OCP2XI Bridge: An OCP to AXI Protocol Bridge

Zdravko Panjkov[1], Juergen Haas[1], Martin Aigner[1], Herbert Rosmanith[1,2], Tianlun Liu[1], Roland Poppenreiter[1], Andreas Wasserbauer[1], and Richard Hagelauer[1,2]

[1] Intel Mobile Communication (IMC), Danube Mobile Communications Engineering
Zdravko.Panjkov@intel.com
[2] Institute for Integrated Circuits, Johannes Kepler University Linz

Abstract. The modern SOC design contains many IP cores with different communication protocols. Improving the bridging and signal translation between these protocols has become a critical factor for the performance of the whole system. In this paper we will address the bridging of two well defined protocols, the Advanced Extensible Interface (AXI) and the Open Core Protocol (OCP). This bridge supports pipelined and multiplied transactions from both AXI and OCP interface. In comparison to related work our bridge offers simpler implementation and handling while containing full protocol functionality. The bridge is implemented and verified in a modern SystemC regression environment with large functional coverage. FPGA emulation is done on Versatile Express board using the CPU board as a main emulation controller. The result shows that the bridge is covering full protocol functionality and that maximal FPGA frequency is acceptable for a wide range of applications.

Keywords: AXI, OCP, FPGA.

1 Introduction

With the increase in process integrity and frequency, the amount of different protocols used in SOC increased substantially. The result is that the configurability and reusability of the chip protocol to the different bus types has become a dominant factor.

In the past, the widely accepted bus protocol was AMBA AHB [1], which facilitates single and burst transactions by using shared multiplexer architecture. The multiplexer architecture performs well with a limited number of IP cores, but with the increase number of cores the amount of transactions increased to more than one transaction at the time which is not supported by the shared architecture. Precisely for this reason two new protocols have been developed: the first is an Advanced extensible Interface (AXI) protocol, and the second is an Open Source Protocol (OCP).

The Open Core Protocol organization (OC-IP) started work in 2001 on something that would eventually become an OCP protocol [2]. OCP-IP aim was to

D. Goehringer et al. (Eds.): ARC 2014, LNCS 8405, pp. 179–190, 2014.

create a standardized interface and thus simplify the SOC's integration problems. The most important benefit of the OCP is its flexibility and configurability. OCP can be configured as a simple peripheral core, a high performance microprocessor or a chip subsystem.

AXI was first launched in 2003 with AMBA 3 architecture [3]. A protocol defines five separated channels with a separate set of signals for each channel. The interface is not restricted by the internal bus architecture so the designer can integrate different IP's by direct connection or by bus infrastructure between them.

This paper presents our experience and methodology with designing, verifying and emulating the bridge between OCP and AXI protocols. Verification is done in a state of the art SystemC verification environment with a wide range of cases and with high functional coverage. For emulation, the bridge is synthesized for Xilinx Virtex6 FPGA, which is part of the Versatile Express board. The entire verification environment is ported and implemented on Versatile Express board which gave a significant increase in verification speed. We showed that this bridge can handle many real time applications such as high definition images, different types of data or even a slow stream transmission.

The key contributions of this paper are:

- The paper presents the design of a full-featured OCP to AXI bus bridge, which is verified in the SystemC verification environment.
- The paper presents our FPGA synthesis methodology for OCP to AXI bus bridge and the porting of whole SystemC verification environment to an emulation board.
- The paper demonstrates how the above bridge can handle real time application directly on FPGA emulation board.

The remainder of the paper is organized as follows. Section 2 reviews related work and provides backward information on the bus bridge. Section 3 describes our experience in designing the bridge. Section 4 describes the verification environment used to verify protocol bridge. Section 5 presents emulation board and procedure to create an emulation environment. Section 6 present results, functionality and performance of the synthesized bridge. Section 7 concludes.

2 Related Work

There are commercially available software tools and solutions for protocol bridging.

The most used software tools are the Sonics Express [4] and Arteries FlexNoC [5]. Sonic Express provides a high bandwidth bridge between AXI and OCP protocols with the capability of crossing clock, power and physical boundaries. Arteries FlexNoC can implement a bridge between any combination of AMBA, AXI, AHB, APB and OCP protocols. The only problem of these tools is their price and usually quite complicated GUI which adds additional complexity.

