International Journal of Electronics Communication and Computer Engineering
Volume 3, Issue 6, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

An Efficient System On-Chip Bus with OCP Interface

Jessi Mallika D.
M.Tech. Student, CVSR College of Engineering
jessimallika@gmail.com

Mrs. A. Krishna Kumari
Professor, CVSR College of Engineering
akk_gnec@rediffmail.com

***Abstract* — The implementation of a huge scale SoC (System-on-chip) is becoming difficult task not only due to its complexity, but also the design of a more amount of IPs. A consensus interface protocol for IP cores is becoming significant and even predictable for a successful SoC establishment. Open Core Protocol (OCP), with its candidness, concerted, not-for-profit nature, and inherent large industry member base, is quickly becoming a feasible and preferable solution over a close or an in-house standard. In this paper a well-defined interface standard, the Open Core Protocol (OCP), has adopted to design the internal bus architecture. An efficient bus architecture to support most advanced bus functionalities defined in OCP has been developed. These functionalities include burst transactions, lock transactions, pipelined transactions, and out-of-order transactions. First model and design the on-chip bus with transaction level modeling for the consideration of design flexibility and fast simulation speed. Then implement the RTL (Register Transfer Level) models of the bus for synthesis and gate-level simulation. Experimental results show that the proposed TLM (Transaction Level Modeling) model is quite efficient for the whole system simulation and the real implementation can significantly save the communication time.**

***Keywords* — AMBA, AXI, FSM, OCP, SoC.**

I. INTRODUCTION

As more and more IP cores are integrated into an SOC design, the communication flow between IP cores has increased drastically and the efficiency of the on-chip bus has become a dominant factor for the performance of a system.

An SOC chip usually contains a large number of IP cores that communicate with each other through on-chip buses. As the VLSI process technology continuously advances, the frequency and the amount of the data communication between cores increase substantially. As a result, the ability of on-chip buses to deal with the large amount of data traffic becomes a dominant factor for the overall performance. The design of on-chip buses can be divided into two parts: **bus interface** and **bus architecture**. The bus interface involves a set of interface signals and their corresponding timing relationship, while the bus architecture refers to the internal components of buses and the interconnections among the IP cores. The widely accepted on-chip bus, AMBA AHB [1], defines a set of bus interface to facilitate basic (single)and burst read/write transactions. AHB also defines the internal bus architecture, which is mainly a shared bus composed of multiplexors. The multiplexer-based bus architecture works well for a design with a small number of IP cores. When the number of integrated IP cores increases, the communication between IP cores also increase and it becomes quite frequent that two or more master IPs would request data from different slaves at the same time. The shared bus architecture often cannot provide efficient communication since only one bus transaction can be supported at a time.

To solve this problem, two bus protocols have been proposed recently. One is the Advanced extensible Interface protocol (AXI)[1], [9]proposed by the ARM company. AXI defines five independent channels (write address, write data, write response, read address, and read data channels). Each channel involves a set of signals. AXI does not restrict the internal bus architecture and leaves it to designers. Thus designers are allowed to integrate two IP cores with AXI by either connecting the wires directly or invoking an in-house bus between them. The other bus interface protocol is proposed by a non-profitable organization, the Open Core Protocol – International Partnership (OCP-IP) [2]. OCP is an interface (or socket) aiming to standardize and thus simplify the system integration problems. It facilitates system integration by defining a set of concrete interface (I/O signals and the handshaking protocol) which is independent of the bus architecture. Based on this interface IP core designers can concentrate on designing the internal functionality of IP cores, bus designers can emphasize on the internal bus architecture, and system integrators can focus on the system issues such as the requirement of the bandwidth and the whole system architecture. In this way, system integration becomes much more efficient. Most of the bus functionalities defined in AXI and OCP are quite similar. The most conspicuous difference between them is that AXI divides the address channel into independent write address channel and read address channel such that read and write transactions can be processed simultaneously. However, the additional area of the separated address channels is the penalty. Some previous work has investigated on-chip buses from various aspects. The work presented in [3] and [4] develops high-level AMBA bus models with fast simulation speed and high timing accuracy. The authors in [5] propose an automatic approach to generate high-level bus models from a formal channel model of OCP. In both of the above work, the authors concentrate on time and accuracy of simulation models at high-level but did not provide real hardware implementation details.

