A WORKBENCH FOR WAVEFORM DESCRIPTION BASED SDR IMPLEMENTATION

T. Kempf, E. M. Witte, V. Ramakrishnan, G. Ascheid
Institute for Integrated Signal Processing Systems,
RWTH Aachen University, Germany
kempf@iss.rwth-aachen.de

M. Adrat, M. Antweiler
Research Establishment for Applied Science (FGAN),
Dept. FKIE/KOM, Wachtberg, Germany,
adrat@fgan.de

ABSTRACT

The use of multiple wireless communication standards in today's communication networks opens completely new opportunities for future wireless communication, e.g. cognitive radios. However, current support for different standards requires implementation of redundant hardware within one handset. To cope with this issue, research in the field of Software Defined Radios (SDRs) aims at implementing the waveforms in software allowing reuse of shared hardware resources. Development of such SDRs is a challenging task as both software and hardware have to be developed under tough constraints of real-time processing, architecture and energy efficiency. To enable the three key properties of portability, interoperability and loadability, a concept has been proposed for a seamless SDR design flow in [1]. Key element of this concept is the orthogonalization of the waveform description, the mapping to an arbitrary hardware platform and the platform design. In this paper, a realization of the concept is introduced. The efficient design of the waveform application, hardware platform and application-to-architecture mapping enables fast exploration cycles during SDR development. The (semi-)automatic generation of the software and hardware implementations (on the basis of IP-blocks) enables designers to quickly verify, debug and test their design decisions in simulation models and on a prototype.

I. INTRODUCTION

Support of multiple wireless communication standards requires multiple dedicated hardware transceivers resulting in a significant hardware overhead. The concept of Software Defined Radios (SDRs) offers a method to cope with this issue by providing flexibility to support updates and/or new standards in the field. The design of such an SDR is a highly complex and challenging task, as the three key properties of *portability*, *interoperability* and *loadability* have to be fulfilled. The JTRS (*Joint Tactical Radio System*) program of the US DoD has defined the *Software Communications Architecture* (SCA) [2] to achieve these key properties. However, JTRS assumes that the waveform functionality is provided in a way, which can directly be applied to any SDR hardware platform. Constraints have neither been defined for the description of waveforms nor for the generation of SW- and HW-code applicable to a specific SDR hardware platform. The SCA makes it easier to achieve portability, interoperability and loadability, but does not guarantee either of these properties.

In our previous works [1] and [3] a concept has been proposed for a seamless SDR design flow from a waveform description to the implementation of the waveform onto an arbitrary SDR hardware platform. The key element of this concept is the orthogonalization of the development of waveform application, hardware platform and application-to-architecture mapping. Portability is enabled by a separate design of the SDR application and HW platform. As the waveform implementation running on different SDR HW platforms is based on a unique waveform application, an important step towards interoperability of different SDRs is already done. The key property of loadability is achieved by loading software executables on a SDR in the field. These software executables can be generated with the concept's waveform application and mapping. In this paper we highlight a realization of the proposed concept by a workbench prototype allowing fast and efficient SDR development.

This paper is structured as follows. In Section II an overview of the proposed concept for a seamless SDR development is given. The main contribution of this paper, a workbench prototype, is described in Section III. Finally, a brief case study is presented to highlight the capabilities of this workbench.

II. CONCEPT FOR SEAMLESS WAVEFORM DEVELOPMENT

In [1] and [3] a concept for a seamless SDR design flow has been proposed, starting at the waveform description and going down to the implementation on a SDR hardware platform. In contrast to formerly known approaches to *Waveform Development Environments* (WDE) ([4–6]) it is

This research project was performed under contract with the Technical Center for Information Technology and Electronics (WTD-81), Germany

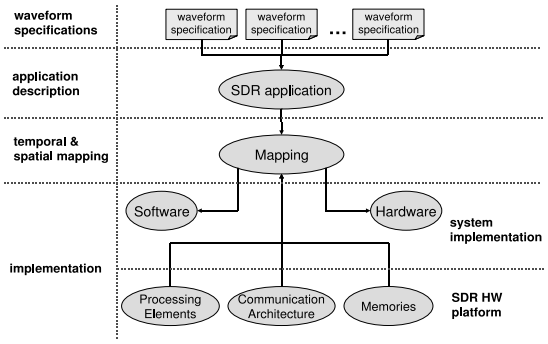


Figure 1: Seamless SDR design flow

neither limited to a single aspect of the overall design environment (e.g., the non-ambiguous waveform description itself) nor to a specific tool (like Matlab/Simulink for the (semi)-automatic code generation). The proposed concept's four key aspects, as depicted in Figure 1, are the following:

