A REUSE APPROACH FOR FPGA-BASED SDR WAVEFORMS

Kevin Skey, John Bradley, Karl Wagner
The MITRE Corporation
Bedford, MA

ABSTRACT

Software targeting general purpose processing (GPP) elements has been successfully reused for software defined radio (SDR) platforms in support of low-bandwidth waveforms. The Joint Tactical Radio System (JTRS) Software Communications Architecture (SCA) promotes reuse of GPP-based software by providing a consistent framework for developing reusable waveform implementations. However, high-bandwidth waveforms, such as those used in above 2 GHz MILSATCOM terminals, overwhelm the capabilities of GPP-only radios making field programmable gate arrays (FPGAs) a necessity in high-bandwidth radio systems. The SCA does not address the development of reusable FPGA-based waveform implementations. This paper presents an approach supplementing the current SCA to address FPGA-based platforms using System-on-a-Chip (SoC) best practices for design reuse, including common interfaces and a robust system simulation environment.

INTRODUCTION

In an effort to reduce hardware and software development costs, many of today's military (and commercial) satellite communication systems use SDR technology to support high data-rate modems. In effect, an SDR-based modem is a reuseable complex heterogeneous hardware platform. These platforms are typically architected for flexibility by using a mixture of GPP, DSP and FPGA processing elements to support a family of radio waveforms implemented as heterogeneous multiprocessor software applications. There are intrinsic benefits to this approach such as the ability to interoperate with numerous legacy radios as well as in-system upgradeability and maintenance through software only modifications.

In addition to the potential benefit of hardware reuse, it is also possible to reuse the radio software application by porting it to a different SDR platform. However, porting an existing software radio application to a different platform can present the largest challenge to the SDR developer. For example, the source and destination SDR platforms could be architected very differently due to radio form factors. SDR platforms can have a wide variation in RF-IF and baseband boundaries, different processing elements (e.g. GPP, DSP, FPGA) as well as different operating systems, APIs, services and transport layers. A successful software port requires the developer to have a thorough understanding of the hardware architecture and software operating environments (OE) for both the source and destination platforms.

To address the porting issues for these complex heterogeneous SDR platforms, the JTRS program contracted the development of the SCA [1] to help enable the reuse of software radio applications (or waveforms). However, the current SCA only supports the reuse of heterogeneous GPP-based software waveform components. For the high data-rate waveforms used in military satellite systems, GPPs and DSPs may not be able to support the required high performance processing. Therefore, code reuse becomes more difficult as much of the modem processing resides in FPGA devices.

Recognizing the deficiencies, there was a JTRS-sponsored industry working group actively addressing portability issues for DSP and FPGA based waveforms [2]. However, the working group was disbanded before a consensus solution could be reached. If waveform reuse is an important cost factor for military programs, then a reuse approach for FPGA-based waveforms is still needed. In this paper, we offer a FPGA reuse strategy for SDR by leveraging techniques common in the development of System-on-a-Chip (SoC) integrated circuits.

SCA HIGH BANDWIDTH CHALLENGES

The SCA was developed to enable portability through a combination of component-based software [3] techniques, common CORBA middleware, and XML descriptions. The primary purpose for SCA was to create an operating environment which enables the creation of portable software radio applications by separating the software concerns from the hardware platform. Although, SCA does aid portability for GPP-based software, it does not sufficiently cover the modem layer processing typically performed by DSP or FPGA resources. In many cases, the modem is abstracted behind an SCA adapter which hides the implementation details from the radio application as shown in Figure 1(a). In some cases, this functionality can be a significant part of the SDR implementation as illustrated in Figure 1(b).

