Design and Analysis of Open Core Protocol for the on Chip Bus Using VHDL

Naseer Rasool .T[1] , Shaik Jaffar[2], Shaik Saheb Basha[3]

[1]M.Tech Student, Department of ECE, Madina Engineering College, Kamalapuram, Andhra Pradesh, India
[2]Associate Professor, Department of ECE, Madina Engineering College, Kamalapuram, Andhra Pradesh, India
[3]Principal , Department of ECE, Madina Engineering College, Kamalapuram, Andhra Pradesh, India

Abstract: ***In this paper, efficient bus architecture to support most advanced bus functionalities defined in OCP, including burst transactions, lock transactions, pipelined transactions, and out-of-order transactions with respect to its suitable application in the real time product. The Open Core Protocol (OCP) was designed and the hardware modeling for that architecture was done using VHDL. This design is Simulated and Synthesized. As 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 work the well-defined interface standard Open Core Protocol (OCP) is adopted, and the internal bus architecture is designed. The Open Core Protocol (OCP) is a core centric protocol which defines a high-performance, bus-independent interface between IP cores that reduces design time, design risk, and manufacturing costs for SOC designs. Main property of OCP is that it can be configured with respect to the application required. The OCP is chosen because of its advanced supporting features such as configurable sideband control signaling and test harness signals, when compared to other core protocols. The OCP defines a point-to-point interface between two communicating entities such as IP cores and bus interface modules. One entity acts as the master of the OCP instance, and the other as the slave. Only the master can present commands and is the controlling entity. The slave responds to commands presented to it, either by accepting data from the master, or presenting data to the master. For two entities to communicate there need to be two instances of the OCP connecting them such as one where the first entity is a master and one where the first entity is a slave.***

Keywords: OCP, IP, SOC, AXI, AHB

1. Introduction

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 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 [2] defines a set of bus interface to facilitate basic (single) and burst read/write transactions. AHB [3] & [4] 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] 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) [3] .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 [2] 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.

2. Over View of OCP Functionalities

We first describe the various bus functionalities including;

- Burst
- Lock
- Pipelined
- Out-Of-Order Transactions

2.1 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.

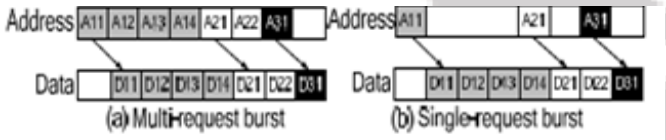

Figure 1: Burst transactions

2.2 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.

2.3 Pipelined transactions (outstanding 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).

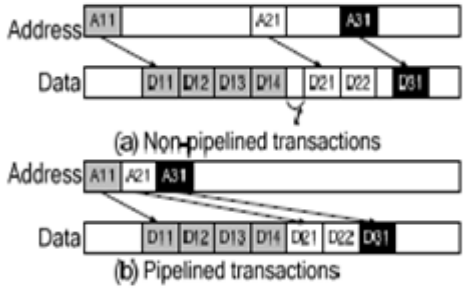

Figure 2: Pipelined transactions

2.4 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 A31 must 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.

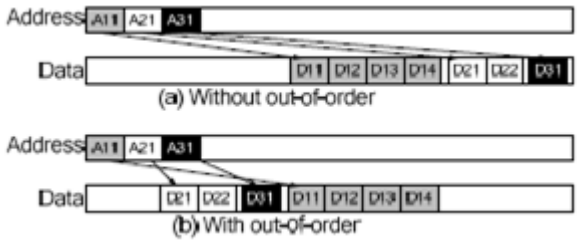

Figure 3: out-of-order transctions

3. Proposed System Block Diagram

The block diagram which explains the basic operation and characteristics of OCP is shown in Figure 5.The OCP defines a point-to-point interface between two communicating entities such as IP cores and bus interface modules. One entity acts as the master of the OCP instance, and the other as the slave. Only the master can present commands and is the controlling entity. The slave responds to commands presented to it, either by accepting data from the master, or presenting data to the master. For two entities to communicate there need to be two instances of the OCP connecting them such as one where the first entity is a master, and one where the first entity is a slave.