A. Description of the Waveform Application:
The waveform application, which is typically given as a textual document, is represented in a non-ambiguous description. In [1] an efficient description has been proposed, based on general programming language structures like sequential statements or parallel tasks (existing in C/C++ and VHDL/Verilog). *Tasks* (blocks/nodes) represent the subfunctionality of a waveform application. *Edges* between tasks characterize the data exchange, separating the functionality and communication. Abstracting communication allows faster and simpler development, as developers do not have to consider the underlying communication scheme, like FIFO implementations. Finally, the application is summarized in a *topology graph* which defines the data dependency between tasks (Figure 2).

B. Description of the SDR Hardware Platform:
The description of the SDR HW platform groups the elements into three classes: *processing elements* (PE), *communication architectures* (CA), and *memories* (MEM).
The class of PEs executes the functionality. This class can be divided into *programmable*, *reconfigurable* and *configurable* elements. Typical candidates for programmable elements are General Purpose Processors

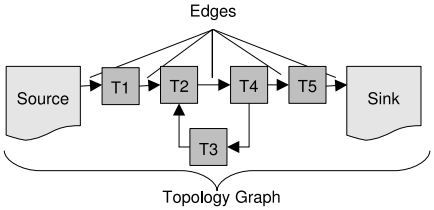


Figure 2: Exemplary decomposition of a waveform application

(GPP) or Digital Signal Processors (DSP). Reconfigurable elements like Field Programmable Gate Arrays (FPGA) usually offer a higher flexibility, but a lower performance than configurable elements like highly efficient Application Specific Integrated Circuits (ASIC).
Communication architectures describe the data exchange capabilities between the PEs. They are mainly described by their *type* (e.g. wires, point-to-point, network-on-chip) and their *interfaces* to the processing elements and memories. The class of *memories* summarizes all types of storage elements.
Both, the waveform application and hardware platform description, which forms the basis of the application-to-architecture mapping is discussed next.

C. Mapping:
The mapping of a waveform application onto an arbitrary platform has to consider *temporal* and *spatial* resource allocations. In [1] three different cases are defined (the notation m:n denotes that m-task are mapped on n-PEs):

- *Temporal Mapping – m:1 Mapping:*
  One single PE concurrently executes *m* tasks. Software solutions typically realize the necessary scheduling within (Real-Time) Operating Systems (RTOS/OS). Hardware solutions are typically implemented in terms of controllers.
- *Spatial Mapping – 1:n Mapping:*
  Spatial mapping splits a single task into *n* subtasks which are individually assigned to *n* different PEs. Thus, a spatial mapping usually leads to a parallel execution of functionality. In this case, a spatial scheduler, which is typically implemented on the previous adjacent node (*task*), becomes necessary to dispatch the work load to the different PEs.
- *Temporal and Spatial Mapping – m:n Mapping:*
  Here, *m* tasks are mapped onto *n* processing elements. In some cases, the *m:n* mapping can be reduced to a selection of *m:1* or *1:n* mappings. In other cases, it requires both types of schedulers at the same time, a temporal and spatial one.

D. Code Generation:
The (semi)-automatic generation of (SCA-compliant) SW/HW implementation is based on the application, the HW platform and the mapping. The concept foresees a mechanism to assemble the software parts (e.g. for GPPs and DSPs) based on the task functionalities. The main issue to be solved is the interfacing problem for task's communication. The application-to-architecture mapping along with the instantiation of communication schemes (e.g. based on libraries) enables automatic generation of optimal interfaces. Today's paradigm of hardware design focuses on IP-based design [7]. We align to this IP-based design concept and do *not* focus on a generic generation of the