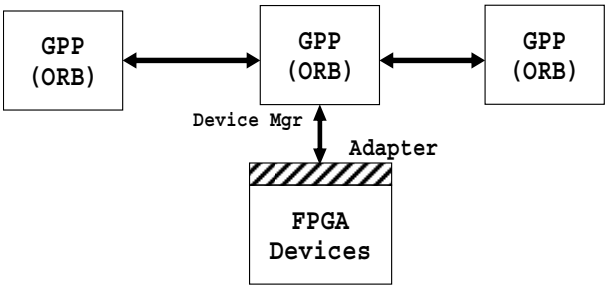

Figure 1(a): SCA Adapter Illustration for FPGA

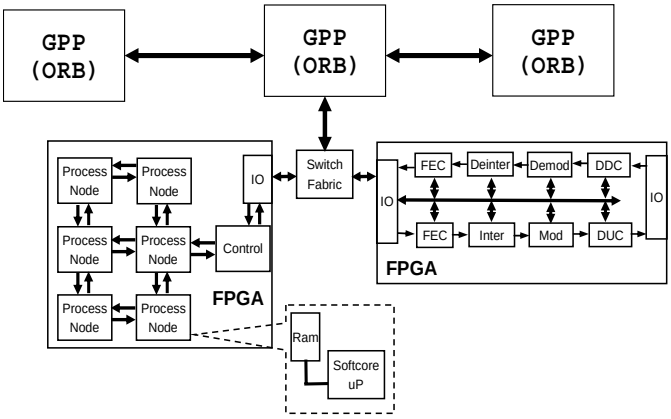

Figure 1(b): Adapters Hiding Complex Functionality

In fact, for high bandwidth and high data-rate waveforms, a significant amount of the modem and protocol processing is indeed implemented using FPGA devices. Even if it is feasible for some of the processing to be accomplished using GPP or DSP devices, there is a trend toward more FPGA devices as these devices offer the benefits of better risk reduction and future proofing. The parallel processing capabilities and reconfigurable nature of FPGAs offer SDR developers greater flexibility to support unforeseen requirement changes as well as the ability to host future, higher performance waveforms. As FPGA devices become dominant in an SDR platform, more of the software radio is being implemented in reconfigurable hardware. Therefore, if reuse and portability are the goals, this fact stresses the need for an additional approach to augment the SCA.

AN FPGA IS NOT A GPP

As previously discussed, FPGA devices are beginning to dominate today's high performance SDR platforms. It is common for those unfamiliar with FPGAs to incorrectly assume that they share the architecture and development flow of GPPs and DSPs. The traditional SDR system is thought to be a software centric platform, so it is understandable how one can conclude FPGA devices are simply software programmable devices.

For comparison, an SDR system containing GPP or DSP devices will have fixed hardware compute resources and peripheral interfaces. A software developer typically uses high-level languages to describe the operations required to process incoming and outgoing data on the mostly fixed hardware while being concerned with task performance and memory management issues. In almost all cases, an operating system (OS) with hardware abstraction layer (HAL) is used to help abstract away the hardware details.

In contrast, FPGAs are very complex reconfigurable System-on-a-Chip (SoC) devices with multiple hard/soft core processors, configurable memory, specialized I/O, and area for multi-million equivalent ASIC gate custom designs. In fact, today's FPGAs are equal in complexity to many SoC devices of 4 to 5 years ago [4]. Therefore, an SDR developer targeting FPGAs needs strong digital hardware design skills as well as intimate knowledge of the device architecture and the available resources such as look-up tables (LUTs), routing, flip-flops, clock generation, and specialized logic core macros. Also, efficient device resource utilization depends mainly on the physical design expertise using the FPGA place and route tools, where concerns such as physical floor planning, logic cell placement, design constraints and routing congestion significantly diverge from those of a GPP-based compile-link software development flow.

The intent of this section was not to over simplify the known design complexity of heterogeneous GPP software development, but rather highlight the contrasting requirements for FPGA devices. In this paper, we will focus on methods to improve the portability of waveforms implemented on FPGA-based SDR platforms. We now start our discussion, beginning on the topic of how separating the platform concerns from the waveform component development can aid portability.

