

A Rapid Prototyping Methodology for Algorithm Development in Wireless Communications

A.Burg, B. Haller, E. Beck, M. Guillaud, M. Rupp, L. Mailaender
Bell-Labs Wireless Research, Lucent Tech.
791 Holmdel-Keyport Rd., Holmdel NJ, 07733-400
apburg@iis.ee.ethz.ch, {bhaller1, ericbeck, maxime, rupp, lm}@lucent.com

Abstract:

Rapid prototyping has become an important means to verify the performance and feasibility of algorithms and concepts for wireless communications. This paper presents an FPGA/DSP based rapid prototyping methodology and platform that requires only minimal IC/FPGA design skills and merely a very basic knowledge of hardware description languages such as VHDL. It allows algorithm designers to implement their ideas quickly and to test them in real-time under real world conditions. The design and implementation of a WCDMA downlink is used as an example to describe our initial experience with the proposed flow.

I. Introduction

In many areas rapid prototyping has become an important means for the functional verification of a design in a well-defined environment. Its main goal is to avoid extremely long simulation runs and to improve the functional assessment of complex systems.

In the development of wireless communication systems rapid prototyping becomes important at an early stage of the development [1–3]. The fact that these systems have to deal with a variety of effects in a multifaceted environment makes the development of new algorithms and their accurate verification via computer simulations a difficult task. A first evaluation of the performance of a novel idea is usually done mathematically based on very basic, typically simplistic assumptions. Gaining confidence in algorithms developed with theoretical models by means of simulations is an essential part of the early development stage, since wireless channels suffer from a wide variety of effects that are often difficult to model accurately. Simulation tools like THE MATHWORKS Simulink [4]

are therefore being used to work around this problem. However, even these simulation models often do not represent the actual system in sufficient detail. Working in a real environment becomes especially critical when algorithms rely on special properties of the transmission channel. Multiple antenna systems employing transmit diversity or space-time coding (e.g., BLAST [5]) are good examples. It is also important for establishing reliable and reasonable standards, and for developing more advanced channel models.

Real-time testbeds also become increasingly important as functions at the application layer start to interact more closely with functions at the physical layer. The design of speech codecs and of joint source/channel or source/modulation coding algorithms for wireless multimedia applications are good examples.

The goal of the “Bell Labs Algorithm Development and Evaluation” (BLADE) initiative is to provide a platform and a methodology that allows a fast and easy verification of research ideas from the physical through to the application layer at an early stage in the development process. It allows researchers to complete simulations in a real environment and to get realistic assessments about the complexity and implementation costs of their ideas.

This paper is organized as follows: In Section II a hardware platform that fits the needs of prototyping for wireless communications is presented and the used software tools as well as the BLADE methodology are described. The system is designed to provide researchers with a means to take their algorithms to a hardware implementation, without requiring in-depth knowledge of IC/FPGA design methodologies or hardware description languages (e.g., VHDL or Verilog). We briefly present a prototype implementation of a WCDMA downlink system together with some measurement results and the integration of a voice codec, based on the BLADE methodology in Section III.

In Section IV we discuss our experiences from this first experiment and give some suggestions for future improvements. Finally, Section V concludes the paper.

II Setup and Methodology

A. System Hardware

The hardware setup consists of a flexible combination of TI 'C6x series digital signal processors (DSPs) [6] and XILINX Virtex series field-programmable gate arrays (FPGAs) [7]. The basis for the system is a development platform from SUNDANCE [8] that consists of carrier boards with a PCI bus interface along with a variety of high performance DSP and FPGA modules. Each module is equipped with four asynchronous bi-directional 20Mbytes/s links. Some FPGA modules provide additional synchronous 16-bit parallel links. A typical PCI carrier board supports up to four modules. Evaluation boards from different vendors are also used for data conversion (i.e., ADC and DAC). The use of well supported off-the-shelf components saves a lot of time during the implementation phase as compared to custom-made systems. **Figure 1** shows a basic setup. Supplementing the high speed DSPs with FPGAs is necessary since the DSPs' capabilities of handling high rates are very limited both in terms of compute power and I/O bandwidth. The FPGAs also help relieve the DSPs from performing the relatively simple but high rate data path type operations that frequently occur in the front end of wideband communication systems. [14]