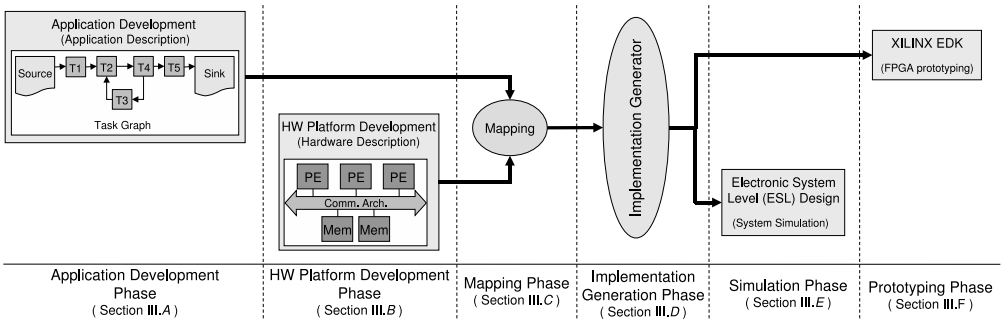


Figure 3: Workbench of the proposed concept

complete hardware or hardware blocks, e.g. from C/C++ to VHDL or Verilog. Although there is ongoing research in this field, tools are not mature enough today and rather limited in use.

III. WORKBENCH

The SDR design flow based on the proposed workbench is illustrated in Figure 3. In the first two phases of the SDR implementation, developers describe the SDR application and HW platform. These phases can be parallelized to reduce development time. The underlying structure of the descriptions of waveform, HW platform and mapping are based on the Extensible Markup Language (XML) [8] format, allowing easy porting between different tools. For demonstration purpose and in order to reuse the graphical environment, we integrated the workbench exemplarily into CoWare's Platform Creator Tool (PCT) [9].

In the following phase the developed application, given as a task graph, is being mapped onto the available PEs of the underlying SDR HW platform.

The output is a mapping description, which is then passed to the implementation generator along with the application and hardware platform description. The implementation generator can be configured to assemble either an Electronic System Level (ESL) [10] model or a prototype implementation based on Xilinx Embedded Development Kit (EDK) [11] and its HW IP-cores. Therefore, it can be used for verification and debugging of the SDR system.

The analysis of the simulation model and the FPGA prototype leads to necessary design iterations, in which the application, HW platform and/or mapping is improved. The workbench supports developers in accomplishing those design iteration steps by the seamless design flow. In the following sections the proposed workbench and the different development phases depicted in Figure 3 are highlighted.

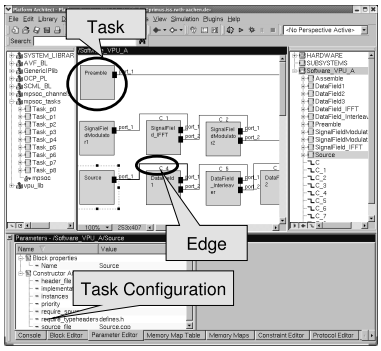
A. Waveform Application Development Phase:

The concept introduced in [1], defines the description of a waveform application by three key aspects: (i) functionality description, (ii) data exchange description and (iii) assembly of all sub-functions into one system topology. The implemented workbench covers these essential aspects in the following way:

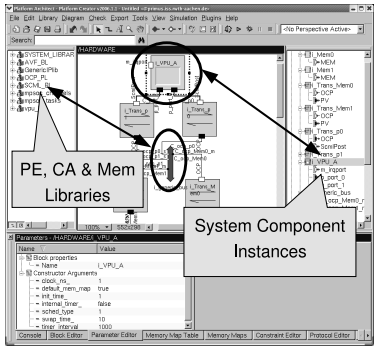
(i) Functionality:
It is essential that developers describe the waveform application in a way that generation of both, soft- and hardware code, becomes feasible. As presented earlier, the waveform description is based on basic programming concepts found in programming languages. Currently the workbench support is limited to C, as the programming constructs of the concept define a subset of the C language and therefore are completely covered by the C programming language. Nevertheless, functionality mapped on HW can be modeled adequately later on.

(ii) Data Types/Communication:
A key element for efficient and modular modeling is the independence of data communication and the task's functionality. Communication is defined by uni-directional edges between tasks. Bi-directional edges can be modeled by pairs of unidirectional edges. The workbench provides a communication interface for data exchange over edges, which is based on the basic principle of communication: sending and receiving of data. Therefore, an edge element can be accessed by a universal API defined by the methods *put* and *receive*. It is important to note that the edge element abstracts the communication at the conceptual phase. During the code generation phase an optimized communication scheme is applied such that all edge elements are replaced by a dedicated communication for implementation.