The authors implement the AXI interface on shared bus architecture. Even though it costs less in area, the benefit of AXI in the communication efficiency may be limited by the shared-bus architecture. In this paper we propose a high-performance on-chip bus design with OCP as the bus interface. We choose OCP because it is open to the public and OCP-IP has provided some free tools to verify this protocol. Our proposed bus architecture features crossbar/partial -crossbar based interconnect and realizes most transactions defined in OCP, including 1) single

transactions,2) burst transactions, 3) lock transactions, 4) pipelined transactions, and 5) out-of-order transactions [7]. In addition, the proposed bus is flexible such that one can adjust the bus architecture according to the system requirement.

The remainder of this paper is organized as follows. In section 2The various advanced functionalities of on-chip buses are described. Section 3 describes functioning of the OCP Block Diagram. Section 4 gives the Implementation of OCP master/slave FSM. Section 5 is experimental results which show the efficiency on both simulation speed and data communication. Conclusions are then drawn in Section 6.

II. On-chip Bus Functionalities

The various bus functionalities includes
A. Burst
B. Lock
C. Pipelined and
D. Out-of-Order transactions.

- *Burst Transactions*

The burst transactions allow the grouping of multiple transactions that have a certain address relationship, and can be classified into multi-request burst and single-request burst according to how many times the addresses are issued. FIGURE 1 shows the two types of burst read transactions. The multi-request burst as defined in AHB is illustrated in FIGURE 1(a) where the address information must be issued for each command of a burst transaction (e.g., A11, A12, A13 and A14). This may cause some unnecessary overhead. In the more advanced bus architecture, the single-request burst transaction is supported. As shown in FIGURE 1(b), which is the burst type defined in AXI, the address information is issued only once for each burst transaction. In our proposed bus design we support both burst transactions such that IP cores with various burst types can use the proposed on-chip bus without changing their original burst behavior.

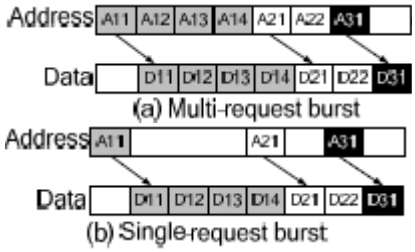

Fig.1. Burst transactions

- *Lock Transactions*

Lock is a protection mechanism for masters that have low bus priorities. Without this mechanism the read/write transactions of masters with lower priority would be interrupted whenever a higher-priority master issues a request. Lock transactions prevent an arbiter from performing arbitration and assure that the low priority masters can complete its granted transaction without being interrupted.

- *Pipelined Transactions*

Figure 2(a) and 2(b) show the difference between non-pipelined and pipelined (also called outstanding in AXI) read transactions. In FIGURE 2(a), for a non-pipelined transaction a read data must be returned after its corresponding address is issued plus a period of latency. For example, D21 is sent right after A21 is issued plus *t*. For a pipelined transaction as shown in FIGURE 2(b), this hard link is not required. Thus A21 can be issued right after A11 is issued without waiting for the return of data requested by A11 (i.e., D11-D14).

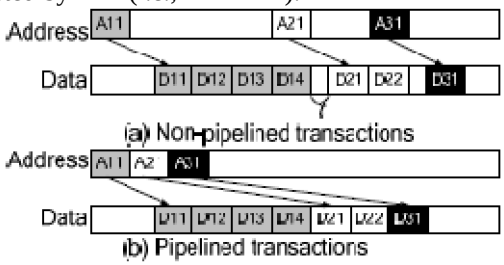

Fig.2. Pipeline transactions

- *Out-of-order Transactions*

The out-of-order transactions allow the return order of responses to be different from the order of their requests. These transactions can significantly improve the communication efficiency of an SOC system containing IP cores with various access latencies as illustrated in FIGURE 3.In FIGURE 3(a) which does not allow out-of-order transactions, the corresponding responses of A21 and A31must be returned after the response of A11. With the support of out-of-order transactions as shown in FIGURE 3(b), the response with shorter access latency (D21, D22 and D31) can be returned before those with longer latency (D11-D14) and thus the transactions can be completed in much less cycles.

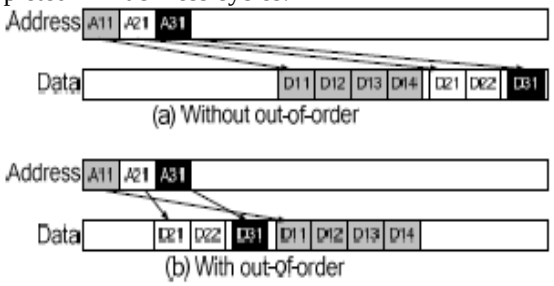

Fig.3. Out-of-order transactions

III. Functional Blocks of the OCP Bus

The architecture of the proposed on-chip bus is illustrated in FIGURE 4, where we show an example with four masters and four slaves. A crossbar architecture is employed such that more than one master can communicate with more than one slave simultaneously. If not all masters require the accessing paths to all slaves, partial crossbar architecture is also allowed. The main blocks of the proposed bus architecture are described next. The main blocks of the diagram are Arbiter, Multiplexer, Decoder, Master and slave.