SEPARATING PLATFORM CONCERNS

Future high performance MILSATCOM waveforms will require the partitioning of functionality onto multiple FPGA devices across several PCBs. This increases the complexity of the waveform deployment significantly compared to a single device implementation since the

waveform must now have a decomposition strategy for efficiently partitioning its components across multiple FPGA devices. Adding to this predicament, the developer must not only be concerned with connections between the waveform component cores internal to the FPGA, but also must be knowledgeable of the interconnect switch fabrics between the FPGAs themselves. These switch fabrics can be multiple Gigabit transport links (e.g. RapidIO) spanning across the PCB, mezzanine connectors and backplanes. Even with a proven COTS product, reaching the maximum specified link data-rate can require significant effort working through many possible signal integrity issues. With these additional challenges, debugging waveform components on the platform while still working the data-path throughput performance issues can significantly increase the overall development time. One solution is to separate the platform integrity and performance effort from the waveform component development. This is a common approach in the development of SoC devices.

For example, when developing an SoC, a major portion of a design effort is spent on the top-level interconnect optimization which is the process of partitioning (or floor planning) the major functions into resource estimated block regions as shown in Figure 2.

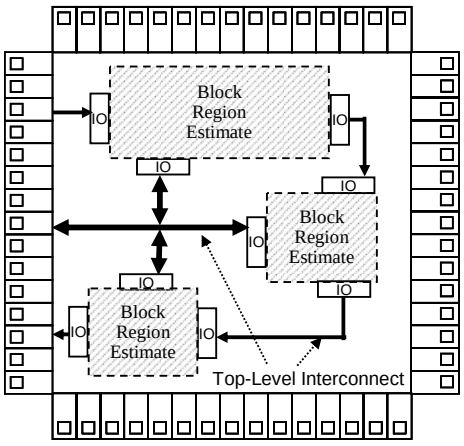

Figure 2: SoC Top-Level Interconnect Optimization

Then, using a predefined (or standardized) I/O interface definition, the top-level interconnect timing performance is optimized between the boundaries of the block regions. A standardized I/O definition offers the benefit of isolation between the individual blocks and the top-level interconnect when design changes are made. The primary benefit is after the functional block development is completed; the blocks can be essentially dropped in without disturbing the top-level optimization.

In this paper, we propose a similar approach to increase the deployment efficiency of FPGA-based waveforms. Through the use of standard interfaces (e.g. OCP), the platform high-speed I/O concerns can be easily separated.

OCP-IP INTERFACES

The Open Core Protocol (OCP) [5] is a socket interface which specifies the communication semantics between SoC blocks. It is supported by a growing industry consortium including integrated device developers, IP providers, EDA vendors and research institutions. Of particular interest, Xilinx, one of the primary FPGA suppliers, has recently joined the consortium. The protocol is defined by a small group of basic elements with optional extensions for enhanced capability. This flexibility allows the OCP to adapt to many different kinds of connections however, it also opens the possibility that two completely compliant cores cannot connect to each other. To address this possibility, the idea of profiles is introduced. An OCP profile defines a subset of the overall protocol. Certain configuration options are fixed to limit implementation overhead and ensure interoperability. Other options can be left open when a suitable default operation is defined. In effect, the communication uses the advanced features if both ends of a link implement them but automatically falls back to simpler behavior if either end does not. We have selected three profiles: dataflow, memory, and system. The dataflow profile emphasizes performance and simplicity. It is intended for a single direction of streaming data. The memory profile allows bi-directional flow and addressing of specific words. It is used for parameter/status interfaces as well as data transfers with arbitrary ordering. Neither of these profiles defines the format or meaning of the data being transferred. The final profile is used for system level control of individual modules. It layers a specific interpretation of the data on top of the basic OCP signals to implement common system level control operations.

With the interfaces to a module fixed to these profiles, generic system specific channels can be written to implement infrastructure connections without knowledge of the functions the modules perform. The FPGA is thus cleanly divided into independent waveform modules which implement the functional portion the design and infrastructure channels which implement the connections. This is analogous to the module/channel model used in our SystemC environment as we will cover below. Figure 3 illus-

trates the blocks which might be instantiated in a typical system. The infrastructure channels can leverage the unique capabilities of a system without affecting waveform portability. They also encapsulate hardware boundaries allowing flexible deployment of modules across FPGA resources. When porting an existing waveform to a new system, the waveform modules stay the same. Likewise, when porting a new waveform to an existing system, the infrastructure channels remain the same.