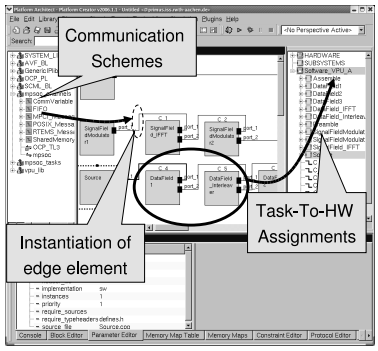
(iii) System Topology:
The system topology describes the complete SDR waveform application by a task graph. As depicted in Figure 4(a) the graphical environment of the workbench embedded into PCT allows the definition of such graphs based on tasks and edges. Developers can specify the parameters of each task and the location of the task's functionality by refering to a certain implementation.



(a) Workbench SDR Application View



(b) Workbench HW Platform View



(c) Workbench SDR Mapping View

Figure 4: Workbench environment

B. SDR HW Platform Development Phase:

Mapping a SDR application onto a SDR HW platform requires at least a coarse-grained model of the HW platform. Such platforms are typically Multi-Processor System-on-Chips (MPSoCs), which can be described with (i) Processing Elements (ii) Communication Architectures and (iii) Memories.

Designing such an MPSoC is a challenging task, therefore recent research in this field has introduced a new design paradigm called Electronic System Level (ESL) [10]. In ESL designs complete system models (virtual prototypes) are assembled. One prominent tool-suite for such ESL design is CoWare's Platform Architect (PA) [9] providing a graphical front-end called Platform Creator Tool (PCT). Modeling the SDR HW platform has been developed with respect to ESL design methodology, thus the concept's realization can utilize the PCT as illustrated in Figure 4. Component libraries provide different IP blocks, e.g. GPPs, Network-on-Chips (NoCs) and Memories. Developers can assemble those blocks within the graphical environment to a complete system model as depicted in Figure 4(b).

C. Application-to-Platform Mapping Phase:

Mapping the application onto the HW platform is, as previously mentioned, a *temporal & spatial task mapping*. Spatial mapping denotes the assignment of a task to one or multiple PEs, whereas a temporal mapping refers to the allocation of a time budget on a particular PE. Mapping is one of the highly crucial steps in the design cycle, where different options have to be considered and an optimal or at least near-optimal solution has to be found. Therefore, fast exploration of different design decisions is inevitable within the mapping phase. The workbench allows investigation of such different mapping strategies. The mapping of tasks to PEs is illustrated in Figure 4(c).

The description of the waveform application does not specify the exact communication mechanism, which developers have to take care of. The workbench allows developers to instantiate each edge element with a dedicated communication scheme, e.g. a shared memory or a FIFO communication scheme. Similar to the assignment of tasks to processing elements, developers can assign these communication schemes to the edge elements in a graphical manner (Figure 4(c)).

The output of the mapping phase is a description of all relevant information for code generation of the SDR implementation in XML format.

D. Implementation Generation Phase:

The result of the mapping phase is fed into the workbench's implementation generator. The generation of the SDR implementation is separated into the generation of the *software executables*, the *system simulation model* and the *prototype*.

The generation of the *software executables* consists of the following steps:

1) The functionality of all tasks, specified by the developer in the application description phase (discussed in Section III-A), is extended by implementations of dedicated communication schemes.
2) Schedulers, whether temporal or spatial, are inserted as necessary and specified.
3) Finally, the software executables are compiled for the underlying programmable device.

The advantage of the (semi-)automatic generation of the software executables releases developers from repeating tedious tasks, allowing more time investment in development of the task's functionality. This advantage allows for faster exploration and analysis of design decisions like different mappings.

The compiled software executables can be executed in both, simulation and prototype, without any modifications. The generation of software executables is followed by either the generation of the ESL system simulation model (Section III-E) or the prototype (Section III-F).

