IMPLEMENTING SCA COMPONENTS ON FPGAs

Joshua Noseworthy (Mercury Computer Systems, Inc., Chelmsford, MA, USA; jnoseworthy@mc.com); James Kulp (Mercury Computer Systems, Inc., Chelmsford, MA, USA; jkulp@mc.com)

ABSTRACT

The design complexity of modern day field programmable gate array (FPGA) systems is increasing as system designers are forced to integrate more FPGA devices into a single system in hopes of meeting the demands of today's computationally intensive applications. The problem is further worsened by applications such as software defined radios that not only demand high levels of performance, but also high degrees of portability and reconfigurability. Satisfying these requirements, while still maintaining a reasonable time to market makes full custom designs for FPGAs nearly impossible. One technique to alleviate a significant percentage of these burdens is to design intellectual property (IP) cores to conform to specific interfaces that can be parameterized to suite the particular needs of the IP core. Our experiences show that a large percentage of core communication patterns can occur through a select number of interfaces provided that the interfaces have customizable attributes that allow them to be made specific to a particular application. This technique eases design complexity since designers no longer spend time learning interfaces specific to a single core. Furthermore, if such guidelines are followed, the connection of cores instantly becomes a well understood problem that yields a finite number of efficient solutions. Change Proposal 289 (CP289) addresses these issues by specifying three profiles to describe the various types of component communications. In this paper we describe the CP289 specification and how using the three profiles can facilitate the design of CP289 systems. The concepts presented are in no way specific to CP289 and can easily be extended to incorporate other FPGA component models.

1. INTRODUCTION

Advancements in silicon technologies continue to fuel generations of field programmable gate arrays (FPGAs) that are capable of delivering unprecedented levels of performance. Coupled with their high level of reconfigurability, and relative low cost, FPGAs are attractive solutions for problems of the highest computational complexity.

Once used exclusively for rapid prototyping, FPGAs now work side-by-side with digital signal processors (DSPs) and general-purpose processors (GPPs) to deliver some of the world's highest performance computing solutions. As the complexity of these systems continues to increase, designers are continually faced with new integration challenges that exist both on and off chip.

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 defining a collection of signals and parameters that can be use to move a specific type of data through a particular interface [OCP]. 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.

The ability to understand how two cores communicate, given a set of parameters, is extremely powerful, especially when it comes to building networks that interconnect multiple cores. Since OCP defines exactly what each parameter means, it becomes possible to build utilities that generate an interconnection network for the designer, even if the interfaces for each node on the network are dissimilar. This is possible because OCP defines the behavior for each signal. Therefore it is possible to introduce intelligence into a utility that will generate resolution functions for dissimilar IP, assuming that an appropriate resolution exists. For instance, consider a scenario where two cores, A and B, want to connect. The problem is that core A has a 64-bit data path and core B has a 32-bit data path. Since the width of the data path can be expressed via an OCP parameter, a utility can be engineered to make a decision about how to

 Copyright © 2006 SDR Forum. All Rights Reserved

connect a 64-bit core to a 32-bit core. The ability to automatically generate an interconnection network of this nature has the potential to abstract a significant portion of a design's complexity away from the designer, thus decreasing a design's time to market.

In this paper we present three OCP profiles we believe are sufficient to satisfy the communication requirements for the majority of modern-day FPGA cores. The three profiles include the worker control profile (WCP), the block dataflow profile (BDP), and the streaming data flow profile (SDP). Finally, we show that adoption of these profiles facilitates the development of large-scale FPGA systems. We do all of this in the context the Joint Tactical Radio System (JTRS) Software Communication Architecture (SCA), although all concepts presented can be further generalized to support almost any designed process.

2. BACKGROUND

2.1 The Open Core Protocol (OCP)

OCP defines point-to-point interfaces between two communicating entities. One entity defines an interface to act as a master, and the other defines an interface to act as a slave. The master is the controlling entity and is the only entity with the capability of issuing commands. A slave responds to commands either by accepting or responding to commands that have been issued by the master.