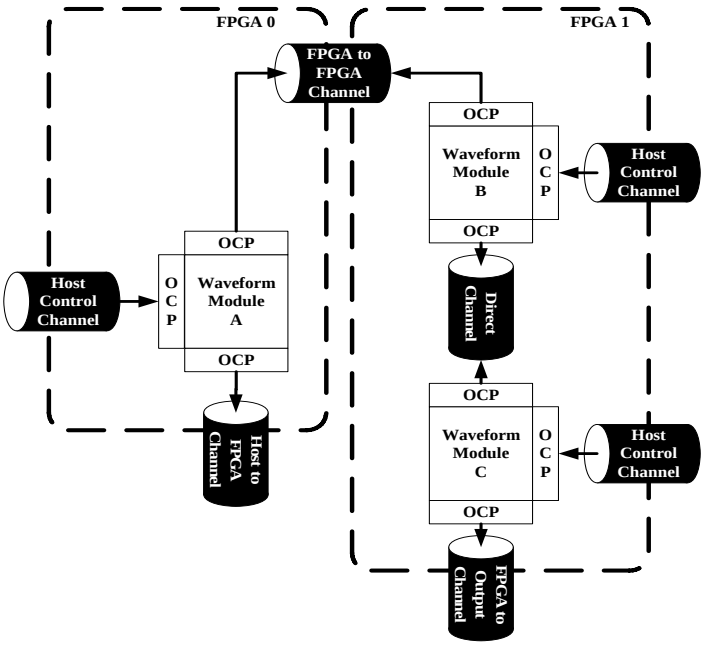

Figure 3: Example System Deployment Showing Waveform Modules and Infrastructure Channels

As mentioned previously, overhead is a major consideration for FPGA module implementation. The initial design of our test system suggests that the overhead incurred using OCP interfaces as described is low and comparable to that of any other connection technique including tailoring each module to connect directly to the next with a custom interface. For this implementation, the internal dataflow channels are designed to transfer a data word on every cycle after an initial latency so little performance degradation occurs.

Table I
OCP Area Overhead

Module	#Luts	#Flops
Direct Channel	34	40
FPGA to Output	69	77
Packetized Host Interface	74	64
Host Controller (base)	14	159
Host Controller (per module)	120	39
Module Overhead	30	55
Total (2 modules)	237	291
Portion of XC2VP100	0.3%	0.3%

Many variables affect actual area overhead. Table I shows the resource requirements of several blocks which might be found in a typical FPGA device. For this implementation, the off-chip connections group samples into formatted packets according to the protocol of the connection fabric. The direct channel is configured with a 2 sample buffer. The FPGA output uses a 1024 sample buffer. The data path and control path use 16 bit and 32 bit samples respectively.

LIMITING FPGA COMPONENT SIZE

Along with a standardized interface, another essential aspect of FPGA portability is the proper constraint of the resources used to implement an individual waveform component. This practice is commonly referred to as sizing. If, for instance, the waveform is to be ported to another platform, an appropriate sizing provides the ability to quickly repartition the waveform components across the new platform. However, blocks which are sized too small incur disproportionately large interconnect overhead; whereas, blocks that are sized too large may be forced to span FPGA device boundaries requiring a painful repartitioning effort. Yet another issue is the total FPGA device resource utilization. As more of the device is used, the place and route (PAR) tools may have difficulty converging, resulting in very long run times and possibly requiring manual placement. We find keeping the device utilizations below 60% helps with PAR turnaround time (TAT). Using readily available Xilinx Virtex-2 Pro (P100) devices, the following is a FPGA portability rule-of-thumb we find useful:

- Limit component cores to 25K LUTs, 2Mb BRAM
  - 2 components with ~50% total utilization on Virtex-2 P100
  - 3 components with ~60% total utilization on Virtex-4 FX140

Applying conservative device utilization rules helps decrease PAR TAT, however, it is still common to have multi-hour cycle times for each HDL change. Additionally, the number of FPGA devices per platform could add a significant multiplier to the PAR TAT. These facts

prompt us to look again towards SoC methodologies for solutions. Many FPGA developers avoid the effort required to use a robust simulation environment, preferring instead to rely on FPGA reconfigurability while testing directly on the host device. In the next section, we will argue the benefits of emphasizing simulation to reduce the overall code-to-test cycle time for HDL core integration.

