International Journal of Electronics Communication and Computer Engineering
Volume 3, Issue 4, ISSN (Online) 2249 –071X, ISSN (Print): 2278 – 4209

Advanced On-Chip Bus Design with Open Core Protocol Interface

K. Anjaiah
Roll. No. 10A01D5715
M.Tech., (VLSID)
Nova College of Engg. & Tech.
Vegavaram, Jangareddygudem
West Godavari District
Andhra Pradesh, India

G. Sravya
M.Tech.
Guide & Associate Professor in ECE
Nova College of Engg. & Tech.
Vegavaram, Jangareddygudem
West Godavari District
Andhra Pradesh, India

***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 we adopt the well-defined interface standard, the Open Core Protocol (OCP), and focus on the design of the internal bus architecture. We develop an efficient bus architecture to support most advanced bus functionalities defined in OCP, including burst transactions, lock transactions, pipelined transactions, and out-of-order transactions. We first model and design the on-chip bus with transaction level modeling for the consideration of design flexibility and fast simulation speed. We 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* – On-Chip Bus Design, Open Core Protocol.**

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.

The 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.

The work presented in and develops high-level AMBA bus models with fast simulation speed and high timing accuracy. The authors in propose an automatic approach to generate high-level bus models from a formal channel model of OCP. 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 RT L 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. Conclusions are then drawn in Section 5.

II. On-Chip Bus Functionalities

We first describe the various bus functionalities including
1) burst, 2) lock, 3) pipelined, and 4) 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.

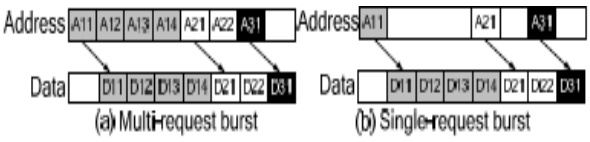

Fig.1. Burst Transactions

- *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 (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).

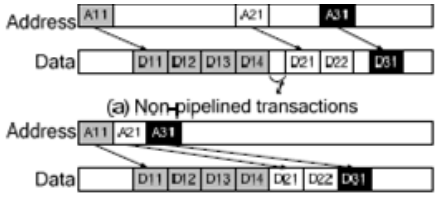

Fig.2. Pipelined Transactions

- *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 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.

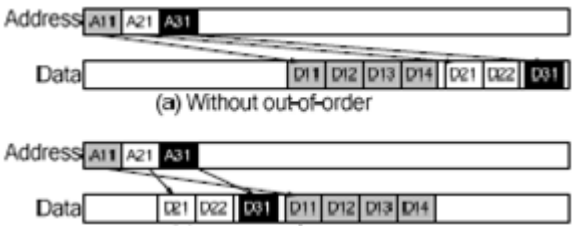

Fig.3. Out-of-order Transactions

III. Hardware Design Of On-Chip Bus

The architecture of the proposed on-chip bus is illustrated in FIGURE 4, 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.

Arbiter

In traditional shared bus architecture, resource contention happens whenever more than one master

International Journal of Electronics Communication and Computer Engineering
Volume 3, Issue 4, ISSN (Online) 2249 –071X, ISSN (Print): 2278 – 4209

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.

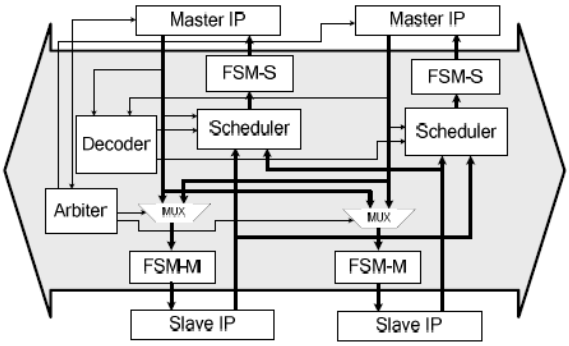

Fig.4. Top view of proposed bus architecture

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.

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.

