFT, performed with Catapult-C in order to obtain different solutions for same FB. The DSE is performed automatically from the scripts generated by the DSL compiler. The exploration of the FFT block exhibits

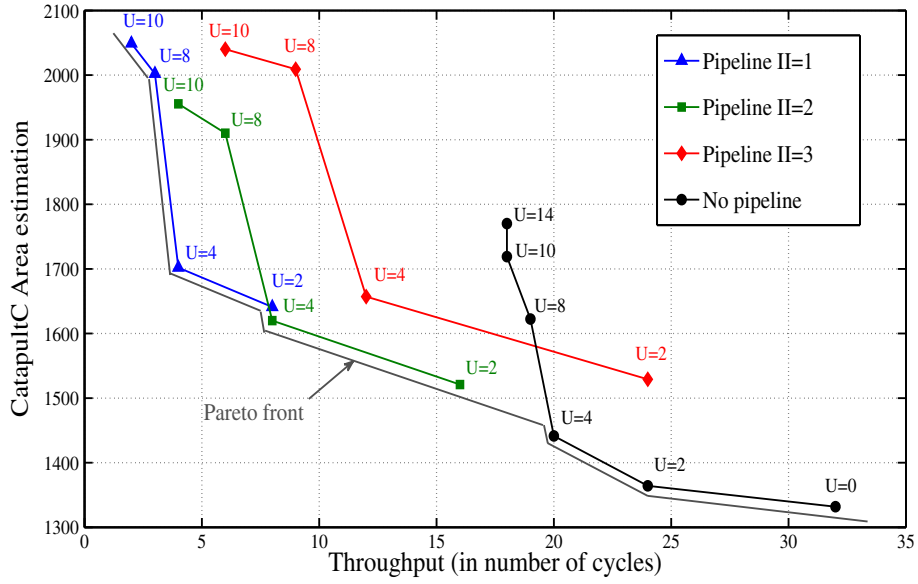

Figure 13 Design space exploration on PHY IEEE 802.15.4 *Decoder* block.

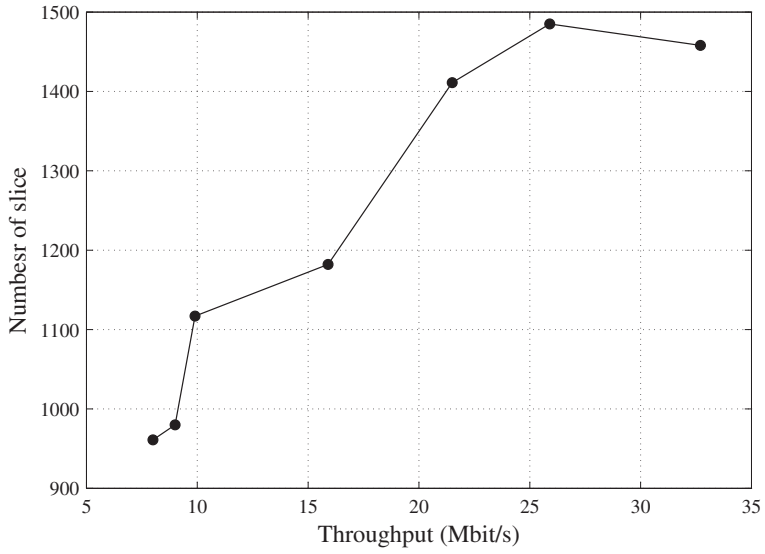

Figure 14 Resource estimation versus data rate of the IEEE 802.11a receiver.

an increasing number of slices versus the achievable data rate. Hence, suitable solution can be chosen to compose the desired waveform.

To conclude, the current version of the flow handles block-level configuration; it enables to select the architecture of a *adaptive* block depending on the constraints. This compile-time reconfiguration takes advantage of the HLS capabilities. An extension would be to make the generated controller able to manage at run-time the handover between two configurations of an *adaptive* block. In practice, it could leverage the dynamic and partial reconfiguration features available on recent FPGA devices. To this end, each *adaptive* block could be interfaced with large memory resources to store the streaming data when a reconfiguration is required.

6 Conclusions

In this paper, we have discussed a design flow which purpose is to enable rapid implementation of SDR applications on FPGA-based platforms. The proposed flow relies upon a DSL which provides a software abstraction to model the waveform. The specified waveform generation results from the analysis of their associated data frame and dataflow structure. Moreover, HLS capabilities are also featured with this flow, aiming at shortening design implementation time.

A case study has been performed through two examples, namely a PHY IEEE 802.11a transceiver and a PHY IEEE 802.15.4 transceiver. It results in a fast method for prototyping SDRs on FPGA devices with considerable flexibility to achieve different design goals. The current perspectives aim at enriching the processing block libraries and integrate different HLS tools within the flow. Another aspect that we will investigate is the use of dynamic and partial reconfiguration features on FPGA which are now missing in the depicted flow.

Competing interests
The authors declare that they have no competing interests.

Acknowledgements
The authors would like to thank the anonymous reviewers for their comments and suggestions which helped improve this paper.

