Venkatarao Gandi, M.Kedareswara Rao / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 3, Issue 1, January -February 2013, pp.903-907

Design of On Chip Bus Using Core Centric Protocol

Venkatarao Gandi, M.Kedareswara Rao
Electronics and Communications Engineering, Avanthi institute of engineering and technology
Instrumentation and control systems, Avanthi institute of engineering and technology

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 work the well-defined interface standard is adopted, the Open Core Protocol (OCP), and focus on the design of the internal bus architecture. 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.

I. 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, 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) 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). 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 fast and accurate simulation models at high level but did not provide real hardware implementation details. In [6], 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. Nevertheless, most bus design techniques developed in this paper can also be applied to the AXI bus. 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. In addition, the proposed bus is flexible such that one can adjust the bus architecture according to the system requirement. One key issue of advanced buses is how to manipulate the order of transactions such that requests from masters and responses from slaves can be carried out in best efficiency without violating any ordering constraint. In this work we have developed a key bus component called the scheduler to handle the ordering issues of out-of-order transactions. We will show that the proposed crossbar/partial-crossbar bus architecture together with the scheduler can significantly enhance the communication efficiency of a complex SOC. Another notable feature of this work is that we employ both transaction level modeling (TLM) and register transfer level (RTL) modeling to design the bus. We start from the TLM for the consideration of design flexibility and fast simulation speed. We then refine the TLM design into synthesizable and cycle-accurate RTL codes which can be synthesized into gate level hardware to facilitate accurate timing and functional simulation. The proposed bus has been employed in a multimedia SOC design and the results show that not only our TLM model has better simulation efficiency comparing to a bus obtained through a commercial ESL tool, but also our RTL on-chip bus design performs much more efficient than the multiplexer-based buses or those without out-of-order feature in real SOC design. The remainder of this paper is organized as follows. The various advanced functionalities of on-chip buses are 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.

B. On-Chip Bus Functionalities

We first describe the various bus functionalities including
1) Burst, 2) lock, 3) pipelined, and 4) out-of- order transactions.

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

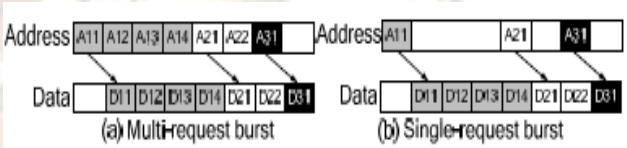


FIGURE 1. Burst transactions

II. 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 (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 there turn of data requested by A11 (i.e., D11-D14).

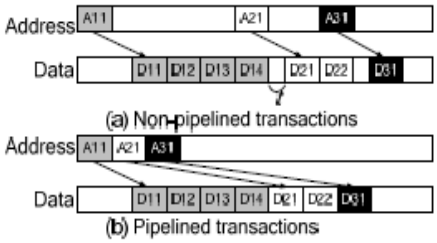


FIGURE 2. Pipelined transactions

III. 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.

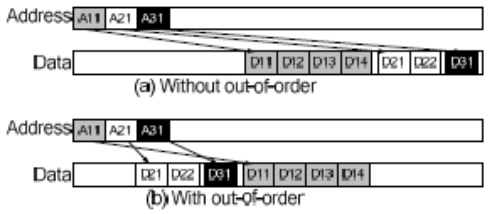


FIGURE 3. Out-of-order transactions

C. Hardware Design of the On-Chip Bus

The architecture of the proposed on-chip bus is illustrated in FIGURE 3, where we show an example with two masters and two 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.

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.

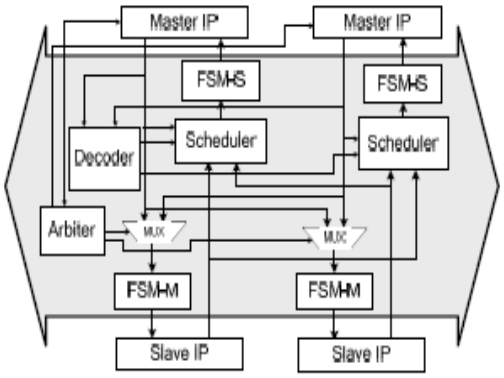


FIGURE 4. Top view of the proposed bus architecture

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 non existent and responses with an error message if necessary.