Scheduler

Out-of-order transactions in either OCP [2] or AXI [1] 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 TagID by a master. For those transactions with the same TagID, they must be returned in the same order as requested, but for those with different TagID, 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. In [7] it is stated that conventional bus scheduling algorithms tend to favor the in-order transactions, while the ordering mechanism proposed in [7] 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 4.

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 TagID 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 TagID. 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 TagID and are stored in the priority queue. For the transactions without TagID, 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 fro m the first priority to the last priority such that the objective of "*transactions with the same TagID are returned in-order, and transactions with different TagID 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.

IV. Experimental Results

We design the proposed bus with both TLM and RTL models. The RTL model can be further synthesized into gate-level description. The specification and the synthesis results when 4 masters and 6 slaves are used are shown in Table1, where all masters can issue burst, lock, pipelined and out-of-order transactions.

Bus architecture	Crossbar
Address width	32-bit
Data width	64-bit
Number of slave ports	6
Number of Master ports	4
Depth of Priority queues	4
Operating frequency	333 MHz.
Synthesis process	TSMC 0.13µm CMOS process
Total ares (um^2)	153,832 (about 30K gates)

Table 1: Bus specifications & synthesis results

The design of the developed FSM is done using VHDL and the integration of the master and slave is made. The simulation result of the integrated design gives the clear view on the OCP Burst Write operation.

Here the burst length is given as 8 and hence the address will be generated for number of memory locations that equals to the burst length and the corresponding input data is given. 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.

V. 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.
- 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.
- The simulation result shows that the communication between different IP cores using OCP is proper.
- 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 OCP supports the simple write and read operation and the burst extension.
- Based on the result obtained, the burst extension is seen to automate the address generation. The initial address alone is provided to the protocol.
- The Various Scenarios for each component in the OCP design are verified effectively during the simulation with respect to its behavior.

VI. FUTURE SCOPE OF THE WORK

- The design can be further extended by developing a total system around it.
  - For Example, we can use this protocol to interface between an ARM processor and any device (like SRAM) provided both the IP cores should have OCP Compliance.
- Burst Mode can be extended further to include various supported types of burst.
- Thread and Tag extension can also to include in this developed protocol with the corresponding supporting signals.

This project work provides an ideal platform for enhancement or further development of the OCP.

REFERENCES

[1] Open Core Protocol (OCP) Specification, http://www.ocpip.org/home.
[2] CoWare website, http://www.coware.com
[3] 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.
[4] Sudeep Pasricha & Nikil Dutt "On-chip communication architectures (system on-chip interconnect)"
[5] N.Y.-C. Chang, Y.-Z. Liao and T.-S. Chang, "Analysis of Shared-link IET Computers & Digital Techniques, Volume 3, Issue 4, pages 373-383, 2009.
[6] VHDL Primer by J.Bhasker
[7] VHDL analysis and modelling of Digital systems by Nawabi
[8] Logic Design Principle by Jr. Roth
[9] Field Programmable Gate Array Technology - S. Trimberger, Edr, 1994, Kluwer Academic Publications.
[10] Engineering Digital Design - RICHARD F.TINDER, 2nd Edition, Academic press.
[11] Fundamentals of logic design -Charles H. Roth, 4th Edition Jaico Publishing House.
[12] Field programmable gate array - S. Brown, R.J.Francis, J.Rose , Z.G.Vranesic, 2007, BSP.
[13] Digital Design Using Field Programmable Gate Array, - P.K.Chan & S. Mourad, 1994, Prentice Hall.

AUTHOR'S PROFILE

K. Anjaiah
Roll. No. 10A01D5715
M.Tech., (VLSID), P.G.D.C.M.P
Nova college of Engg. & Tech.
Vegavaram, Jangareddygudem
West Godavari District
Andhra Pradesh, India

G. Sravya
M.Tech.
Guide & Associate Professor in ECE
Nova College of Engg. & Tech.
Vegavaram, Jangareddygudem
West Godavari District
Andhra Pradesh, India