Given the wide range IP core functionality, performance, and interface requirements, a fixed-definition protocol fails to address the vast spectrum of suitable core requirements. Furthermore, the ability to support sufficient test and verification practices adds an even higher level of complexity. To address these requirements, OCP supports highly configurable interfaces. Using these highly configurable interfaces, a designer can tailor an instance of an OCP interface to a specific application. The ability to tailor an OCP interface to meet the needs of a specific application is provided through the use of OCP parameters. A collection of OCP parameters is referred to as an OCP profile. These parameters indicate to the designer which OCP signals need to be included inside of a core's OCP interface. In addition, the inclusion of certain parameters within a profile can be used to imply behavior, as well as indicate signal properties, such as a signal's bit-width.

2.2 The JTRS Communication Architecture

The SCA specification establishes an implementation independent framework with baseline requirements for the development of JTRS software defined radios [1]. The requirements include both interface and behavioral specifications that ensure the maintenance of portability and configurability across vendor platforms.

The SCA uses the Common Object Request Broker (CORBA) to provide a common language through which all distributed elements (components) within the system communicate. Although CORBA has been adopted as a middleware for GPPs, CORBA implementations for processing elements such as DSPs and FPGAs are far less common.

Change Proposal 289 (CP289) is intended to address components that cannot use the portability requirements defined in section 3.2 of the SCA specification. Specifically, it defines specific portability requirements that target resource constrained environments such as DSPs, FPGAs, and Application Specific Integrated Circuits (ASICs).

Portability in the SCA implies that a SCA component should be capable of executing on any system that has been deemed SCA compliant. Meaning that if a software component connects into System A, it should also connect into System B provided that System A and B are both SCA compliant. This example ignores any differences in the underlying architecture of the processing element responsible for executing the component's functionality. However, from an interface standpoint, component instances in either system should be identical. This is an important facet of the SCA as it allows components to be obtained from various sources and then seamlessly integrated into a single system.

For FPGAs, the mechanisms that enable portability to be maintained are provided by OCP. Each FPGA component must contain interfaces that operate in accordance with the WCP, BDP, and SCP profiles. If these details are maintained for each FPGA component implementation, then CP289 says the component implementation can connect into any SCA system that is CP289 compliant.

In addition, CP289 defines additional protocol semantics that specify how specific types of information can be communicated through an interface that identifies itself with a particular OCP profile. For instance, according to CP289, a read command on an interface that conforms to WCP will put the worker in an operating state. These additional semantics are not part of the OCP specification, but have been introduced by the authors of CP289 in order to further define how information on an FPGA should be communicated if a system is to be SCA and CP289 compliant. These additional semantics further simplify SCA application development because SCA utilities have prior knowledge of how data is communicated to a specific component. With this knowledge, the utilities can generate software libraries that facilitate access to the applications underlying hardware.

Proceeding of the SDR 06 Technical Conference and Product Exposition. Copyright © 2006 SDR Forum. All Rights Reserved

This dramatically improves the development experiences of the application developed as the designer needs only to understand how to use a set of software libraries. The tools determine the details of how the libraries interact with the physical hardware. This scenario is made possible not only because OCP provides a set of constrained signals that have a well-defined behavior, but also because using a constrained set of signals enables a designer to write well-defined rules on how to connect OCP interfaces that associate with dissimilar parameter sets.

3. CODE GENERATION FOR SCA COMPONENTS RUNNING ON FPGAs

In the remainder of this paper we describe an ongoing effort at Mercury Computer Systems, Inc. to develop a model for components that execute on FPGAs. To date, most of the work has been fueled by our efforts to develop a CP289-compliant system. However, this work is by no means limited to CP289. In fact, it is our intent to introduce these same concepts into future releases of existing solutions, such as Mercury's FPGA Development Kit (FDK). Ultimately, introducing these concepts into existing processes will alleviate a large percentage of the design effort that is currently expected from designers. In addition, the automation of specific processes will result in a more efficient design process that is less susceptible to error and more forgiving of future changes. This will significantly improve the overall design experience.