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.

4. Scheduler

Out-of-order transactions in either OCP or AXI allow the order of the returned responses to be different from the order of the requests. In the OCP protocol, each out-of order transaction is tagged with a Tag ID by a master. For those transactions with the same Tag ID, they must be returned in the same order as requested, but for those with different Tag ID, they can be returned in any order. In general, both in

order and out-of-order transactions are supported in an out-of order SOC system. Whether to favor in-order or out-of-order transactions is a design issue of the bus. It is stated that conventional bus scheduling algorithms tend to favor the in-order transactions, while the ordering mechanism proposed in favors out-of order transactions. In our proposed scheduler, we reserve the flexibility of being in-order response first or out-of-order response first, which means that system integrators are allowed to select either order based on the applications. The architecture of the proposed scheduler is shown in FIGURE 5.

A multiplexer, MUX1, 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. The function of MUX2 will be described shortly. The recorder shown in the figure is used to keep track of the ID of the target slave and the Tag ID of every out-of-order transaction. Whenever a response arrives, the comparator determines whether the ordering restriction is violated or not by comparing the ID of the target slave and Tag ID. If no ordering restriction is violated, the response is sent forward to the priority setter. If the restriction is violated, the response is sent backward to one of the inputs of MUX2, which is always a preferred input over the input from MUX1.The responses sent forward are given a priority, which is different from the slave priority, according to the Tag ID and are stored in the priority queue. For the transactions without Tag ID, which are regarded as in-order transactions, the priority setter sets the priority to 0 or the largest value to reflect whether in-order first or out-of-order first policy is used. Finally, the responses stored in the priority queue are returned to the masters from the first priority to the last priority such that the objective of "transactions with the same Tag ID are returned in-order, and transactions with different Tag ID can be returned out-of-order" can be achieved. To further improve the efficiency of the scheduler, the response can be forwarded to the master directly without going through the priority queue when the priority queue is empty.

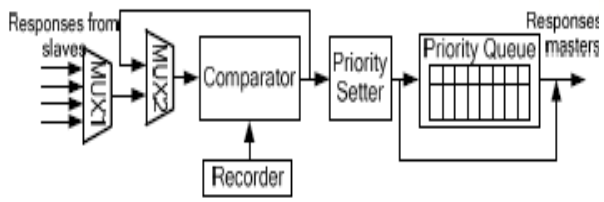


FIGURE 5. Block diagram of the scheduler

D. Simulation result for simple write and read

The above developed FSM for the OCP Master and Slave which supports the simple write and read operation is designed using VHDL and is simulated. The designed OCP master and slave are integrated as a single design and is simulated waveform represents the complete transaction of simple write and read operation from master to slave and vice-versa which is shown in Figure 6

