LEVERAGING SYSTEMC AND OCP TO IMPROVE THE VERIFICATION OF FPGA-BASED SOFTWARE DEFINED RADIOS

Joshua Noseworthy (Mercury Computer Systems, Inc., Chelmsford, MA, USA) jnosewor@mc.com

ABSTRACT

The design complexity of the modern day Software Defined Radio (SDR) is increasing as system designers continue to explore ways to successfully integrate Field Programmable Gate Arrays (FPGAs) into SDR-based systems. One of the many challenges associated with the integration of FPGAs into SDR-based systems is verification at both the component and system levels. Unlike software-based components which can rely on common interface standards such as those set forth by the Software Communication Architecture (SCA), FPGA-based components have no common standard upon which to draw. This makes it extremely difficult to leverage verification Intellectual Property (IP) assets across multiple FPGA-based components since each component could potentially have a unique interface. The result is a time-consuming process of customizing verification IP to suite the needs of a particular FPGA component interface.

Another challenge of introducing FPGAs into SDR-based systems is verification at the system or application level. Typical systems are heterogeneous, containing a mix of FPGAs and other devices operating at varying levels of abstraction. This degree of heterogeneity makes the provision of a single verification environment, in which both hardware and software elements can be verified, difficult.

These challenges can be alleviated through the introduction of modeling languages, such as SystemC, that are capable of supporting varying levels of modeling abstraction and standardized interface techniques for FPGA components, such as those described by the Open Core Protocol (OCP). SystemC provides a C++ based standard for the specification of both hardware and software systems at varying levels of abstraction. SystemC is capable of modeling at both the abstract transaction level and the Register Transfer Level (RTL), as well as at levels in between. Many commercially available hardware description language (HDL) simulators offer mixed-language environments where SystemC models can interact with both VHDL and Verilog models. This creates a powerful verification environment that enables designers to fully test and verify their synthesizable RTL together with C/C++ based structures. In addition, this type of environment allows for the co-verification of the application's FPGA and software-based components prior to the application's actual deployment.

1. INTRODUCTION

1.1 The JTRS Communication Architecture

The Software Communications Architecture (SCA) specification establishes an implementation-independent framework with baseline requirements for the development of Joint Tactical Radio System (JTRS) software defined radios (SDRs). The requirements include both interface and behavioral specifications that ensure the maintenance of portability and configurability across vendor platforms.

The operating environment mandated by the SCA includes a real-time, POSIX-based operating system, minimum CORBA support, and the Core Framework. This type of environment cannot be realistically supported on specialized hardware devices such as digital signal processors (DSPs) and field programmable gate arrays (FPGAs). A standardized way in which the goals of the SCA can be achieved for DSPs and FPGAs is still forthcoming.

1.2 Open Core Protocol

The Open Core Protocol (OCP) delivers a non-proprietary, openly licensed, core-centric protocol that comprehensively describes the system-level integration requirements of intellectual property (IP) cores. OCP eliminates the task of repeatedly defining, verifying, documenting and supporting proprietary interface protocols by providing a standard way to specify point-to-point interfaces between two communicating entities. A clear advantage to using OCP to describe a core's interface(s) is that the mechanisms through which one OCP interface can talk to another are clearly defined by the OCP specification. Even if two connected cores have dissimilar interfaces, the fact that they are valid OCP interfaces means the information needed to resolve those dissimilarities is readily available.

OCP interfaces are specified by their OCP profiles. An OCP profile is a collection of OCP parameters and their associated values. Changing the values associated with OCP parameters results in either the inclusion or exclusion of specific OCP signals and/or changes their behavior. A designer can completely reconstruct the interface given the interface's profile and vice versa. Profiles are most easily described using an XML schema. However, at the present time a standardized schema that is recognized by OCP is forthcoming.

1.3 SystemC

SystemC is a collection of C++ libraries and macros that facilitate the modeling and simulation of concurrent processes. SystemC enables designers to model complete systems at varying levels of abstraction. Levels of abstraction correlate to the accuracy of the model with respect to the number of clock cycles that have elapsed. The highest level of abstraction is the least accurate with respect to timing. At this level, the model assumes that an infinite amount of time exists between two successive clock cycles.

The lowest levels of abstraction are the most accurate with respect to timing. Models are constructed such that the amount of work performed on a cycle by cycle corresponds to the amount of work that would be expected from an identical time lapse in hardware.

The principle use of SystemC surrounds the development of verification assets, though a synthesizable subset of the language does exists. These verification assets are most often developed to enable the functional verification of a device under test (DUT). The verification assets can exist at almost any level of abstraction as what is most important is that they are capable of mimicking the behavior of the device that will connect to the DUT when it is implemented in hardware. In most cases, it is to the benefit of verification engineers to develop their assets at the highest level of abstractions as this will result in the fastest simulation speed.