Beside software tools there are also commercial IP solutions. MIPS-Imagination offers OCP to AXI Bridge [6]. The MIPS IP is relatively specialized as it only

allows the connection of OCP 32 bit width interface to AXI bus. This makes sense because the bridge was developed for MIPS32 processor but it is not applicable in any different solution. Xilinx offers many different protocol bridges as part of the Xilinx Platform studio [7], including AHB2AXI, AXI2APB, AXI2PLB. However, the OCP2AXI bridge is not supported.

There are many applications where protocol bridging plays an important role. In applications such as sound, graphic and network the peripheral component interconnect (PCI) has to be connected to the AMBA bus throw AHB-PCI bridge/adapter [8][9][10]. In the SoC test techniques the AHB-APB bridge is extensively adopted to adjust different protocol speeds [11][12][13].

It is true that commercial tools offer a form of bridging but additional complexity and the introduction of a new tool is not always desirable in modern design. However solutions both scientific and commercial do not offer a functional OCP to AXI Bridge.

3 Hardware Design of the On-chip Bridge

The OCP to AXI bridge connects two protocols with different functionality. The bridging involves mapping the inputs and outputs of the OCP slave to the inputs and outputs of AXI master. The signals coming from OCP master are inputs to OCP2AXI bridge which are then converted into AXI signals and delivered to AXI master.

Figure 1 shows the block scheme of OCP2AXI bridge, with key components highlighted. The bridge consists of:

- OCP slave
- OCP to AXI kernel
- AXI master
- AXI Downsizer

Because we can't directly connect AXI and OCP interfaces we developed our own intermediate interface which can be easily translated to both OCP and AXI

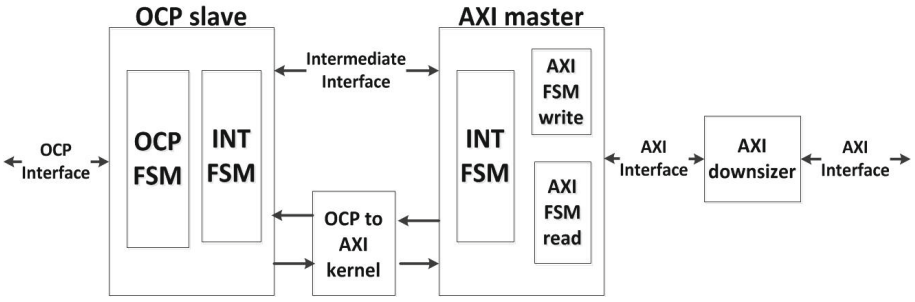

Fig. 1. Block scheme of OCP to AXI bridge

interfaces. The interface preserves all key features of both protocols but without specialized optimization of one particular protocol.

The intermediate interface consists of read and write channels with one common address channel (see Figure 2). Both read and data channels have similar interface with one FIFO's input for better handling of different protocol timing. Address channel covers all other protocol functionality, including out-of-order transaction, multiple transactions, and burst size. The interface is designed to provide a quick way to implement a lightweight interface between OCP and AXI interface. Finite state machine (intermediate FSM) is developed to implement functionality of intermediate interface and to serve as slaves for OCP/AXI protocol. The intermediate interface does not support specific optimizations such as separate read/write transaction for AXI protocol or configurable interface of OCP protocol.

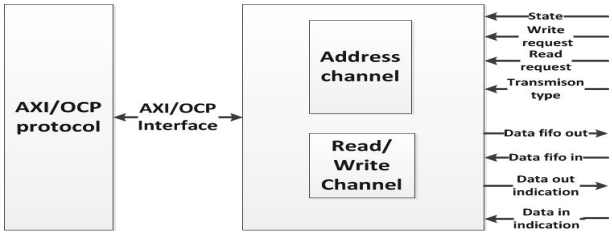

Fig. 2. Block diagram of intermediate interface

The OCP slave design applies a translation of the OCP interface signals to intermediate interface signals. The intermediate FSM is capable of executing translation but we still needed to generate handshake signals for OCP side of the bridge. Simple OCP FSM is implemented to control the flow of each OCP transaction and to generate signals as OCP slave interface (MCmd, MAddr, MData). The OCP FSM supports burst, pipelined, out-of-order transaction and its output is delivered to intermediate FSM.