The following sections examine two key concepts that are critical in the development of CP289-compliant systems. The first being the process of building CP289-compliant components and the second being how these components interconnect. The former concept is captured by a software utility described below.

4. CP289 COMPONENT CODE GENERATION FOR FPGAS

4.1 Software Component Descriptors

CP289 components are described at the highest level by their Software Component Descriptor (SCD). Each SCD describes the ports that are used by the component to send and receive data from other entities within the system. Included in this description is a port name, repository ID, and type. The port type either provides or uses depending on the responsibilities the port assumes. The port is a provider if it assumes the responsibilities of servicing requests and issuing responses. Similarly, the port is a user if it assumes the responsibility of issuing requests and servicing responses.

The SCD also specifies the location of the component's property file. The property file associates a collection of properties with a specific component. Each property consists of a name, a value, and the value's type. The value's type is used to determine the amount of memory that must be allocated to accommodate a worker's property space, as well as the offset into that memory that should be used to access the property value. The repository ID indicates the location of the ports interface description within the interface repository. The worker developer uses the repository ID to query the interface repository about the interfaces the worker needs to support.

4.2 Interface Definitions

Each SCA component port must be accompanied by an interface definition. The interface definition describes how the interface through which the port can be accessed. Specifically, it describes the types of data and operations that an interface to a specific port can support. The interface definitions reside in an interface repository (IR). The IR is a CORBA object that acts as a container for interface definitions. Our implementation approach uses the IR to store the interface definitions associated with the RPL components being developed. For a detailed explanation of the IR, the reader is referred to [2]. It should be noted that the SCA specification does not reference any specific IDL files. Therefore, there is no standard way of using the SCA meta-data to discover the appropriate IDL files.

4.3 Inferring OCP Configurations for Component Ports

The list of the ports belonging to a component is defined in the SCD. Each port must be associated with a name, type, and repid. Figure 1 shows how this information is used to generate the appropriate OCP interfaces. Each component port corresponds to at least one OCP interface that uses a Worker Port Profile. For each component port, an IR lookup is done using the ports repository ID. The IR lookup determines if the ports associated interface definition contains one-way operations. If the ports associated interface definition contains at least one one-way operation, then two interfaces must be created to support the port. The first interface enables the communication of the operation itself, while the latter enables the return values, output arguments, or exceptions to be communicated back to the requesting entity.

4.4 Generating an OCP Configuration

The result of this process is an OCP configuration that describes the interfaces through which data is communicated between the component and the immediate

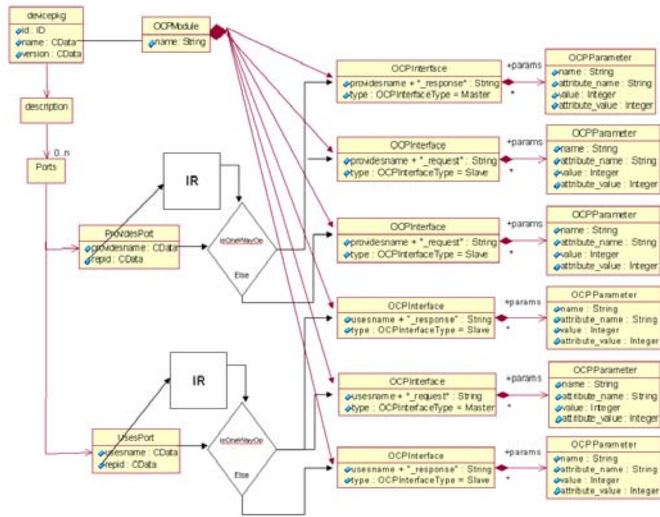

Figure 1. Generating OCP-Component Interfaces.

surrounding environment. Given an OCP configuration the tool is capable of generating an OCP core configuration file. This file is required to maintain OCP compliance.

In addition to the OCP core configuration file, the tool generates a collection of HDL specific files. These files specify the components top-level signals. The component developer uses these files as the starting point. The developer is required to populate the skeletons with the RTL that best describes the components functionality.