1.4 Transaction Level Modeling

Transaction Level Modeling (TLM) is a technique used to model communications between two interconnected modules at very high levels of abstraction. In this model, transactions being communicated between the two entities are completely decoupled.

Figure 1 shows an example of a typical transaction-level model. Communications between the bus functional model and the DUT are mediated through the TLM's channels. The BFM generates transactions over the channel through the use of very simple put/get mechanisms as shown in Figure 2.

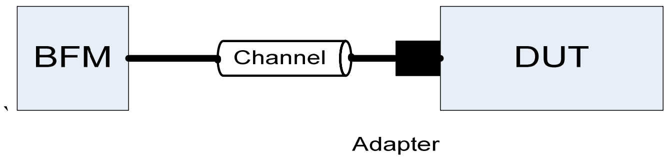

Figure 1: TLM Example.

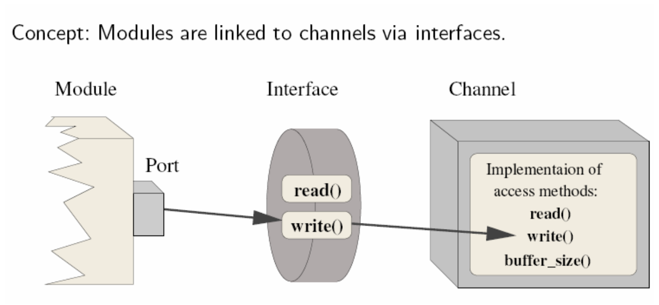
Concept: Modules are linked to channels via interfaces.
Module
Interface
Channel
Port
read()
write()
Implementaion of access methods:
read()
write()
buffer_size()

Figure 2: BFM Module talking to channel.

The channel communicates these transactions to the DUT through an adapter. The adapter translates generic transactions that have been communicated over the channel into protocol specific transactions.

SystemC lends itself to the implementation of TLMs as most of the abstract data types necessary to do such an implementation are part of the C++ standard template library. The channel itself need not be anything more than a simple request FIFO where the BFM module is responsible for pushing requests into the FIFO, whilst the DUT is responsible for pulling them out. Similarly, the DUT is responsible for pushing responses into a response FIFO, whilst the BFM is responsible for pulling them out.

The use of a TLM has several benefits associated with it. First, the verification engineer need not be concerned with the generating protocol-specific transaction in order to communicate with the DUT. Instead, the engineer need only understand the simple put and get methods that are necessary in order to access the channel. The TLM takes care of the rest.

Second, most protocol specific transactions are associated with some degree of overhead. This overhead includes pack headers, and/or extra signaling that is necessary to drive the interface. The use of TLM enables transactions to be communicated over the channel using the least amount of information possible. Immediately before the presentation of the transaction to the DUT, the adapter translates the transaction into protocol specific signaling.

Finally, TLMs create an opportunity for the existence of a single verification environment that services multiple

IP modules. If the interfaces of the modules differ, the engineer need only change the adaptor to perform the appropriate translations. This process can be further simplified through the adoption of standard IP interfaces. Doing would enable a single BFM

2. OCP-BASED VERIFICATION ENVIRONMENTS

As systems become increasingly complex, the cost of developing verification assets increases substantially. This increase in cost is a result of the amount of effort required to develop more sophisticated system models. These cost increases can be softened through leveraging verification assets over multiple projects. This is often hard to do as most IP interfaces are often not consistent across multiple projects. As a result, verification assets that were used on previous occasions may require substantial rework before they are suitable for use on new projects.

The adoption of standard interfaces, such as those defined by OCP, enables designers to maintain interface consistency across every IP module that is designed with interfaces that conform to the chosen standard. As a result of this consistency, designers can begin to leverage IP across several projects. OCP makes this possible by limiting the number of forms a particular interface can assume. The fact that an interface contains a limited number of signals, each having well-defined behavior, facilitates the specification of the interface through a collection of parameter values. This collection of parameter values, known as a profile, completely specifies the inclusion or exclusion of specific OCP signals within a particular interface, as well as the behavior of each signal within the interface.

Given a limited set of interfaces and a suitable mechanism for specifying the appearance and behavior of the interface, it becomes reasonable to expect that a verification engineer could develop a single verification environment that is capable of conforming to any valid OCP interface. Such an environment requires that upon instantiation of the environment, the profile is analyzed and the environment is adjusted to suit the needs of the particular OCP interface. This is exactly what the OCP TLM does.

2.1 OCP Transaction Level Model

The OCP TLM is a collection of C++ source and header files that implement an OCP-based TLM. The TLM exists to facilitate the simulation of modules that interconnect with other modules, memory, and/or system buses.