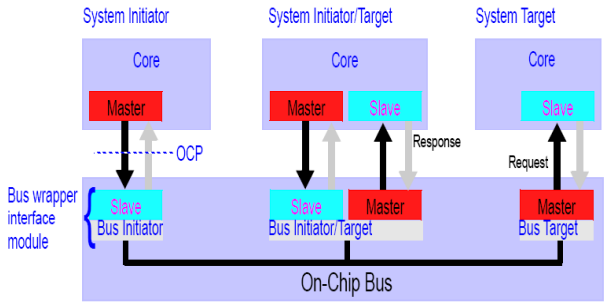

Figure 4: Basic block diagram of OCP instance

Figure 4 shows a simple system containing a wrapped bus and three IP core entities such as one that is a system target, one that is a system initiator, and an entity that is both. The characteristics of the IP core determine whether the core needs master, slave, or both sides of the OCP [5] and the wrapper interface modules must act as the complementary side of the OCP for each connected entity. A transfer across this system occurs as follows.

A system initiator (as the OCP master) presents command, control, and possibly data to its connected slave (a bus wrapper interface module). The interface module plays the request across the on-chip bus system. The OCP does not specify the embedded bus functionality. Instead, the interface designer converts the OCP request into an embedded bus transfer. The receiving bus wrapper interface module (as the

OCP master) converts the embedded bus operation into a legal OCP command. The system target (OCP slave) receives the command and takes the requested action.

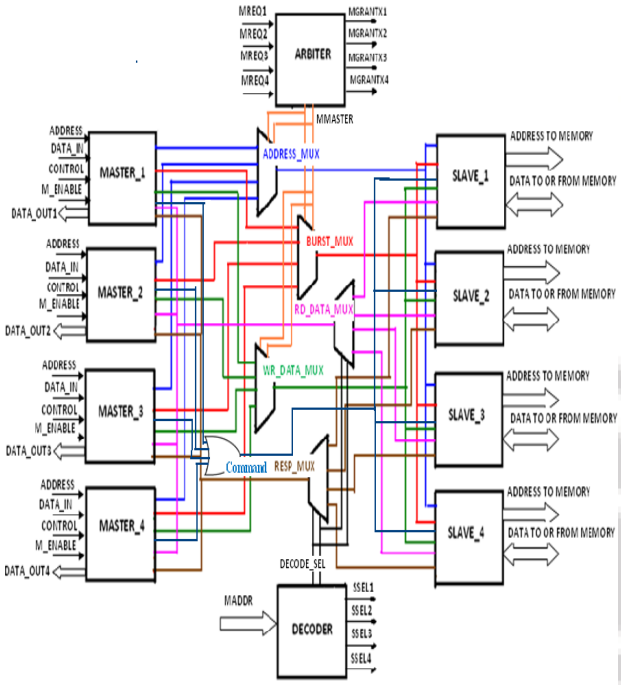

Figure 5: Architecture of Proposed OCP

4. Hardware Design of the On- Chip Bus

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.

4.1 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.

4.2 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.

4.3 FSM-M & FSM-S

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.

A. Experiment and analysis

The module open core protocol (ocp) has been designed using verilog,vhdl code. The functionality of the design was verified using modelsim and synthesized using Xilinx ISE design suite 12.1 version.

