IMPLEMENTATION OF OCP FOR THE ON-CHIP BUS

Mr. K. Santhosh[1], Mr. V. Rama Krishna[2], Dr. Syed Musthak Ahmed[3]
PG Student, Dept. of Electronics and Communication Engineering, SR Engineering College, India[1]
Assistant Professor, Dept. of Electronics and Communication Engineering, SR Engineering College, India[2]
Professor & H.O.D, Dept. of Electronics and Communication Engineering, SR Engineering College, India[3]

***Abstract*—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. The on-chip bus design can be divided into two parts, namely the interface and the internal architecture of the bus. 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 models of the bus for synthesis and gate-level simulation. Experimental results show that the proposed TLM model is quite efficient for the whole system simulation and the real implementation can significantly save the communication time.**

Keywords — Single transactions, Burst transactions, Lock transactions, Pipelined transactions, and out-of-order transactions.

I. INTRODUCTION

The On-Chip bus plays a key role in the system-on-a-chip (SoC) design by enabling the efficient integration of heterogeneous system components such as CPUs, DSPs, application- specific cores, memories, and custom logic.

Recently, as the level of design complexity has become higher, SoC designs require a system bus with high bandwidth to perform multiple operations in parallel. To solve the bandwidth problems, An efficient OCP protocol has been developed.

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 IP cores increase substantially. As a result, the ability of onchip 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, 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.

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). 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.

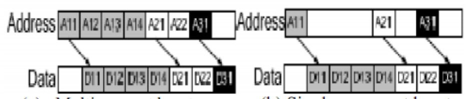
In this paper a high-performance on-chip bus design with OCP as the bus interface has been proposed. OCP has chosen, because it is open to the public and OCP-IP has provided some free tools to verify this protocol. The proposed bus architecture features crossbar/partial-crossbar based interconnect and 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. 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. The various advanced functionalities of on-chip buses are

- Pipelined, and
- Out-of-order transactions

A. *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. Fig.1 shows the two types of burst read transactions. The multi-request burst as defined in AHB is illustrated in Fig.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 Fig.1(b), which is the burst type defined in AXI, the address information is issued only once for each burst transaction. In the proposed bus design both burst transactions are supported such that IP cores with various burst types can use the proposed on-chip bus without changing their original burst behaviour.



(a) Multi-request burst (b) Single – request burst

Fig.1 Burst transactions

B. *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.

C. *Pipelined transactions (outstanding transactions)*

Fig. 2(a) and 2(b) show the difference between non-pipelined and pipelined (also called outstanding in AXI) read transactions. In Fig. 2(a), for a non-pipelined transaction a read data must be returned after its described in Section 2. Section 3 details the hardware architecture of the proposed bus. Section 4 gives the experimental results which show the efficiency on both simulation speed and data communication. Conclusions are then drawn in Section 5. 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 Fig. 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).

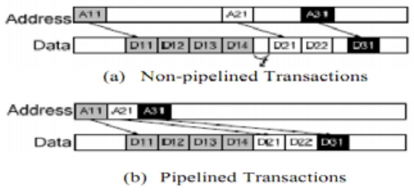


Fig. 2 Pipelined transactions.

D. *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 Fig. 3. In Fig. 3(a) which does not allow out-of-order transactions, the corresponding responses of A21 and A31 must be returned after the response of A11. With the support of out of-order transactions as shown in Fig. 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.

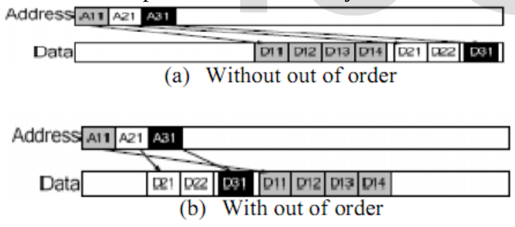


Fig.3. Out-of-order transactions

III. ON-CHIP BUS DESIGN

The architecture of the proposed on-chip bus is illustrated in Fig. 4, where an example with two masters and two slaves is shown. 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.

Basically OCP has the address is of 13bits, data is of 8bits, control signal is of 3bits and burst is of integer type.