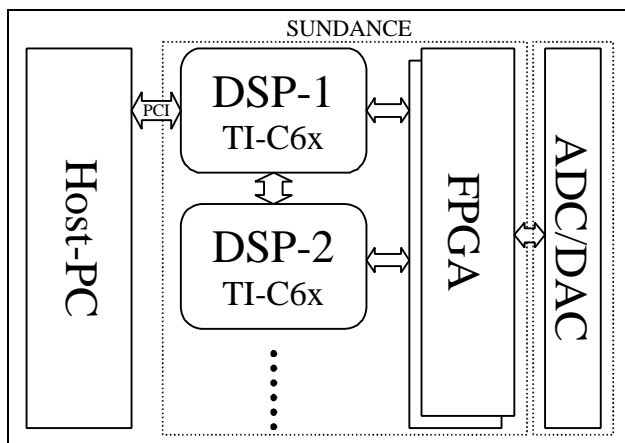


Figure 1: Basic hardware setup for BLADE

B. BLADE Methodology

The main goal of the BLADE methodology is to provide a way to make use of the combined capabilities of DSPs and FPGAs without having to employ assembler

language or HDLs in order to rapidly realize ideas in hardware. It should also provide algorithm designers with a way to integrate their blocks into an existing high-level Simulink simulation setup for verification and successive refinement down to the implementation level.

To achieve this the suggested design methodology relies on the use of C as the primary description language for the implementation and on the 3L Diamond real-time DSP operating system [9] for inter-process communication. The proposed design flow is outlined in **Figure 2**. The right side of the figure shows the simulation flow and the left side depicts the flow that leads to the real-time implementation. C code that describes cycle or block oriented operations can be simulated using the block oriented Simulink environment. Automatically generated wrappers are used to create the required S-functions. However typical systems also contain a certain amount of control flow that can either be described as a finite state machine (FSM) or as procedural (i.e., linear) C code with synchronization capabilities. Procedural descriptions are often more convenient and intuitive. They are also required for complex block oriented functions that execute asynchronously with respect to the other functions in the system and are preferably mapped onto a DSP. During the simulation phase, these pieces of code need to be co-simulated, running as separate tasks in a multitasking environment in parallel to the cycle oriented Simulink simulation. The interfaces into the Simulink environment (and later to the FPGA) are provided through a library of interface templates that are specific to the current simulation (or realization) environment. In a UNIX simulation for example sockets are used to emulate the com-port interface. Memories, ADCs or DACs and their interfaces are also instantiated from the template library.

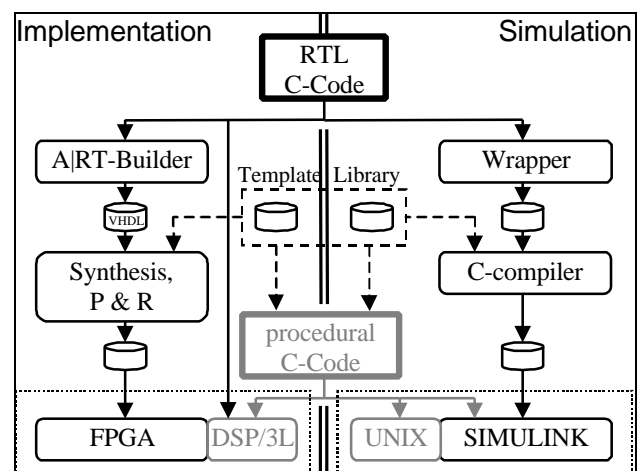


Figure 2: BLADE design flow

After a successful initial simulation of the system, the design can be transferred to the real-time platform. A final decision has to be made regarding whether functions should be mapped onto the DSP or FPGA. Procedural functions are placed on the DSP, while both DSPs and FPGAs are possible targets for the remaining cycle/block oriented functions, depending on their performance requirements.

For those block operations that are mapped onto the DSP a real-time operating system (3L Diamond) is used to provide the communication interfaces between them. Each block runs as a separate task. Activity on the ports can control the scheduling process. Communication channels replace the ports of the blocks in the Simulink block diagram and a configuration file is used to specify the links between them. The linear code that was used for the simulation in the Unix environment can be reused with the DSPs real-time operating system by linking it with the system specific interface template library for the interfaces to the FPGA and to other DSPs. This system also allows transferring functionality onto the host PC if necessary.