The AXI master design is responsible for the reverse translation form the intermediate interface to the AXI interface. Again the intermediate FSM is implemented to execute translation and AXI FSM is created to handle hand shake between AXI interface and OCP2AXI bridge.

The AXI FSM is created with two subs FSMs, one FSM for reading and one FSM for writing. This preserves AXI ability to read and write independently of channels. Each new address initiates new AXI transaction and AXI FSM expect a response from the AXI interface to proceed with a next transaction.

OCP to AXI kernel is a time multiplexer between two intermediate FSM. Because there is usually differences in OCP and AXI timing it is necessary to synchronize two protocols. The kernel is monitoring the status signal of both intermediate interfaces and depending on the current state it gives permission for the next read/write transmission.

The AXI Downsizer is responsible for the downsizing data width of 256, 128, 64 to 32. This is a design from Xilinx [14] and it is offered as part of their ISE tool. Reason for implementing AXI Downsizer is necessity to simulate and emulate 32 data width within our verification/emulation environment while DSP design used as driver is designed with configurable OCP data width.

4 RTL Simulation of the Bridge

We used RTL simulation as the primary approach to debug RTL modifications until the whole design became stable enough. Once the design became stable to allow basic transactions we proceeded to more sophisticated levels of verification.

For sophisticated verification we developed SystemC verification environment. The environment contains a large number of high quality regression tests with high coverage over a large functional domain. Figure 3 shows the block design of the verification environment. The environment consists of four parts:

- OCP driver/monitor
- AXI random access memory (RAM)
- OCP2AXI Bridge
- System C environment

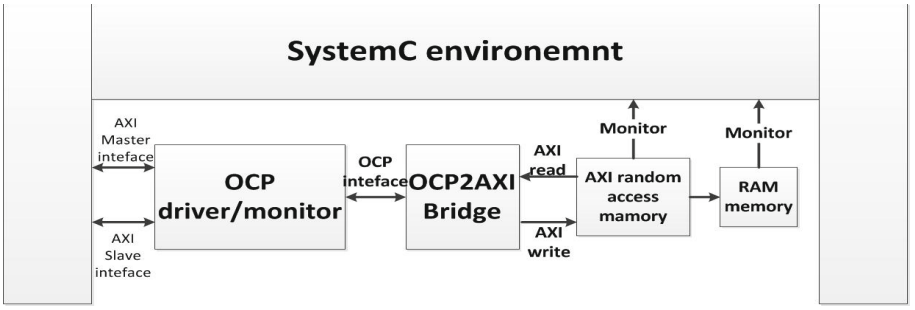

Fig. 3. Block design of the verification environment

The OCP driver/monitor is essentially an IMC DSP processor. The DSP was developed in our group during previous project and it has an OCP interface with adjustable bit data width (from 32 to 256 data bit width). With a limited interface optimization the DSP becomes precisely what we needed for this verification procedure. The DSP gives great controllability and visibility which can be easily combined with verification environment. It is relatively easy to create a wide range of input vectors and to cover almost all functionality of the OCP interface.

AXI random access memory is developed to support the AXI side of the bridge and it consists of two designs. The first design is the Xilinx AXI BRAM Controller which is a soft IP core designed as an AXI endpoint slave and the master device to local RAM [15]. The second design is a RAM with a regular interface, the RAM is a simple concurrent design with variable matrix as its basis.

The verification environment is a state of the art environment developed using the SystemC language. The SystemC language is an advanced set of C++ libraries that provides an event driven simulation [16]. The environment is successfully used in many different projects including previously mentioned DSP project. The environment has complete control over DSP which allows a wide variety of different OCP regression tests. These regression tests could be run over a couple of days in our ModelSim batch mode transaction pool.

During designing even after small changes it was necessary to rerun the whole regression pool to be sure that change did not affect any other bridge functionality. In case of new functionality it was necessary to develop a new test case which would cover new functionality. Until the whole design was not working properly we did not start the next step (synthesis). Even when everything was working correctly in simulation, we still continued to use simulation flows to ensure that the synthesis tool chain performs correctly.