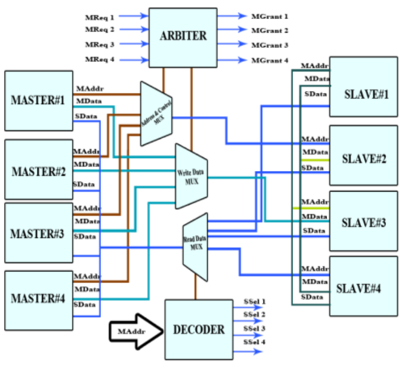
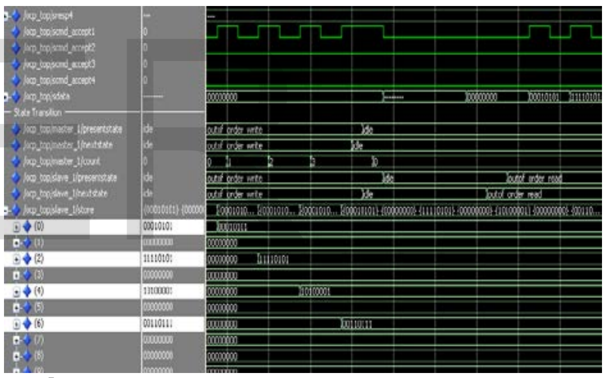


Fig.4. OCP BLOCK DIAGRAM

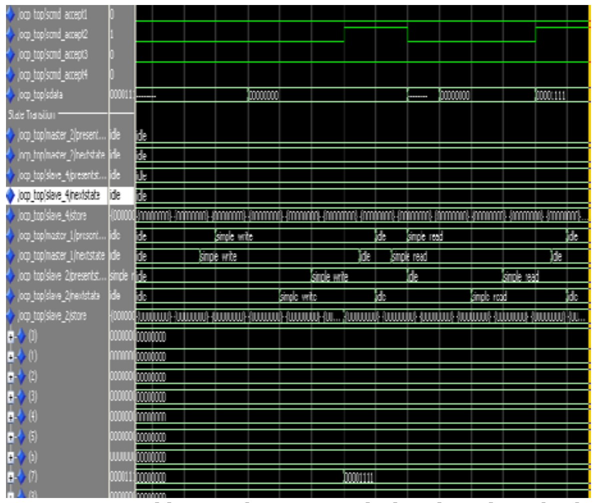
A. *Arbiter*

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.



B. *Decoder*

Since more than one slave exists in the system, the decoder decodes the address and decides which slave return response to the


/ocp_top/scmd_accept1
/ocp_top/scmd_accept2
/ocp_top/scmd_accept3
/ocp_top/scmd_accept4
/ocp_top/sdata
State Transition
/ocp_top/master_2/presentstate
/ocp_top/master_2/nextstate
/ocp_top/slave_4/presentstate
/ocp_top/slave_4/nextstate
/ocp_top/slave_4/store
/ocp_top/master_1/presentstate
/ocp_top/master_1/nextstate
/ocp_top/slave_2/presentstate
/ocp_top/slave_2/nextstate
/ocp_top/slave_2/store
simple write
simple read
idle

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 response from the slave that has the highest priority.

IV. RESULTS

The proposed design is coded in VHDL language and the 8kbit memory (2^{13}= 8192bits = 8kbits) is used in the slave side in order to verify the protocol functionality. The System will give the inputs to OCP Master during Write operation and receive signals from OCP Slave during Read operation. The main blocks of the proposed bus architecture are described below.

Fig.6 Burst transaction

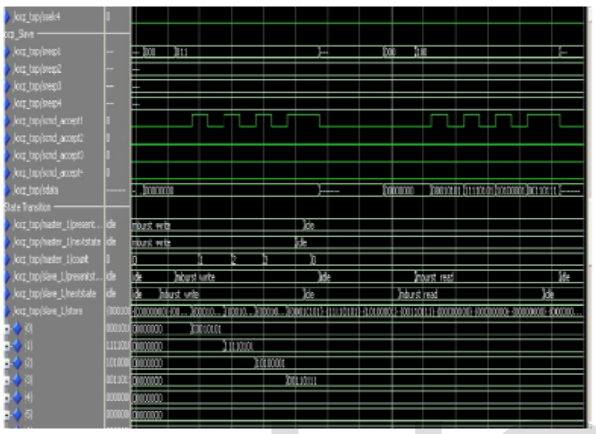


Fig.7 Out of order Transaction

Simulated using Xilinx ISE tool. The simulated waveforms for simple transactions, burst transactions, pipelined transactions and out of order transactions are shown in following figures..

V. CONCLUSION

This project work presents the OCP (Open Core Protocol) design which acts as an interface between two different IP cores. 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, plugand-play integration. The simulation result shows that the communication between different IP cores using OCP is proper. Based on the result obtained, the burst extension is seen to automate the address generation. The initial address alone is provided to the protocol.

REFERENCES

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

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

[3] 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] Kim Y.-T., T. Kim, Y. Kim, C. Shin, E.-Y. Chung, K.-M. Lo C.-K. and R.-S. Tsay, "Automatic Generation of Cycle Accurate and Cycle Count Accurate Transaction Level Bus Models from a Formal Model," *Asia and South Pacific Design Automation Conference*, pages 558-563, 2009.

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