4.5 Future Work

In the future we plan to introduce capabilities that will enable the tool to introduce limited functionality into the component skeletons it creates. The functionality could include a sizeable percentage of the logic necessary to manage communication through specific OCP interfaces. Interfaces that conform to the WCP must be able to discriminate between read and write commands (among others things). State machine(s) that would allow the component to do so could be included as part of the RTL code that is generated automatically. In addition, the potential exists for this tool to expel software header files that would facilitate access to the component. Specifications such as CP289 can specify protocol semantics that define how data is communicated through a particular interface. For example, CP289 specifies that control operations are communicated to components via read transactions through a WCP interface. The responsibility of the component is to execute a control operation when it has been instructed to execute a read at a specific address. A software header file facilitates the executions of these control operations by providing an application developer with data structures that list each control operation and its associated address. This way developers need not concern themselves with generating the address mappings for specific components. The tools do it automatically. The developers need only to be concerned about ensuring the application includes the appropriate header files.

5. ON-CHIP INTERCONNECTS FOR CP289

The creation of CP289 guarantees FPGA-component portability across compliant systems. This guarantee can be maintained only through adherence to the OCP profiles that CP289 defines. The use of OCP as a means to describe a component's interface to the external world has advantages that extend beyond the maintenance of component portability. The absence of standardized interfaces, such as OCP, results in companies developing their own proprietary interfaces, soon to be followed by their own proprietary networks.

Developing networks that interconnect cores is a task that could easily lend itself to machine automation. The reason why it doesn't is because the interconnection of non-standardized interfaces is next to impossible. The solution space is nearly infinite. The use of OCP as a standard interface constrains that solution space, making machine automation a real possibility.

Each OCP profile defined in CP289 specifies a finite number of signals, each having a well-defined behavior with a limited amount of parameterization. The fact that the behavior and parameterization of specific signals is limited makes it possible to define specific rules for resolving connections between two dissimilar instances of the same OCP signal. The rules that provide these resolutions are easily interpreted by a machine. In the paragraphs to follow, we will explore what can be done to automatically generate an interconnect network that is capable of communicating control and status information to WCP slave entities.

Figure 2 shows an example of the internal control ring (ICR) that distributes control and status information to CP289 compliant components. The network consists of a single master and N nodes. A worker control interface (WCI) master contains an interface that conforms to the WCP, hence it is called WCI. Each slave on the network is identified by a unique address map. When the master issues a request, the first slave connected into the network examines the address that has been associated with that request. If the slave recognizes the address that has been presented to it, the slave absorbs the transaction away

Proceeding of the SDR 06 Technical Conference and Product Exposition. Copyright © 2006 SDR Forum. All Rights Reserved

from the ring. Otherwise the data is communicated to the next slave connected to the ring. This process repeats itself once for each slave in the network. If the master receives the request back, an error bit is asserted. This bit indicates that there was a problem executing the request.

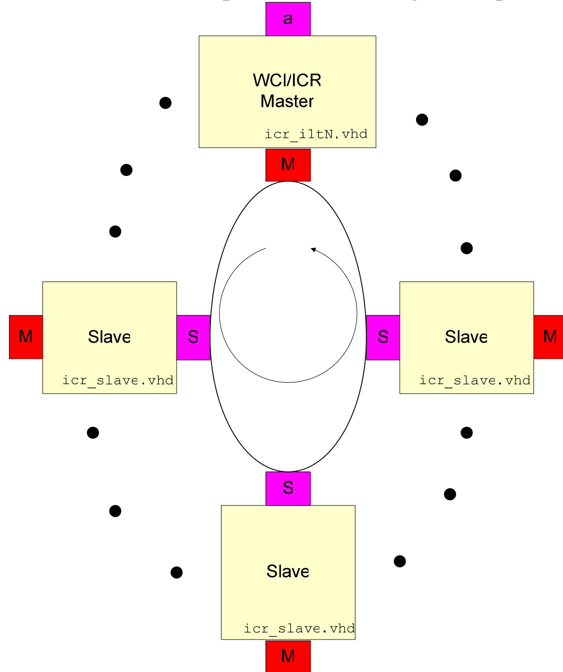

Figure 2. Worker Control Interface (WCI) Internal Control Ring.