5 FPGA Emulation of the OP2XI Bridge

RTL simulation is a basic methodology for verifying and debugging RTL design. The main problem of RTL simulation is its limited speed, one second of real time is usually around a couple of days in the simulator. To improve verification of real time situations we used the Versatile Express board as emulation environment.

5.1 Versatile Express Board

The Versatile Express platform provides a quality solution for rapid prototyping and hardware verification of the next generation of digital designs [17]. Versatile board is a highly configurable solution with high speed interfaces and the ability to develop and verify both software and hardware applications. Figure 4 gives a block description of the board components. The board consists of three parts:

- Versatile Express Motherboard
- Versatile Express CoreTile
- Versatile Express LogicTiles

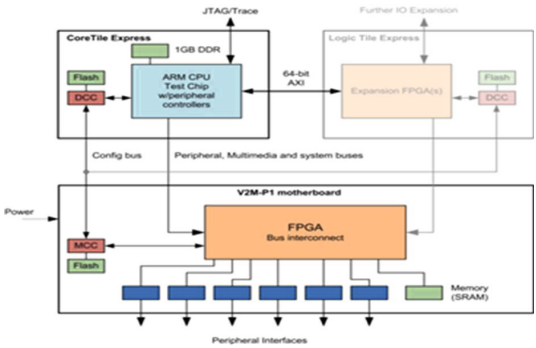

Fig. 4. Versatile Express board, block description

The motherboard has been specially designed to support future generations of software development. The board envelops all the necessary interfaces and peripherals for development of any new digital designs or graphic engine.

The CoreTile is delivered with different types of processor cores (Cortex-A15, Cortex-A9, Cortex-A7 and Cortex-A5) and it is mainly used as a CPU for the whole environment.

The LogiTile is delivered with a choice of different types of FPGAs and with capabilities to stack up to four boards one on top of each other. The board enables rapid prototyping, software/hardware codesign, verification and hardware driver development alongside a CoreTile CPU. This greatly reduces bring-up time and testing of IP design in parallel with software driver.

5.2 Synthesis Tool Flow

Most of our RTL design is coded in Verilog language so at the beginning we chose Xilinx Ise [18] as main synthesis tools. Xilinx Ise is relatively cheap and easy to use but does not support all features and languages available on the market. Unfortunately some part of our design is written in System Verilog which is not supported in XILINX Ise so it was necessary to find a different solution.

We had two solutions to either use Synplify Pro [19], or to use new Xilinx Vivado [20] suit. Both tools are high performance, cost effective FPGA synthesis tool with the ability to interpret System Verilog syntax.

The Vivado Design Suite implements all steps necessary for the FPGA bit generation (synthesis, mapping and placement) in one tool. Synplify Pro synthesis software is the industry standard FPGA tool and its unique Behavioral Technology performs optimization at a highest level. It is important to notice that in the case of Synplify Pro we still needed to use Xilinx Ise for mapping and placement.

In this case we have chosen a Synplify Pro not just for its optimization abilities but also because of good customer and time proven reviews. Compared to Synplify Pro Vivado Design Suit is a relatively a newcomer to the field and it is still necessary to pass the test of time.

Cshel scripts are created to control synthesis procedure. Synplify Pro interprets the RTL files and creates the edf file. Xilinx ISE collects the edf files and implements mapping and placement. In Table 1 you can see the synthesis results.

Table 1. Synthesis results

Device	Xlinx Virtex 6 xc6vlx760	%
Slices:	79.940 out of 948,480	8
LUTs:	82,027 out of 474,240	17
RAM36:	122 out of 720	16
RAM18:	17 out of 1440	1
IOB:	635 out of 1200	52
MAX. FREQ:	27.083 MHz	

5.3 Emulation on Versatile Express Board

The whole Versatile Express board is connected by an AXI interface [21] so the only way to send data between LogicTile board and CoreTile board is through the AXI bus.