Received: 11 March 2014 Accepted: 20 October 2014
Published: 18 November 2014

References

1. J Mitola, Software radios: survey, critical evaluation and future directions. IEEE Aerospace Electron. Syst. Mag. **8**(4), 25–36 (1993)
2. M Dardaillon, K Marquet, T Risset, A Scherrer, in *IEEE International Wireless Communications and Mobile Computing Conference (IWCMC)*. Software defined radio architecture survey for cognitive testbeds (Limassol, Cyprus, 27–31 Aug 2012), pp. 189–194
3. 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. Wireless Commun. Netw. **2011**, 5 (2011)
4. JF Jondral, Software-defined radio: basics and evolution to cognitive radio. EURASIP J. Wireless Commun. Netw. **2005**, 275–283 (2005)
5. M Cummings, S Haruyama, FPGA in the software radio. IEEE Commun. Mag. **37**(2), 108–112 (2010)
6. AD Stefano, G Fiscelli, CG Giaconia, An FPGA-based software defined radio platform for the 2.4 GHz ISM band. Res. Microelectronics Electron. 73–76 (2006). doi:10.1109/RME.2006.1689899
7. SA Edwards, The challenges of synthesizing hardware from C-like languages. IEEE Des. Test Comput. **23**(5), 375–386 (2006)
8. ED Willink, The waveform description language: moving from implementation to specification. IEEE Mil. Commun. Conf. (MILCOM 2001) **1**, 208–212 (2004)
9. Y Lin, R Mullenix, M Woh, S Mahlke, T Mudge, A Reid, K Flautner, in *Software Defined Radio Technical Conference and Product Exposition (SDR-Forum 06)*. SPEX: a programming language for software defined radio (Orlando, FL, USA, 13–16 Nov 2006)
10. J Gonzales-Pina, R Ameur-Boulifa, R Pacalet, in *38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*.

DiplodocusDF, a domain-specific modeling language for software defined radio applications (Cesme, Izmir, Turkey, 5–8 Sept 2012), pp. 1–8
11. G Jianxin, Y Xiaohui, G Jun, L Quan, in *IEEE Conference on Computational Intelligence and Industrial Applications (PACIIA)*. The software communication architecture specification: evolution and trends (Wuhan, China, 28–29 Nov 2009)
12. A Gelonch, X Revès, V Marojevik, R Ferrús, in *SDR Forum Technical Conference*. P-HAL: a middleware for SDR applications (Orange Country, CA, USA, 14–17 Nov 2005)
13. GNU Radio, The free and open software radio ecosystem. www.gnuradio.org
14. PrismTech Corporation, *Prismtech spectra SDR: spectra CF high performance low footprint SCA core framework*. (PrismTech Corp., France, 2014)
15. The WARP Project. http://warp.rice.edu
16. S McCloud, *Catapult-C, synthesis-based design flow: speeding implementation and increasing flexibility*. (White Paper, Mentor Graphics, 2004)
17. Altera, *Implementing FPGA design with the OpenCL Standard*. (White Paper, Altera Corporation, 2013)
18. K Shagrithaya, K Kepa, P Athanas, in *IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*. Enabling development of OpenCL applications on FPGA platforms (Washington, D.C., USA, 5–7 June 2013), pp. 26–30
19. G Economakos, in *16th Panhellenic Conference on Informatics*. ESL as a gateway from OpenCL to FPGAs: basic ideas and methodology evaluation (Piraeus Attica, Greece, 5–7 Oct 2012), pp. 80–85
20. DC Schmidt, Model-driven engineering. IEEE Comput. **39**(2), 25–31 (2006)
21. M Fowler, R Parsons, *The Addison-Wesley Signature Series: Domain-Specific Languages*. (Pearson, Indianapolis, 2011)
22. A Pasha, S Derrien, O Sentieys, System level synthesis for wireless sensor node controllers: a complete design flow. ACM Trans. Des. Automation Electron. Syst. (TODAES) **17**(1), 2–1224 (2011)
23. IEEE, *IEEE Std 802.15.4: IEEE Standard for Information Technology: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANS)*. (IEEE, Piscataway, 2006)
24. IEEE, *IEEE Standard for Local and Metropolitan Area Networks: Part 16: Air Interface for Fixed Broadband Wireless Access Systems*. (IEEE, Piscataway, 1999)
25. IEEE, *Supplement to IEEE Standard for Information Technology: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. (IEEE, Piscataway, 2001)
26. EA Lee, DG Messerschmitt, Synchronous data flow. Proc. IEEE **31**(1), 24–35 (1987)
27. J Eker, J Janneck, EA Lee, J Liu, X Liu, J Ludvig, S Sachs, Y Xiong, Taming heterogeneity - the Ptolemy approach. Proc. IEEE **91**(1), 127–144 (2003)
28. V Bahtnagar, GS Ouedraogo, M Gautier, A Carer, O Sentieys, in *IEEE Vehicular Technology Conference (VTC-Spring13)*. An FPGA software defined radio with a high-level synthesis flow (Dresden, Germany, 2–5 June 2013), pp. 1–5

doi:10.1186/1687-6180-2014-164
Cite this article as: Ouedraogo *et al.*: A frame-based domain-specific language for rapid prototyping of FPGA-based software-defined radios. *EURASIP Journal on Advances in Signal Processing* 2014 **2014**:164.