For the mapping of blocks that will be placed onto the FPGA we use FRONTIER DESIGN's A|RT Builder [10] to automatically create synthesizable VHDL code from the original C code. It performs a direct translation of C code into VHDL, without doing any type of behavioral synthesis. The designers are therefore responsible for refining their C code designed for FPGA implementation into a syntax that describes a register transfer level (RTL) structure of the block that can be mapped into hardware. In contrast to System-C [11] the tools expect sequential C code that has no capability of expressing concurrency. Floating-point operations in these translated functions are replaced with fixed-point operations by using the A|RT Library [10] fixed-point data types. Simulink blocks are either replaced by VHDL templates or also described in RTL-C. Note that each of these blocks can still be used in the Simulink simulation, as RTL-C code is standard executable C.

Due to the moderate size of the designs a top-down synthesis approach using the SYNOPSIS Design Compiler [12] can be used to translate the generated VHDL code into an EDIF netlist, which is imported into the XILINX Alliance place & route tool. Scripts are used to hide the details of the synthesis process from the designer.

III. Design Example

As an example we implemented a basic WCDMA downlink system. Its high-level block diagram is depicted in **Figure 3**. The system is a 4Mchips/s DS-

SS-SSM WCDMA system, with a separate sync/pilot-channel and a variable set of users transmitting at different rates. The algorithm designers' motivation behind putting this system onto the test platform was to explore the performance gain that can be achieved in such a system by using an equalizer structure instead of the usual rake receiver. The system also contains many of the frequently used blocks in wideband communications and it was therefore a good candidate to build up a collection of blocks for future reuse.

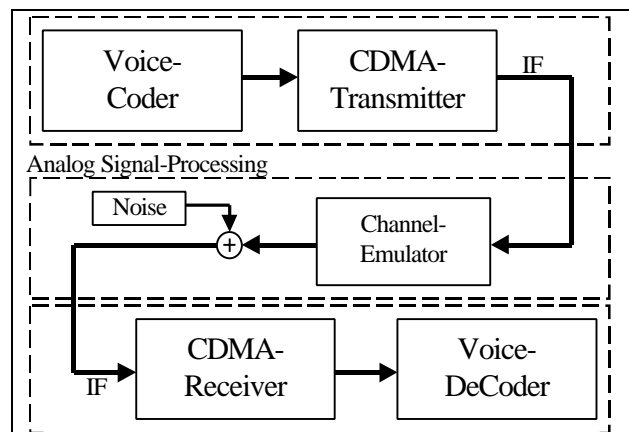


Figure 3: WCDMA downlink prototype

A. Transmitter

The transmitter consists of a combination of one Virtex 400k gate FPGA and a TI 'C67 DSP. The DSP runs a standard ACELP voice encoder and generates the data stream. It also configures the FPGA and controls the transmitter parameters. The FPGA contains the actual WCDMA downlink functionality for a variable data rate main user. Besides the transmission of the desired user, "dummy-users", sending random data, can be turned on. Their number and power can be adjusted to simulate different traffic situations. A sync/pilot signal is also added before the signal is digitally up converted to an IF frequency and sent to the DAC.

In the laboratory the wireless channel is modeled with a radio channel emulator and an additive white Gaussian noise (AWGN) generator. Alternatively, an RF air interface may be used to do measurements in an exterior environment.

B. Receiver

The receiver is the more interesting and more challenging part of the system. It consists of a 'C62 fixed-point and a 'C67 floating-point DSP and a Virtex 1000k gate FPGA. It incorporates the following functionality:

- IF interface
- Synchronization
- Tracking of frequency offset
- 4-finger rake receiver
- Polyphase equalizer
- Data interface/buffer and BER measurement

Table 1 shows some of the criteria that were applied for the partitioning of the functionality between the DSP and the FPGA. It also shows where the main blocks of the system were finally placed. Putting all the functions that operate on a sample by sample basis in a 4Mchips/s 4× oversampled system onto the FPGA is unavoidable, because of the tremendous number of operations. An important reason for implementing almost all the control flow operations in the DSP was to allow a high degree of flexibility to make changes quickly and on the fly, without the need to redo the time consuming FPGA synthesis. This was particularly important, since the turnaround time for the receiver FPGA implementation was on the order of 5 hours.