LogicTile environment. The AXI bus was delivered as a basic driver for the LogicTile board. The problem was that the basic driver has only one AXI master for the emulation environment while in our case we needed at least two masters and one slave. For the generation of additional interfaces we used Xilinx Platform Studio [22] which enables easy generating of additional AXI interfaces. Figure 5 (a) presents a LogicTile environment which consists of AXI bus with connections to Logic Tile Fash DRAM memory, processor and emulation environment.

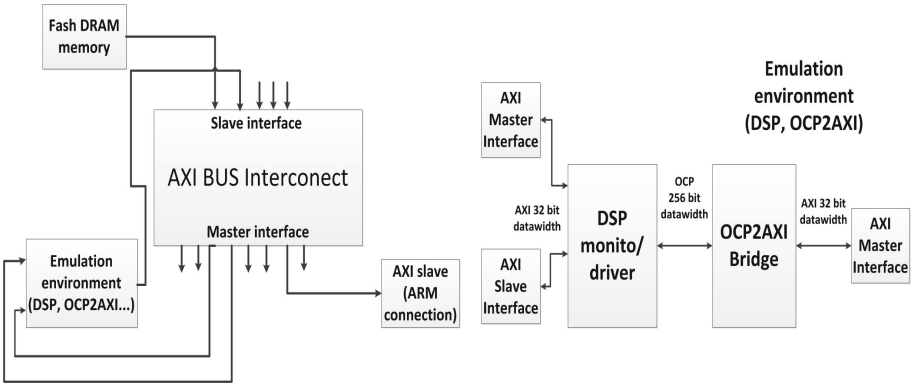

Fig. 5. Top system design (a) LogicTile environment (b) Emulation environment

Emulation environment (DSP, OCP2AXI...). During emulation development we tried to reuse as much of RTL simulation as possible knowing that we can greatly improve functional coverage and also simplify our development process. Figure 5 (b) shows the block design of the emulation environment. The environment consists of:

- DSP processor
- OCP2AXI bridge
- Emulation environment

Both DSP and OCP2XI bridge are described in previous chapters.

The emulation environment consists of two AXI master interfaces and one AXI slave interface. The first AXI master is connecting the OCP2AXI bridge with flash DRAM memory (see Figure 4) enabling collecting data from the bride output. The second master interface transfers data between DSP and flash DRAM memory delivering input date to the DSP. The AXI slave is giving the CPU full control over DSP design.

During emulation individual functionality or even entire blocks can be separately verified. This was enabled by DSP design which has nearly the same

functionality as in RTL simulation. The CPU is used to control the DSP from outside of FPGA using Linux drivers and partially ported SystemC verification environment.

Drivers for CPU core. The Core Tile board is delivered with Boot Monitor which is a standard application built on CPU platform library. The library handles the system initialization and provides basic I/O subsystem that supports simple drivers [23]. Unfortunately this was not enough to enable communication between CPU AXI interface and FPGA environment design on LogicTile boards.

In order to run the bridge core in the FPGA emulation board, we installed Linux image and developed two Linux drivers to enable delivering and collecting data from the design. Figure 6 presents Linux driver which consists of two components.

- Direct memory driver
- DSP driver

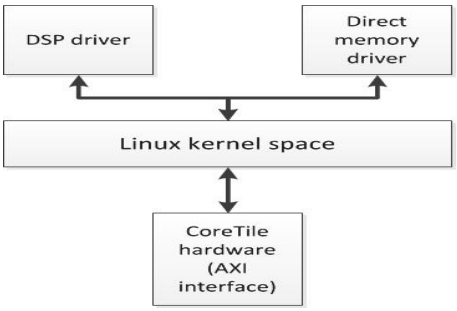

Fig. 6. CPU Linux driver design

The direct memory driver is responsible for delivering data to the flash DRAM memory located on the LogicTile board Figure 4. Its function is to provide necessary service to direct memory access so that data can be sent without any problems. This data will be used during a transaction as input/output data to the AXI interface of the bridge.

DSP driver is partly redesigned DSP control from SystemC environment. The driver controls the DSP AXI interfaces from CPU giving similar DSP controllability as in RTL simulation.

Beside these two drivers, the almost entire verification environment is ported and translated from the SystemC language to CSHELL scripts. For porting we extract SystemC functionality and optimized it for CSHELL scripts. It was not possible to use the whole environment because in simulation we had full visibility while in emulation only a small subset of signals was extracted by Xilinx ChipScope [24].