The OCP TLM defines four levels of abstraction: Layer 0, Layer 1, Layer 2, and Layer 3. For the purpose of this discussion, we will only consider Layer 0 and Layer 1. Layer 0 is considered nothing more than pure RTL. Layer 1 is said to be cycle true but faster than RTL. It provides the user with a set of SystemC-based APIs that can be used to access the channel. For example, if the user wants to communicate a request over the channel, the engineer need only call the method putOCPRequest(). The user need not be concerned with the assertion of the correct signals as if the work were being done at Layer 0.

In many instances, there is a need to initiate a request at Layer 1, and then have it communicated to a module that is running at Layer 0. This scenario is almost always the case when a BFM is being used to verify the functionality of a DUT. To accomplish this, the OCP TLM requires the use of a layer adapter. The layer adaptor accepts Layer 1 requests and then in turn generates the equivalent Layer 0 request. Responses are similar except that they originate in Layer 0 and are converted into Layer 1 transactions. This process is summarized in Figure 3.

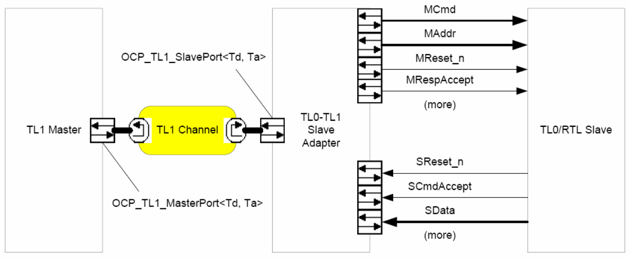

Figure 3: L0/L1 Slave Adapter Connectivity.

2.2 Mixed Language Simulation Environments

The vast majority of commercially available HDL simulator support mixed-language environments. Mixed-language environments support the interconnection of VHDL, Verilog, and SystemC modules. Such an environment enables synthesizable VHDL and/or Verilog to be functionally verified in a C++ based environment. This is a powerful capability as the level of effort to build a C++ based verification environment is a fraction of what would be required to build the same environment in either VHDL or Verilog. Coupled with the use of standard OCP interfaces, the OCP TLM has the potential to fulfill the needs of functional verification so long as the designers that are designing the modules are willing to begin designing using standard OCP interfaces.

3. COVERIFICATION USING SYSTEMC AND OCP

The verification of an SDR application can be a challenging task as an increasing number of application are beginning to use both hardware and software components. Traditionally, co-simulating hardware and software has been a difficult task as the number of tools that provide a means of doing so are fairly limited. This often leads designers no other choice but to independently verify each component with the hopes that things will just work when the application gets deployed as a collection of verified components. Often times this is not the case.

Furthermore, it is more often the case that an application is being developed prior to or independently of any available hardware. Without an adequate co-simulation environment, there is no mechanism that enables these designers to accurately observe component interactions prior to the availability of hardware. Since hardware availability is often at the latest stages of many projects, the existence of an adequate co-simulation environment would enable designers to retire the risks associated with an application's deployment long before the hardware is made available. Retiring such risks earlier in the design process will facilitate the integration processes that occur once hardware does become available.

Using SystemC based verification environments for SDR platforms have the potential to enable the co-verification of both hardware and software components within the same environment. Since SystemC is a C++ based environment, it becomes possible to execute the software infrastructure, including CORBA, within the hardware simulator as one or more SystemC modules.

An alternative approach to running the entire infrastructure inside of the simulator might be to run a CORBA server that accepts requests from other processes and then communicates them to the hardware components running in the simulator. The software infrastructure runs outside of the simulator, as it would in an actual system, communicating with hardware only when it needs to. This latter model has the potential to result in faster simulations.

4. CONCLUSION

This paper has highlighted the advantages of standardizing component interfaces using OCP and how it might affect verification. We have seen that the use of standardized interfaces for FPGA components satisfies many of the needs of modern day communication systems including reusability and portability across multiple platforms.

In addition, we saw how SystemC and OCP can be used to create powerful verification environments that are leveraged across multiple projects. This reduces cost, decreases time to market, and results in better verification assets over time.

Finally, we examined how SystemC can be used to create powerful co-simulation environments for verifying interactions between hardware and software components. Using standard hardware interfaces, such as OCP, is critical for the success of these environments as the majority of companies could never afford the costs of re-customizing the environment's interface to support each application.

5. REFERENCES

[1] M. S. Programmable Radio Consortium, "Software Communication Architecture specification," November 2001, v2.2.

[2] T. O. M. Group, "Common object request broker architecture: Core Specification," March 2004, v3.0.3.

[3] O. I. Partnership, "Open core protocol specification," 2005, v3.1 Available: http://www.opc-ip.org.

[4] J.T.R.S.J.P. Office, "Extension for component portability for specialize hardware processors," March 2005, v3.1.