FUNCTIONAL VERIFICATION: SIMULATION OR IN-SYSTEM

Due to the reprogrammable nature of FPGAs, some designers contend that a design can be verified primarily on the FPGA, since the FPGA can be loaded in a trial-and-error approach until a successful design is reached. This approach is acceptable when FPGA designs are simple, such as glue logic or interface translators, or perhaps when designs are small and run at a low rate, i.e. require less than 10% of the target FPGA's resources and operate at a clock frequency below 25MHz. However, the verification of the large, complex, high-performance designs that can span several of today's multi-million gate FPGAs must leverage more state-of-the-art techniques and resources in order for acceptable productivity to be sustained. Advanced verification techniques are the topic of much attention in the SoC community and are finding thorough treatment in technical references today [6], [7]. For SoC designs today, designers invest much time and effort to create a comprehensive verification plan encompassing every stage of the design, from the interpretation of the design specification to simulation and to deliverable testing, and take advantage of myriad tools and methods to ensure a successful design. At a minimum, the functional verification of designs that are intended to be reused should include dynamic simulation that covers a large extent (if not all) of the functionality of the design.

A thorough simulation provides the means for validating the retargeting of a design to another platform with disparate resources than the original platform. It is much more practical to verify modifications to an HDL design in a full-featured simulation environment than to churn in an in-system verification procedure. Furthermore, a designer will have a much higher confidence in a design if the design can be shown to operate properly upon delivery in the form of a simulation. A design without a working functional simulation carries a significant hurdle to reuse.

The reusability of a design is tightly tied to the means of verification that are developed for and delivered with the design. While some in-system debugging will always be warranted and in-system test is valuable for completing lengthy test scenarios, in-system operation should not be the primary means of design verification. All components of a design should be simulated before being loaded onto an FPGA, and the greatest emphasis should be placed on validating all interfaces and start-up operation. Once it has been verified that data can traverse through all interfaces in the design, then it may be reasonable to transition aspects of the design verification to the in-system FPGA. However, the functional simulations that accompany a design are the best means of verification of a design in reuse because the simulations can be common between the original design and the incarnation modified for the new platform.

Even though a design operating in an FPGA will run orders of magnitude faster than a simulation, the PAR TAT for even the smallest change in a design can be hours or even days. Then the design must be loaded onto the FPGAs (perhaps with internal probe points, which may alter the behavior of the design), and the behavior of the design must be determined to be valid with little observability into the real-time inner workings of the design. This can be a daunting task if there is no working simulation to serve as a point of reference. Also, having this in-system procedure in a regression test loop would be quite taxing, particularly when failures needed to be investigated.

The TAT of a well-designed simulation should be less than that for PAR and an in-system execution of the design. In addition, a simulation provides the benefit of complete visibility into the design, simulation licenses are usually cheaper than a target platform, and a working simulation of the design increases the value of the product, especially in terms of reusability. These benefits outweigh the cost of developing the simulation.

The increase in the capabilities of FPGAs has tended to outpace the capabilities of PAR tools. On the other hand, simulations using current, high-level languages keep better pace with the designs that leverage current FPGA capabilities. In order for such leading edge complex waveform designs to be affordably reusable, the capabilities and advantages of leading edge simulation-based functional verification must be harnessed.

WAVEFORM AND PLATFORM MODELING

The simulation of a waveform design includes models of the waveform and models of the platform on which the waveform will be implemented.