6 Results

For emulation and verification we used images of different sizes and definition. Besides images we also used many different types of data but images remain the

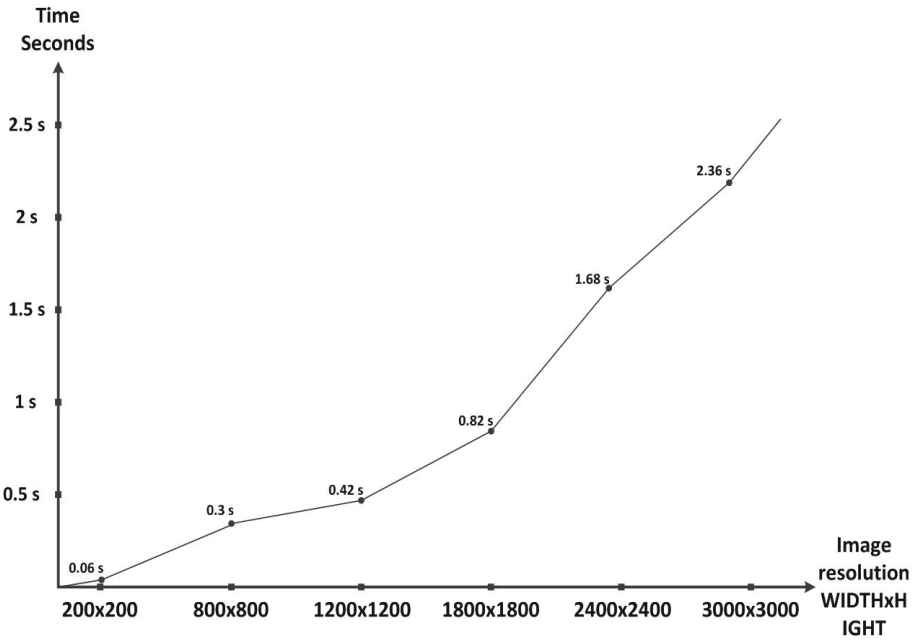

Fig. 7. Evaluation results

best and the simplest way to measure the quality of the bridge. Figure 7 presents an execution time for different image sizes on FPGA emulation board. The Y-presents size/resolution of images and the X-presents execution time.

We also tried to use the live stream for system emulation but unfortunately the max frequency is too low to support this feature (see table 1). The system can only process around 3 images per second, (this greatly depends on image quality) and for live stream you need at least 20 images per second.

The benefits of the RTL simulation environment such as high functional coverage, large number of tests and relatively long regression run are implemented in the emulation environment. Emulation still adds large speed increase so almost all simulation tests are rerun in less than a couple of hours. Table 2 presents a functional coverage results.

Table 2. Presents a functional coverage results

Design	Branch %	Conditions %	STMT %	Toggled %
OCP slave	91.176	94.737	96.00	96.585
OCP2AXI kernel	95.277	93.888	87.821	92.223
AXI master	90.887	96.551	90.221	98.002
AXI Downsizer	92.222	93.367	89.032	96.988

Our OCP2AXI bridge preserves all instructions and functionality of both protocols. Table 3 (a) presents protocol specifications which are supported in the bridge.

Table 3. Result tables: (a) Supported protocol specification (b) Synthesis report of the stand alone OCP2AXI Bridge

AXI interface	OCP interface	Device	Xlinx Virtex 6	%
Separate address and	Small set of	Slices:	79.940 of 948,480	1
data channel	mandatory signals	LUTs:	82,027 of 474,240	1
Unaligned data	Configurable address	RAM36:	122 of 720	0
transfers	and data width	RAM18:	17 of 1440	0
Limited burst transactions	Burst transfers	IOB:	635 of 1200	52
Separate read and	Inclusion of sideband	MAX. FREQ:	83.773 MHz	
write channel	signals			
Multiple addresses	Pipelined transfers			
Out-of-order transac.	Out-of-order transac.			