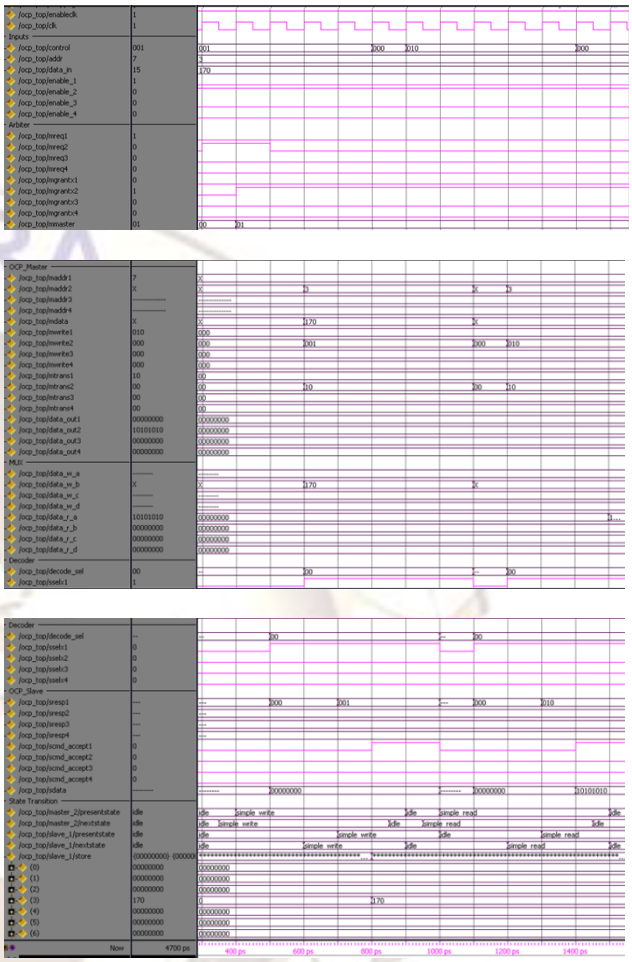


Figure 6. Waveform for OCP master and slave simple write and read

E. Simulation results for burst operation

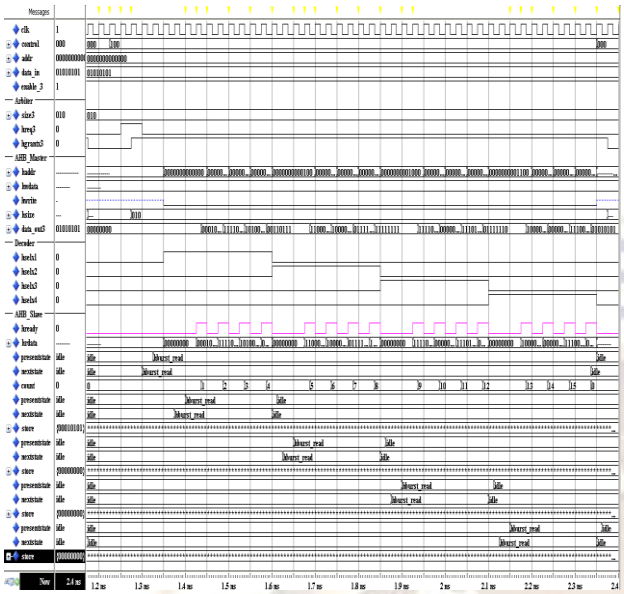
The basic working of OCP master and salve is discussed based on their FSMs and in the design totally four OCP master and slave are present. OCP supports burst size of only 4, 8 and 16.

1. Burst operation of size_16

The simulation result for the OCP master and slave of burst size 16 is shown in the Figure 7.

The size is given as "010" which represents the burst size 16 and hence four continuous write or read operation happens. Here the count is introduced in order to generate the address with respect the given initial address and the count increment. The operation remains the same as simple read and write but the only change is that after each operation, count will check for the burst size. When the count is not equal to the burst size given, the count will get incremented and the next address is get generated based on which

the read or write operation that currently performed is carried out. When the count is equal to burst length, that represents the burst operation over and count resets to zero. Hence master and slave go IDLE state.



IV. Conclusions

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. This OCP will include two types of operation such as Simple Write and Read and Burst Operation.

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

[7] IBM Corporation, "Prioritization of Out-of-Order Data Transfers on Shared Data Bus," US Patent No. 7,392,353, 2008.

[8] David C.-W. Chang, I.-T. Liao, J.-K. Lee, W.-F. Chen, S.-Y. Tseng and C.-W. Jen, "PAC DSP Core and Application Processors," International Conference on Multimedia and Expo, pages 289-292, 2006.

[9] CoWare website, http://www.coware.com