International Journal of Electronics Communication and Computer Engineering
Volume 3, Issue 6, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

A. Arbiter

In traditional shared bus architecture, resource contention happens whenever more than one master requests the bus at the same time. For a crossbar or partial crossbar architecture, resource contention occurs when more than one master is to access the same slave simultaneously. In the proposed design each slave IP is associated with an arbiter that determines which master can access the slave and then it sends grant signal to the requested master.

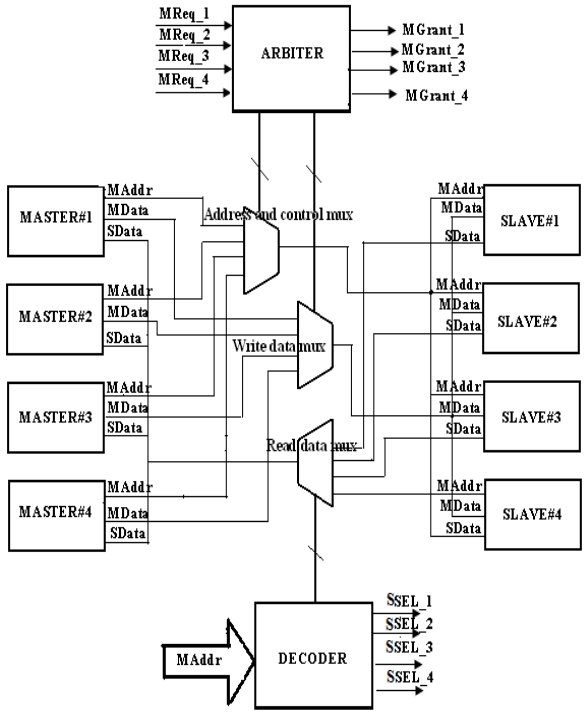

Fig.4. OCP Block Diagram

B. Decoder

Since more than one slave exists in the system, the decoder decodes the address and decides which slave return response to the target master. In addition, the proposed decoder also checks whether the transaction address is illegal or nonexistent and responses with an error message if necessary.

C. Multiplexer

A multiplexer is used to solve the problem of resource contention when more than one slave returns the responses to the same master. It selects the corresponding slave/master according to the given selection lines. There are multiplexers one is address and control mux and the two are for data read and write.

D. Master/Slave

Master and Slave are the entities consisting of different Address and Data Locations. Only the master can send the commands and is the controlling entity. The Slave responds to commands presented to it, either by accepting data from the master, or representing data to the master.

E. FSM-M & FSM-S

FSM-M acts as a master Finite State Machine and generates the OCP signals of a master, while FSM-S acts as a slave Finite State Machine and generates those of a slave. Depending on whether a transaction is a read or a write operation, the request and response processes are different. For a write transaction, the data to be written is sent out together with the address of the target slave, and the transaction is complete when the target slave accepts the data and acknowledges the reception of the data. For a read operation, the address of the target slave is first sent out and the target slave will issue an accept signal when it receives the message. The slave then generates the required data and sends it to the bus where the data will be properly directed to the master requesting the data. The read transaction finally completes when the master accepts the response and issues an acknowledge signal. In the proposed bus architecture, we employ two types of finite state machines, namely FSM-M and FSM-S to control the flow of each transaction. FSM-M acts as a master and generates the OCP signals of a master, while FSM-S acts as a slave and generates those of a slave. These finite state machines are designed in a way that burst, pipelined, and out-or-order read/write transactions can all be properly controlled.

IV. IMPLEMENTATION OF OCP

The important aspect of OCP is not only which components or blocks it houses but also interfacing between them. Here are some main signals that plays main role for the signal transactions.

The request issued by system is given to slave by *MCmd* signal. Similarly, in Write operation, the input address and data provided by the system will be given to slave through the signal *MAddr* and *MData* and when those information's are accepted, slave will give *SCmdAccept* signal to the master which ensures that the system can issue next request. During Read operation, system issues the request and address to slave which will set *SResp* and fetch the corresponding data that is given to output through *Sdata*.