As explained in synthesis section, the max frequency of the system on FPGA is 27 MHZ (see table 1). This is a relatively low frequency, especially knowing that the bridge is not a large design. The reason for low frequency is coming from a complicated emulation environment. DSP is probably a synthesis overhead but it would be rather complicated to create dedicated OCP driver with the same emulation purpose and functional coverage. The synthesis results of the stand alone bridge are presented in Table 3 (b). The results are extracted from the bridge alone synthesis report and they represent the real capability of a bridge. From this report it is obvious that a stand alone bridge can handle real time applications like live streaming.

7 Conclusion

In this paper we presented our experience in designing, verifying and emulating an OCP to AXI bridge. The described design technique provides an easier implementation of the protocol bridge for modern digital design. Verification and emulation is done using a modern regression environment with large coverage and long test execution.

The proposed OCP to AXI bridge improves SOC integrations by allowing connection and reuse of IP's with different on-chip protocols. Furthermore it is also possible to connect two different bus protocols with almost no functional or bandwidth loss. Our FPGA emulation based on the Versatile Express board demonstrates that the bridge can be successfully used in many different real time situations such as live stream, high-definition images, and data transfer.

For future design we plan to optimize the emulation environment by designing a specific OCP driver and monitor so that we do not need to use IMC DSP as

a driver. With this improvement the maximal frequency should increase from 27 MHz to approximately 50 MHz which would enable sending a high definition live stream through an FPGA board.

References

1. AMBA AHB interface, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0203f/I1054045.html
2. Open Core Protocol International Partnership (OCP-IP), http://www.ocpip.org/
3. AMBA Open Specifications, http://www.arm.com/products/system-ip/amba/amba-open-specifications.php
4. Sonics Express, http://sonicsinc.com/products/
5. Arteris FlexNoC Interconnect IP, http://www.arteris.com/
6. MIPS-Imagination BusBridge3 Module, http://www.imgtec.com/mips/mips-busbridge3.asp
7. Xilinx Platform studio, http://www.xilinx.com/ise/embedded/edk_ip.htm
8. Zhonghai, W., Yizheng, Y., Jinxiang, W., Mingyan, Y.: Designing AHB/PCI bridge ASIC. In: Proceeding of the 4th International Conference on ASIC, pp. 578–580 (2001)
9. AMBA-AHB PCI Bridge IP Introductory Document PLDA Ltd., http://www.plda.com/
10. ARM PrimeCell External Bus Interface (PL220), ARM, ARM DDI 0249B (2002), http://www.arm.com
11. Song, J., Yi, H., Han, J., Park, S.: An efficient SoC test technique by reusing on/off-chip bus bridge. Journal IEEE Transactions on Circuits and Systems 56(3), 554–565 (2009)
12. Lin, C., Liang, H.: Bus-oriented DFT design for embedded cores. In: Proc. IEEE Asia-Pacific Conf. on Circuits and Systems, pp. 561–563 (2004)
13. Song, J., Min, P., Yi, H., Park, S.: Design of Test Access Mechanism for AMBA-Based System-on-a-Chip. In: IEEE VLSI Test Symmposium, pp. 375–380 (2007)
14. Xilinx data-width conversion Downsizer, http://www.xilinx.com/products/intellectual-property/axi_interconnect.htm
15. AXI BRAM Controller, http://www.xilinx.com/products/intellectual-property/axi_bram_if_ctlr.htm
16. SystemC, http://www.accellera.org/activities/committees/systemc-language/
17. Versatile Express Product Family, http://www.arm.com/products/tools/development-boards/versatile-express/index.php
18. ISE WebPACK Design Software, http://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.htm
19. Synplify Pro, http://www.synopsys.com/Tools/Implementation/FPGAImplementation/FPGASynthesis/Pages/SynplifyPro.aspx
20. Vivado Design Suite, http://www.xilinx.com/products/design-tools/vivado/
21. Example LogicTile Express 13MG design for a CoreTile Express A15x2, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dai0305a/index.html
22. Xilinx Platform Studio, http://www.xilinx.com/tools/xps.htm
23. Versatile Express Boot Monitor, http://infocenter.arm.com/help/topic/com.arm.doc.dui0465f/DUI0465F_boot_monitor_trm.pdf
24. Xilinx ChipScope, http://www.xilinx.com/tools/cspro.htm