The DSP-FPGA interface uses a global bus on the FPGA that connects to all modules and can be used to read and write parameters to and from each block. The designer needs to handle these interface commands in his C code that describes the blocks on the FPGA. A relatively simple protocol, incorporating only marginal overhead was designed. **Figure 4** shows the Simulink model of the receiver. The interfaces from the blocks are collected in the left bottom corner of the top level schematic. They connect to the “DSP” interface from the template library at the next level of hierarchy. The procedural code that handles the control flow and the other DSP-based operations listed in Table 1 is not shown, as it runs as separate task in the Unix environment in parallel to the cycle oriented Simulink simulation. In the real-time environment it run on the DSP in parallel to the FPGA implementation.

Figure 5 shows the BER-performance of the RAKE-receiver plotted vs. the number of users at a rate of 256kb/s for a standard outdoor-to-indoor channel model at 5Hz doppler frequency. The results of a floating point

simulation and of a theoretical analysis are given as well for comparison.

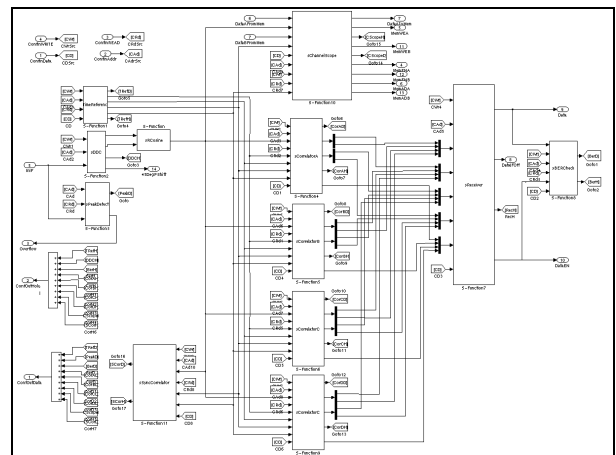


Figure 4: Simulink model of the receiver

Besides the actual receiver functionality a voice decoder was implemented on the second DSP, together with independent interface blocks to the CDMA receiver and to an audio codec. It works on a frame by frame basis and is embedded in the system using 3L Diamond to handle the blocks connections as described above. It was also simulated in Simulink just like the actual WCDMA receiver and then mapped solely onto the second DSP.

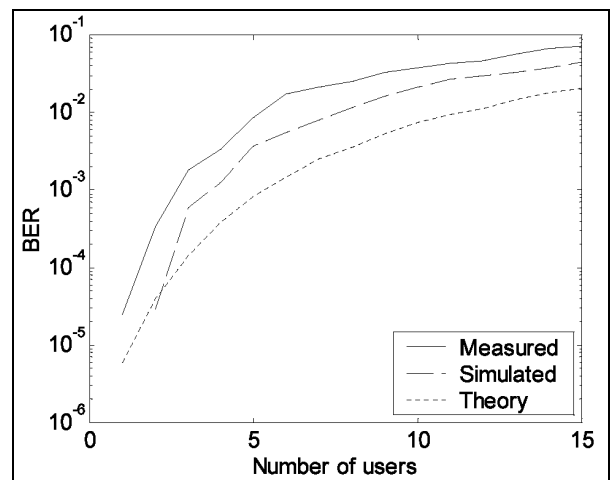


Figure 5: BER vs. number of users

Table 1: DSP/FPGA partitioning for the receiver

	Criterion	Function
DSP	<ul style="list-style-type: none"> • Control flow (linear code) • Complex & irregular operations • Complex data path ops. Operations at low rates with low I/O requirements • Floating-point or high precision fixed-point operations 	<ul style="list-style-type: none"> • Synchronization and tracking control • Rake finger assignment • Equalizer computation (matrix-inversion)
FPGA	<ul style="list-style-type: none"> • Data path operations • Operations at sampling and chip rate 	<ul style="list-style-type: none"> • IF down-conversion, filters • Rake receiver, equalizer • Correlators for sync. and channel estimation

IV. Summary

Slow turn-around times, mainly due to long P&R runs for the FPGA turned out to be one of the major problems in the rapid prototyping and evaluation process. This overhead was reduced by keeping parameters in the FPGA modules programmable and controlling them from the DSP at run time. This also keeps modules flexible and easier to reuse. The fact that C code is being used to describe RTL-type parallel structures imposes certain restrictions on the design hierarchy that are sometimes less elegant and less intuitive than the structure that one would impose when writing VHDL. The use of multiple clock designs is currently not supported in the proposed flow. This is one of the main restrictions that limits the possibility to make use of optimizations that are especially well suited for medium data rate systems on FPGAs (e.g., bit-serial architectures or decomposition of multipliers). The use of C as design language also limits the use of specifically optimized blocks from vendor specific FPGA libraries, such as the XILINX CoreGen library. This is because our methodology does not provide hooks to instantiate these blocks directly from within a C module.