Maddr: The Transfer address, *MAddr* specifies the slave-dependent address of the resource targeted by the current transfer. To configure this field into the OCP, use the addr parameter. To configure the width of this field, use the addr_wdth parameter.

MCmd: Transfer command. This signal indicates the type of OCP transfer the master is requesting. Each non-idle command is either a read or write type request, depending on the direction of data flow.

Mdata: Write data. This field carries the write data from the master to the slave. The field is configured into the OCP using the **mdata** parameter and its width is configured using the data_wdth parameter. The width is not restricted to multiples of 8.

ScmdAccept: Slave accepts transfer. A value of 1 on the *SCmdAccept* signal indicates that the slave accepts the master's transfer request. To configure this field into the OCP, use the cmdaccept parameter.

Sdata: This field carries the requested read data from the slave to the master. The field is configured into the OCP using the sdata parameter and its width is configured using the data_wdth parameter. The width is not restricted to multiples of 8.

International Journal of Electronics Communication and Computer Engineering
Volume 3, Issue 6, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

Sresp: Response field is given from the slave to a transfer request from the master. The field is configured into the OCP using the resp parameter.

Size: This is for the length of the sequence for burst signal and out-of-order. Length indicates the number of transfers. For precise bursts, the value indicates the total number of transfers, and is constant throughout the burst. The burst length that can be configured represents that many read or write operation can be performed in sequence. In out-of-order the transfer will be in non sequence form.

Control: Control is the main command that which decides simple read/write or burst or out-of-order. In our OCP there are 6 controls simple read,simple write, burst read, burst write, out-of-order read, out-of-order read.

FSM for OCP master: The Finite State Machine (FSM) is developed for the simple write and read operation of OCP Master. The simple write and read operation indicates that the control goes to IDLE state after every operation. Basically, the operation in the OCP will be held in two phases.

- Request Phase
- Response Phase

Initially the control will be in IDLE state (Control = "000") at which all the outputs such as MCmd, MAddr and MData are set to "don't care". The system will issue the request to the master such write request which leads to the WRITE state (Control = "001"). In this state, the address and the data will be given to the slave that is to be written and hence the process will get over only when the SCmdAccept is asserted to high. If SCmdAccept is not set, this represents that the write operation still in process and the control will be in the WRITE state itself. Once the write operation is over the control will go to the IDLE state and then it will check for the next request.

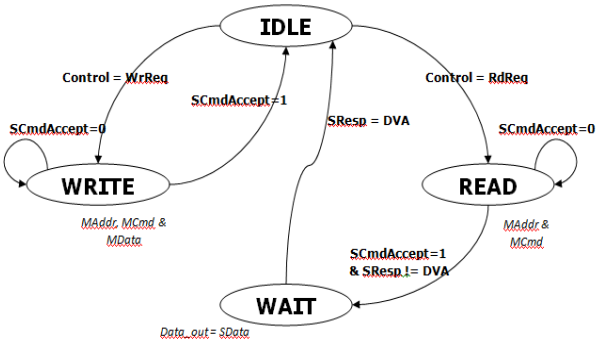

Fig.5. FSM for OCP master

When the read request is made, the control will go to the READ state (Control = "010") and the address is send to the slave which in turn gives the SCmdAccept signal that ends the request phase. Once the SCmdAccept is set and SResp is not Data Valid (DVA), the control will go the WAIT state and will be waiting for the SResp signal. When the read operation is over which represents that the SResp is set to DVA and the data for the corresponding address is taken. Hence the SResp signal ends the response phase and the control will go the IDLE state, then checks for the next request.

The design of the Open Core Protocol starts with the initial study based on which the development of FSM (Finite State Machine) for the various supporting operation after which the development of VHDL for the FSM.

FSM for OCP slave: The FSM for the OCP Slave which has the simple write and read operation is developed and is shown in the Figure 3.9 The slave will be set to the respective state based on the MCmd issued by the master and the output of this slave is that the SCmdAccept and SResp. Initially control will be in the IDLE state and when the master issues the command as

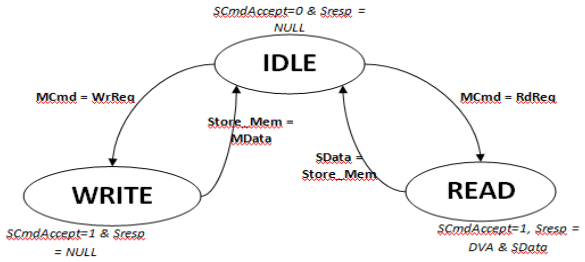

Fig.6. FSM for OCP slave

write request, and then the control will go the WRITE state in which the data will be written to the corresponding memory address location which is sent by the masters. Once the write operation is finished, the SCmdAccept signal is set to high and is given to the master.When MCmd is given as read request, then the control will move to the READ state in which the data will read from the particular memory address location that is given by the master. Hence the SCmdAccept is set to high and the SResp is set to the DVA which represents that the read operation over and control goes to the IDLE state.