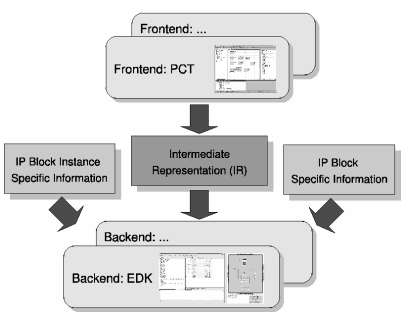


Figure 6: Prototyping Phase

E. Simulation Phase:

The workbench utilizes the build-in capabilities of PCT to generate a simulation model out of the SDR HW platform description. Processing elements, communication architectures and memories are instantiated by their corresponding simulation model allowing simulation of the complete system. Simulation allows developers to debug and analyze their SDR HW platform and application. In [12] the capabilities regarding analysis and debugging of simulation and prototyping is discussed in detail. Typically, the technique of Transaction Level Modeling (TLM) is utilized for system simulation, which abstracts the communication such that simulation of a complete system gets feasible. The language used for system modeling is mostly SystemC [13], which is a HW oriented extension for C++. Please note that the workbench expects the existence of TLM simulation models. Development of such models is not in the scope of this paper or the workbench. The only exception is made for hardware accelerators: As previously mentioned simulation can make use of abstracting the system, which allows the generation of hardware IP-blocks out of C-code for simulation purposes. The functionality is encapsulated within a SystemC block, which allows developers to quickly investigate different design decisions by simulation. However, these models can only be generated for simulation and can not be used in the later prototyping.

F. Prototyping Phase:

Fast prototyping is essential to verify functional correctness and to investigate real-time effects as well as interaction of the SDR with other wireless communication devices. Today, HW prototyping can be efficiently performed using Field Programmable Gate Arrays (FPGAs). One of the major vendors of FPGAs, Xilinx [14] provides a tool called Embedded Development Kit (EDK) for IP block based FPGA designs.

The proposed workbench bridges the gap between the ESL design and the Xilinx EDK in an automatic manner such that consistency is ensured[1] as long as HW IP blocks and simulation models are consistent. Figure 5 depicts the basic principle of this step, which is split into two phases, named the front-end and back-end phase. In the front-end phase the description of the application, the HW platform and the mapping is parsed and an Intermediate Representation (IR) is generated. The back-end generates the Xilinx EDK's configuration applying user- and vendor-specific mapping rules for each IP block. These rules describe the relation between the HW IP block and the simulation model. The generic structure of front- and back-end allows future support for other ESL and/or FPGA tools by simply replacing the front- and/or back-end. The output configuration can then be used to compile the FPGAs design and to run the system on the FPGA prototype in order to verify the SDR system in a real-time environment.

IV. CASE STUDY

The following case study has been realized as a test case for our workbench. A simple waveform has been selected as a testcase to keep the design complexity reasonable while putting the focus on the investigation of the SDR development flow aspects. Therefore, the waveform does not include channel coding, interleaving, pilot insertion, etc. In this section we focus only on the receiver part.

The waveform application is depicted in the upper left part of Figure 6 and consists of the following elements:

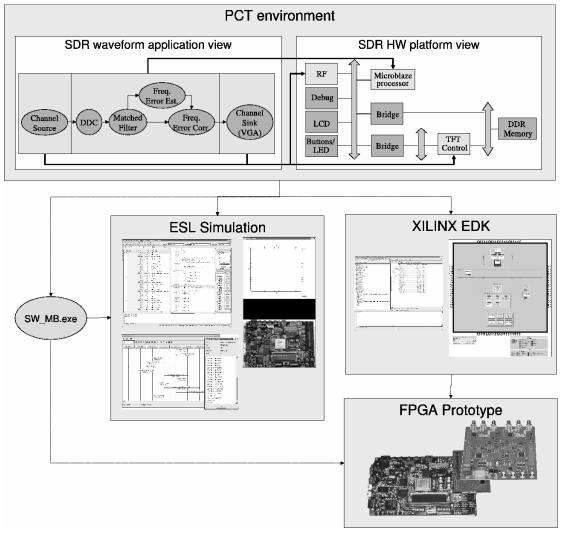


Figure 5: Case study

[1] Please note that there exists an initiative called SPIRIT [15], which defines a common data structure between IP vendors allowing exchange of IP blocks between different tools and use cases. However, this is at the time of this paper not mature and still in development.

- Channel source, providing complex IQ samples at an intermediate frequency (IF)
- Digital down conversion (DDC) from IF to the baseband Matched filter (MF)
- Frequency error estimation
- Frequency error correction
- Channel sink

This waveform has been mapped on a SDR hardware platform based on Xilinx ML402 FPGA [14] board. This platform comprises:

- Signalion SRFC RF board [16]
- Xilinx Microblaze processor [14]
- Hardware accelerator for MF and DDC
- OPB bus (communication architecture)
- On-chip block memories (BRAM)
- VGA output

In the mapping phase, frequency error estimation and correction have been mapped to the Microblaze processor in order to exploit the flexibility for testing different synchronization strategies. The output (channel sink) has been visualized on the VGA output. The MF and DDC tasks require for the case study less flexibility and thus have been mapped onto the hardware accelerator. This step comprised the instantiation of the communication scheme "memory mapped IO" to the edges between the DDC task and the error estimation/correction tasks. This mapping process and the resulting simulation models and prototype implementations are depicted in Figure 6.