B. Simulation and Synthesis Results

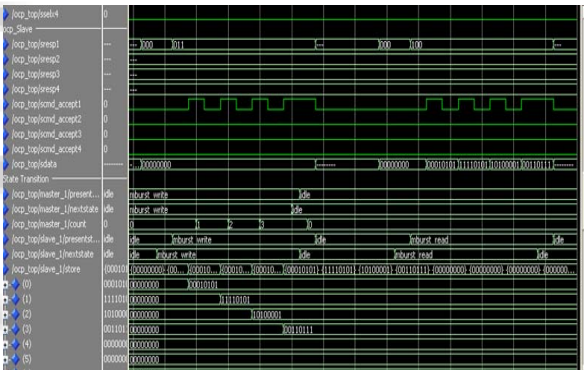

Figure 6: Simulation result of 4-bust operation

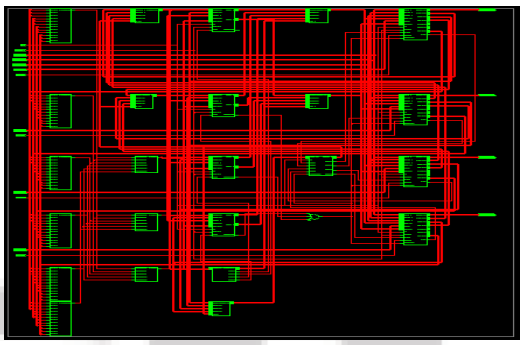

Figure 7: Synthesis of OCP

A multimedia SOC design [8] is used that contains an ARM9 processor, a Parallel Architecture Core (PAC) DSP processor, a RAM, a ROM and some peripheral devices. Originally the on-chip bus is a multi-layer bus consisting of two AHB-lite busses modeled with a commercial bus library (CoWare) The ARM9 processor is connected to one AHB-lite bus, and the PAC DSP processor is connected to the other. In the experiment, we replace the AHB-lite bus connecting the PAC DSP processor with the proposed on-chip bus and verify the whole SOC design with an H.264 decoding procedure.

Experimental results show that the proposed on-chip bus deals with all the communications in the SOC well. The simulation times of decoding one frame are about 48.6 and 44.3 seconds before and after the replacement, respectively. It should be pointed out that even though both the proposed bus and the one using CoWare bus library are cycle-accurate; our

proposed bus is further a pin-accurate one but the one from CoWare is not. To focus on the efficiency evaluation of the crossbar bus architecture, we design an SOC system which contains four masters (IP1 and IP2, IP3, IP4) and Four slaves (IP5, IP6 and IP7, IP8). In this experiment we compare the communication efficiency of the SOC systems with the shared bus architecture and the crossbar bus architecture. The transactions used in the experiment are described as follows.

1) Master IP2 first requests a series of burst WR transactions to slave IP6and a WR transaction to slave IP7.
2) Master IP1 then requests a series of burst RD transactions to slave IP6.
3) Master IP1 then requests a series of burst WR transactions to slave IP8, and master IP2 requests a series of burst WR
4) Transactions to slave IP6.
5) Master IP2 requests a series of burst RD transactions to slave IP8.

The procedure from step2 to step4 is iterated for 100 iterations.

The execution of these transactions on the crossbar architecture can be illustrated where parallel communication between different masters and slaves happens in time intervals .The behavior on the share bus architecture is all in series without any parallel communication. Experimental results show that the proposed bus with crossbar interconnection reduces about 23.35% communication cycles as comparing to the tradition shared bus architecture such as AHB.

In the last experiment, we evaluate the efficiency of out-of order transactions using the proposed scheduler. We assume that the access latency of the slave IP5 and IP6 in are 1 cycle and 3 cycles, respectively. The slave IP8 has the access latency of 5 cycles when it is accessed by master IP1 and two cycles when it is accessed by master IP2. In the simulation, IP1 first requests a series of out-of-order WR transactions, IP2 then requests a series of out-of-order RD transactions. After that, IP2 requests a series of out-of-order WR transactions, and IP1 finally requests a series of out-of order RD transactions. The simulation results show that when each series of these requests contain 6000 transactions, the proposed scheduler reduces 67.16% communication cycles as comparing to a bus that supports only in-order transactions.

4.4 Application

Since it is an IP block, it can be used in any kind of SOC Application. The application can be listed as follows.

- SRAM
- Processor

5. Conclusion

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 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.

References

[1] 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] 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
[4] Level Modeling of an Extended AMBA2.0 Bus Architecture," *Design, Automation, and Test in Europe*, pages 138-139, 2005.
[5] G. Schirner and R. Domer, "Quantitative Analysis of Transaction Level Models for the AMBA Bus," *Design,*
[6] *Automation, and Test in Europe*, 6 pages, 2006.
[7] C.-K. Lo and R.-S. Tsay, "Automatic Generation of Cycle Accurate and Cycle Count Accurate Transaction Level Bus
[8] Models from a Formal Model," *Asia and South Pacific Design Automation Conference*, pages 558-563, 2009.
[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] IBM Corporation, "Prioritization of Out-of-Order Data Transfers on Shared Data Bus," *US Patent No. 7,392,353*, 2008.
[11] CoWare website, http://www.coware.com