Depending on whether a transaction is a read or a write operation, the request and response processes are different. So for burst, burst count is specified that is burstsize both for read and write. The main advantage of this burst extension is that the address will be generated with respect to the burst length mentioned. This enables the automatic generation of the address and acts as a major advantage of OCP.

V. RESULTS

The proposed design is coded in VHDL language and simulated using Modelsim 6.4b. The simulation result of the integrated design gives the clear view on the OCP Burst operation which is shown in Figure 7.

Here the burst length is given as 8 and hence the address will be generated for number of memory locations and the corresponding input data is given along with the size. The sequence address generation and writing the input data to the corresponding memory location is represented in the waveform clearly. Also the increment of count value is shown in the waveform based on which the sequence of address get generated and the corresponding data that stored in the memory are read out. The count is implemented which will be incremented with respect to the burst length. The master and slave will go to the IDLE state when the burst operation got over which can be indicated by the count i.e. when the count reaches the

International Journal of Electronics Communication and Computer Engineering
Volume 3, Issue 6, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

burst length given, it got reset and hence the address will be generated from the initial value.

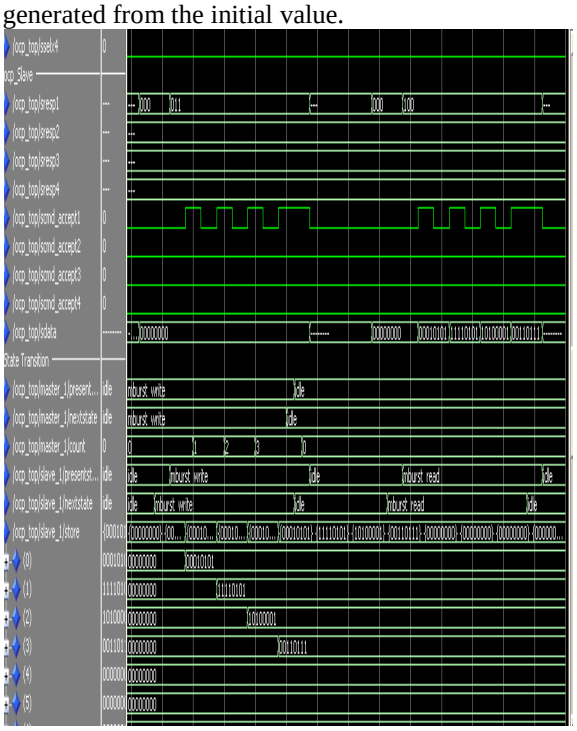

Fig.7. Burst transaction

Fig.8. shows the result of out-of-order in which the generated address and data will be in non sequence i.e, generating the alternate addresses and data. Here also the count is implemented which will be incremented with respect to the length. The master and slave will go to the IDLE state when the burst operation got over which can be indicated by the count.

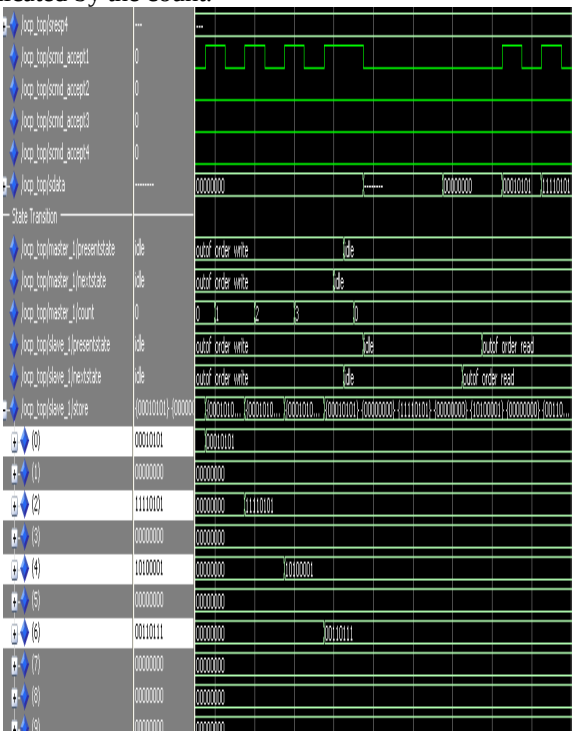

Fig.8. Out-of-Order transaction

By using Xilinx the proposed design is synthesized and RTL (register transfer logic) models are obtained shown in the Figure 9. The Top level RTL schematic shows the signals that are used in the design.