Previously, developers had to implement both the ESL simulation and the FPGA model separately, which caused high development effort and severe consistency issues. The encounter of errors in the simulation model and/or the FPGA prototype forces developers first to verify that both the model and the prototype are consistent. Only afterwards, debugging of the error can be addressed. The proposed workbench allows developers to maintain only one HW platform description from which both, the ESL simulation and the FPGA prototype, can be generated. This ensures consistency of both models and reduces development time significantly. The workbench has been proven very helpful during our small case-study. It will help shortening significantly the design time of future SDR systems and prototypes.

V. CONCLUSION

In this paper, a workbench for a design flow for Software Defined Radios has been proposed. The contribution is the workbench realization of a concept allowing a seamless SDR development from the waveform's specification down to the implementation. Key elements of the workbench are the description of the (i) SDR application, (ii) the SDR HW platform and (iii) the application-to-architecture mapping. Based on these descriptions the implementation generator can (semi-)automatically assemble an Electronic System Level (ESL) model and/or FPGA prototype design. Simulation of the ESL model allows developers to debug, verify and analyze their complete SDR. Prototyping of the generated FPGA design in a real-time environment allows efficient verification and debugging of the SDR. On top of a small case study we have highlighted the capabilities of the proposed workbench and methodology.

In future we will concentrate on applying the workbench on larger examples with more realistic waveforms, e.g. the MIL-STD-188-110B [17]. Additionally we will investigate the key issue of portability for SDRs in depth.

VI. ACKNOWLEDGMENT

The authors would like to thank J. Holzer, C. Hatzig, S. Hartmann and H. Siegmar of the WTD-81 for inspiring discussions. Additionally the authors would like to thank J. Reinecke, M. Rustagi and S. Wallentowitz for their valuable contributions.

VII. REFERENCES

[1] T. Kempf, M. Adrat, E.M. Witte, et al.. A Concept for Waveform Description based SDR Implementation. In *4th Karlsruhe Workshop on Software Radios (WSR'06)*, Karlsruhe, Germany, March 2006.

[2] JTRS. Software Communications Architecture (SCA) Specifications V2.2. *http://jtrs.army.mil/*.

[3] T. Kempf, et al. An SDR Implementation Concept based on Waveform Description. *FREQUENZ: Journal of RF-Engineering and Telecommunications, Berlin*, (9-10), 2006.

[4] E. D. Willink. Waveform Description Language: Moving from Implementation to Specification. *IEEE Military Communications Conference (MILCOM 2001)*, Oct. 2001.

[5] R. S. Prill. E2e reconfigurability using osi layered wdl canonical radio model. *E2R Workshop on Reconfigurable Mobile Systems and Networks Beyond 3G*, Sept. 2004.

[6] M.S. Gudaitis and R.D. Hinman. Practical Considerations for a Waveform Development Environment. *IEEE Military Communications Conference (MILCOM2001)*, October 2001.

[7] Wander O.Cesario, et al. Multiprocessor SoC Platforms: A component-based design approach. *IEEE Des. Test*, 19(6):52–63, 2002.

[8] Extensible Markup Language (XML) *http://www.w3.org/xml*

[9] CoWare inc. *http://www.coware.com/*.

[10] Martin Grant, et al. *Electronic System Level Design and Verification*. Morgan Kaufmann, Feb. 2007.

[11] Xilinx. Platform Studio and the EDK. *http://www.xilinx.com/ise/embedded_design_prod/platform_studio.htm/*

[12] E.M. Witte, T. Kempf, et al. Waveform and Prototype Debugging in a Seamless SDR Development Flow. In *Military CIS Conference 2007 (MCC2007)*, Bonn, Germany, September 2007.

[13] T. Grötker, S. Liao, G. Martin, and S. Swan. *System Design with SystemC*. Kluwer Academic Publishers, 2002.

[14] Xilinx inc. *http://www.xilinx.com/*.

[15] The SPIRIT Consortium. *http://www.spiritconsortium.org/*.

[16] Signalion GmbH. Prototyping the Wireless Future. *http://www.signalion.de*.

[17] MIL-STD-188-110B DoD Interface Standard. April 2000