However, using a C-based approach for an FPGA implementation yielded a lot of advantages. Avoiding the use of VHDL was highly appreciated by the algorithm designers working on the project, even though the C to VHDL translation does not involve any behavioral synthesis and therefore forces the designer to write register transfer level (RTL)-type C code. A new tool called A|RT Designer [10] may avoid this problem in the future by offering behavioral synthesis capability.

Using Simulink for debugging of the design, it turned out that a capability to store waveforms of the signals and/or variables inside the C code is missing. Waveforms can only be recorded on the block boundaries in Simulink. A regular C debugger was used to track changes inside the C code on a cycle by cycle basis, but this was a tedious task. Using a standard VHDL simulation tool with a Simulink interface or an on-chip logic analyzer like XILINX' ChipScope [7] might be a valuable add-on. Additionally, an add-on to Frontier Design's fixed-point library also offers a statistics capability to track overflows and other numerical problems in the simulation. Using Simulink proved to be a valuable means for system verification as existing blocks could be incorporated at an early evaluation/design stage. However functional Simulink blocks need to be replaced with custom C code in the implementation phase or need to be replaced by templates.

V. Conclusions

We presented a methodology that allows rapid prototyping for research in wireless communication. It uses C as the primary language and integrates common system design tools such as Simulink into the flow. The C based approach is well suited for hardware/software co-design, allowing one to exploit the unique features of DSPs and FPGAs in a combined system. The BLADE methodology also provides a way to co-simulate DSPs and FPGAs in a multitasking environment. It offers very high flexibility and a methodology to combine cycle oriented code with procedural code to efficiently describe control flow and relatively irregular operations in a natural and intuitive way. For algorithm research and development it is a real alternative to the use of relatively expensive high-level system design tools such as COSSAP [12] and SPW [13]. However, it does not claim to be a solution for a production environment, as its capabilities to achieve performance optimized designs are certainly limited.

References

- [1] M. RUPP, E. BECK, AND R. KRISHNAMOORTHY, "Rapid Prototyping for High Data Rate Wireless Local Loop," in *Proc 33rd Asilomar Conf.*, 1999, Vol. 2, pp. 993–997.
- [2] S. DAS, S. RAJAGOPAL, C. SENGUPTA, AND J. CAVALLARO, "Arithmetic Acceleration Techniques for Wireless Communication Receivers," in *Proc 33rd Asilomar Conf.*, 1999, Vol. 2, pp. 1469–1474.
- [3] V. SUNDARAMURTHY AND J. CAVALLARO, "A Software Simulation Testbed for Third Generation CDMA Wireless Systems," in *Proc 33rd Asilomar Conf.*, 1999, Vol. 2, pp. 1680–1684.
- [4] MATLAB and Simulink are trademarks of THE MATHWORKS INC. <http://www.mathworks.com>.
- [5] G.D. GOLDEN, G.J. FOSCHINI, R.A. VALENZUELA, AND P.W. WOLNIANSKY, "Detection Algorithm and Initial Laboratory Results Using V-BLAST Space-Time Communication Architecture", *Electronics Letters*, Vol. 35, No. 1, pp. 11–14, Jan. 1999.
- [6] TEXAS INSTRUMENTS TMS320C6000 DSP Platform <http://www.ti.com/sc/docs/products/dsp/c6000/index.htm>.
- [7] Virtex and ChipScope are trademarks of XILINX INC.
- [8] Sundance is a trademark of SUNDANCE MULTIPROCESSOR TECHNOLOGY LTD. & SUNDANCE DSP INC.
- [9] Diamond is a trademark of 3L LTD.
- [10] A|RT Builder, A|RT Designer and A|RT Library are trademarks of FRONTIER DESIGN <http://www.frontierd.com>.
- [11] <http://www.systemc.org>.
- [12] Design Compiler and COSSAP are trademarks of SYNOPSIS INC. <http://www.synopsys.com>.
- [13] SPW is a trademark of CADANCE CORP.
- [14] R. BAINES, "The DSP Bottleneck," *IEEE Comm. Magazine*, 33(5):46–54, 1995.