Performance models: One common method of modeling the waveform's modem signal processing is use tools such as Matlab or SPW. This modeling view is usually used by a system engineer to determine the performance characteristics of the processing on the signal-in-space (SiS). In many cases, these models are described to bit-level accuracy so they can be used to generate input test-vectors and expected values for the RTL simulation environment described previously. However, this modeling to simulation coupling method has a few limitations. First, revision control of the many files containing the test vectors becomes necessary as changes in the performance model will require regression testing. Depending on the complexity of the signal processing, for each test this could include managing many files which contain the intermediate results used to track down bugs.

Some vendors offer solutions to this issue by developing APIs which tie the modeling tools (e.g. Matlab) directly to the logic simulators (e.g. Modelsim). However, this technique could significantly slow down the overall simulation time which is undesirable in a regression testing environment.

C/C++ Bit-accurate Modeling: Most of the commercially available logic simulators can easily co-simulate C/C++ with VHDL or Verilog. This eliminates the need for test vector files by offering self-checking via comparison of the RTL and C model outputs. Also, a completely modeled system provides the ability to incrementally simulate finished RTL blocks in the context of the entire design rather than just applying individual unit tests. Depending on the level of abstraction, C model simulation can be orders of magnitude faster than Matlab and RTL-only simulation.

Virtual Platform: Since the platform-to-developer ratio is often much greater than one, we found it necessary to create an efficient development environment that provides parallel development capability in the context of the platform without requiring the actual hardware. Enabling a number of developers to target a virtual platform simultaneously is a significant benefit to project efficiency. By using an IEEE standardized language called SystemC [8], we have developed a way to accurately model the target platform in simulation. SystemC is now supported in all commercially available logic simulators.

Figure 4 shows a high-level diagram of our SystemC environment which is built on top of a commonly available logic simulator. Based on transaction-level modeling (TLM) techniques, the platform communication dependencies are abstracted using channel models and SystemC wrappers for the functional components (or worker functions). Much of the wrapper code can be automatically generated.

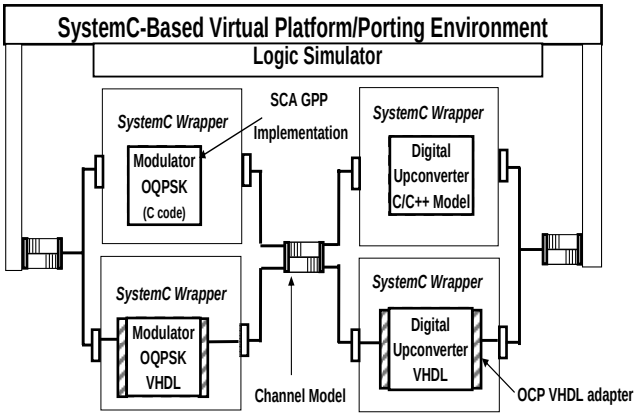

Figure 4: SystemC-based Virtual Platform

The channel models can be bi-directional and have the capability to model transport behavior at any level. Also, the channel models can be used to collect data, inject data, or self-check data at runtime.

Another benefit is the ability to provide co-simulation capability with the abstracted C-models and RTL. Adapters break message transactions into the signals required to exercise RTL designs. Selecting OCP as the interface to all RTL modules allows a common collection of adapters to be used for all modules, easing the movement between the platform and simulation environment. In essence, we have modeled our exact platform communication infrastructure to provide the necessary abstraction while properly simulating the entire waveform in a platform-independent way. Using this approach can significantly enhance the portability of any waveform.

It should be mentioned that, although the original intent was to promote portability for FPGA-based waveforms, we intend to extend this method to include C-based SCA compliant components by taking advantage of the C-based co-simulation capabilities of SystemC.

FPGA REUSE: A FULL METHODOLOGY

We have talked about several point strategies derived from SoC development methodologies; these included physical separation of platform and waveform concerns

using standard interfaces, proper limits on FPGA component sizes, modeling and simulation. Each method alone only offers a partial solution. Only by combining all these techniques into a single systematic approach can waveform reuse be fully maximized.