VI. Conclusion

In this work, initially the investigation on the OCP is carried out and the basic commands and its working are identified based on which the signal flow diagram and the specifications are developed for designing the OCP using VHDL. Cores with OCP interfaces and OCP interconnect systems enable true modular, plug-and-play integration; allowing the system integrators to choose cores optimally and the best application interconnect system. This allows

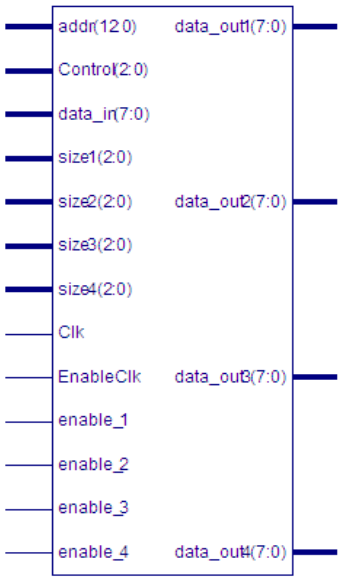

Fig.9. Top level RTL Schematic

the designer of the cores and the system to work in parallel and shorten design times. In addition, not having system logic in the cores allows the cores to be reused with no additional time for the core to be re-created. Depending upon the real time application these intellectual properties can be used. The basic aim of our project is to model the master and slave of OCP and we have successfully modeled both MASTER and SLAVE along with internal memory design using VHDL. All of the commands and data are successfully transferred from one IP core to the other IP core using OCP. There is no loss of data or control information. The Various Scenarios for each component in the OCP design are verified effectively during the simulation wit respect to its behavior. And the proposed design is verified and synthesized using Xilinx tool.

References

[1] Advanced Microcontroller Bus Architecture (AMBA) Specification Rev 2.0 & 3.0, ttp://www.arm.com.

[2] Open Core Protocol (OCP) Specification, http://www.ocpip.org/home

[3] Y.-T. Kim, T. Kim, Y. Kim, C. Shin, E.-Y. Chung, K.-M. Choi, J.-T. Kong, S.-K. Eo, “Fast and Accurate Transaction Level Modeling of an Extended AMBA2.0 Bus Architecture,” *Design, Automation, and Test in Europe*, pages 138-139, 2005.

[4] G. Schirner and R. Domer, "Quantitative Analysis of Transaction Level Models for the AMBA Bus," *Design, Automation, and Test in Europe*, 6 pages, 2006.

[5] C.-K. Lo and R.-S. Tsay, "Automatic Generation of Cycle Accurate and Cycle Count Accurate Transaction Level Bus

[6] Models from a Formal Model," *Asia and South Pacific Design Automation Conference*, pages 558-563, 2009.

[7] IBM corporation, "prioritization of out-of-order data transfers on shared data bus," us patent no. 7,392,353, 2008.

[8] Partha Pratim Pande, Cristian Grecu, Michael Jones, Andr´e vanov, and Resve Saleh, "Performance evaluation and design trade-offs for network-on-chip interconnect architectures", IEEE Trans. Computers, vol. 54, no. 8, pp. 1025–1040, Aug. 2005.

[9] N.Y.-C. Chang, Y.-Z. Liao and T.-S. Chang, "Analysis of Shared-link AXI," *IET Computers & Digital Techniques*, Volume 3, Issue 4, pages 373-383, 2009

[10] James Aldis, "Use of OCP in OMAP 2420", http://www.ocpip.org/, 2005.

[11] Partha Pratim Pande, Cristian Grecu, Michael Jones, Andr´e vanov, and Resve Saleh, "Performance evaluation and design trade-offs for network-on-chip interconnect architectures", IEEE Trans. Computers, vol. 54, no. 8, pp. 1025–1040, Aug. 2005.

[12] Natale Barsotti, Riccardo Mariani, Matteo Martinelli, and Mario Pasquariello, "Dynamic verification of OCP-based SoC", in Proc. IEEE Int'l Conf. on System-on-Chip, Tampere, Finland, Nov. 2005, p. 22.