A WCI transaction requires multiple clock cycles to complete. The exact number of required clock cycles is dependent on how many nodes have been connected to the network, as well as the width of the interconnect itself. Each slave on the network increases the network's communication latency by one clock cycle. This is because the insertion of a slave entity will either increase the number of paths that a request must traverse by one, or increase the number of paths that a response must traverse by one.

The communication latency is also affected by the value of the parameter that indicates the width of the ring. A single WCI transaction requires 72 bits be communicated. This requires 9 clock cycles when the width of the network is 8 bits.

Building a WCI network is a matter of instantiating multiple OCP interfaces and then specifying connections between them. This is easily performed in software. The tool requires that each node instance be accompanied by an XML file that describes specific pieces of information about the slave. This information includes the size of the address space required to access the slave and the parameter values that are specific to the interface instance. A similar XML file describes the interconnection of all of the components.

The software looks at the XML file for each slave in order to determine where each slave should be connected into the ring and into what address space it should be mapped. If two dissimilar interfaces need to connect, the tool generates IP suitable for resolving the dissimilarities for two connected interfaces. It should be noted that the tool considers two interfaces dissimilar if the two interfaces share the same profile, but specify unequal parameter values.

The output of this process is an HDL-specific, top-level description. The description instances a single master and multiple slaves. Each node is instanced with parameter values that are in accordance with the parameter values specified by the node's XML file. The description can be synthesized for download onto a specific FPGA device.

In addition to the top-level HDL files, the tools are capable of producing header files that facilitate access to the network via software. These header files contain data structures that define the address ranges into which specific components have mapped. Once again, the application developer need only be concerned with including the appropriate header files into the application.

The state of our work is such that we have prototyped software to perform the aforementioned task using non-standardized XML formats. The full utility of this concept will not be apparent until there is a way to express OCP-related meta-data in a format that is understood by all. The current way of doing this is through the use of an OCP core configuration file. However, the file format that is used for the configuration file is intended to be read by humans, and thus requires a significant amount of work to parse by machine. This simply won't work. OCP, along with other standards organizations such as IP-XACT, are looking at formats that would capture a core's meta-data in a format that is more easily manipulated by machine.

6. CONCLUSION

Maximizing component portability and reuse is a critical part of accelerating the development of today's high-performance applications. Standardizing component interfaces is one technique for doing this. We have seen that standardizing component interfaces using standards such as OCP eliminates proprietary core interfaces by specifying a set of signals with well-defined behaviors. This maximizes portability since systems and components that conform to specific OCP interfaces will be interoperable. This is critical in application domains-, such as software defined radios, where component and system developers are almost never the same people.

A side effect of using standard interfaces is that they limit the ways in which they can be interconnected. This

 Copyright © 2006 SDR Forum. All Rights Reserved

finite solution space allows for the automatic integration and configuration via standards such as IP-XACT. IP-XACT is a standard that enables one to describe connections between OCP interfaces through the use of schema, such as XML. Software parses the XML and then generates the appropriate interconnect base on the parameters that had been defined by the XML schema.

7. Acknowledgment

The authors would like to recognize Shepard Siegel of Mercury Computer Systems, Inc. for his tireless efforts in advancing this initiative.

8. References

[1] "Software communication architecture specification," *M. S. Programmable Radio Consortium*, November 2001, v2.2.
[2] "Common object request broker architecture: Core Specification," T. O. M. Group, March 2004, v3.0.3.
[3] "Open core protocol specification," O. I. Partnership, 2005, v3.1 Available: http://www.opc-ip.org.
[4] "Extension for component portability for specialize hardware processors," J. T. R. S. J. P. Office, March 2005, v3.1.

Proceeding of the SDR 06 Technical Conference and Product Exposition. Copyright © 2006 SDR Forum. All Rights Reserved