Today, in almost every case, in-house or 3rd party intellectual property (IP) cores are reused to accelerate the development of SoC devices. The IP core delivery mechanism is normally performed in two phases; a design view and implementation view. The design view is typically a platform independent, model and simulation environment that is used by the user to understand the functionality of the IP core. The implementation view would be all the necessary artifacts required to target the IP core towards a specific technology (or platform).

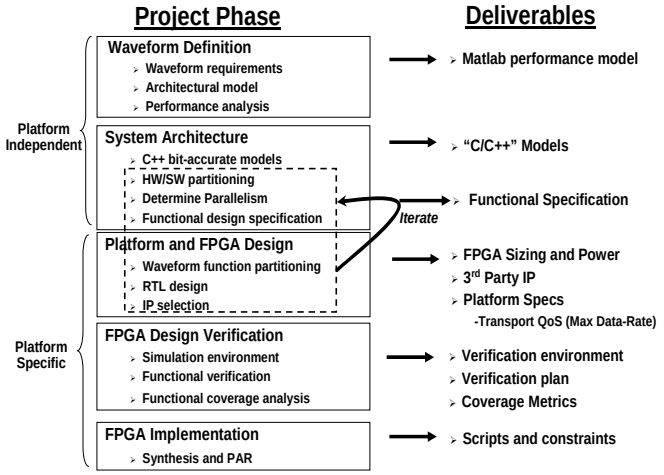

Figure 5: Layered Waveform Reuse Methodology

Figure 5 represents our waveform reuse methodology modeled after SoC IP core reuse. The layers represent abstraction levels in the waveform presentation. The higher the layer, the more abstract the waveform presentation. Notice each project phase requires a deliverable artifact. These deliverables are the important items that assist for later reuse. The top-layer represents waveform documentation. In our approach, executable behavioral models are a key part of the waveform documentation as paper documents are frequently misinterpreted. As one moves down the methodology layers, each presentation of the waveform moves closer to the actual implementation. The dotted box is highlighting the iterations normally associated with matching the platform resources to the waveform requirements (or moving from platform independent to platform specific implementation). This full approach is very important for reuse because it provides all the means to retarget the waveform to a new platform. Depending on the new platform differences, one moves up the layers as far as necessary to retarget the waveform for the new platform dependent implementation. For example, an extreme case would be when multi-core processors become readily available for SDR platforms this will cause the FPGA platform specific artifacts to become obsolete. Then having access to platform independent models becomes critical to understanding the waveform functionality. This provides the means to a successful and timely waveform port.

CONCLUSION

This paper has illustrated how modern SDR systems are becoming increasingly complex, particularly the role played by FPGAs and how SoC design techniques can be applied to more efficiently manage that complexity. A common interface separates functions from the system and each other; a simulation environment allows variations to be quickly explored and verified. The combination of these techniques works together to minimize the effect of changes on unrelated elements. Or, stated the other way, these techniques maximize the portion of a design which can be reused.

We have developed the framework for a virtual platform model to demonstrate our findings. It provides a flexible simulation environment with common block interfaces based on industry standard practices. Future effort will expand the framework to increase its ease of use as well as support additional features such as the integrated simulation of FPGA and GPP elements.

REFERENCES

[1] JTRS JPEO, SCA version 2.2, http://jtrs.spawar.navy.mil/sca/

[2] JTRS JPEO, SCA Specialized Hardware Supplement version 3.0, http://jtrs.spawar.navy.mil/sca/

[3] Object Management Group, PIM and PSM for Software Radio Components, 1st FTF Convenience Document, dtc/2005-04-02, April 2005

[4] http://fpgajournal.com.

[5] (Open Core Protocol International Partnership) [Online]. Available: http://www.ocp-ip.org

[6] J. Bergeron *Writing Testbenches: Functional Verification of HDL Models*. Norwell, MA: Kluwer Academic Publishers, 2000.

[7] J. Bergeron, E. Cerny, A. Hunter, and A. Nightingale *Verification Methodology Manual for SystemVerilog*. New York, NY: Springer, 2006.

[8] IEEE Standard SystemC Language Reference Manual, IEEE Standard 1666-2005, 2006