ISSN 23499842(Online), Volume 1, Special Issue 2(ICITET 15), March 2015
International Journal of Innovative Trends and Emerging Technologies

DESIGN OF NOCs TO AVOID DEADLOCK PROBLEM IN ON-CHIP BUSES

N.PRIYADHARSHINI[1], S.KALAISELVY[2]
[1]PG Student, VLSI Design, 2Assistant Professor (Sr.G), [1,2]Department of ECE,
[1,2]Surya Group Of Institutions, India.

Abstract*:*
The modern on-chip has increased their application thereby increasing the transaction in the NoCs. We have improved the communication efficiently by using advanced extensible interface (AXI) and open core protocol(OCP). The increasing transaction causes the deadlock problems in SoC. If the transaction are not properly accessed the deadlock problem occur. A deadlock problem occurs if a cycle exists in the bus. The transaction is represented as graph to resolve the deadlock problem. We propose deadlock free transaction by logical analysis.

I. INTRODUCTION

With the rapid increase in the modern electronic systems, more IP cores are embedded in the system on-chip Designs. This increasing core in the system causes incredible increase in the transaction. Therefore, the major factor is designing the communication architecture for the modern electronic systems which dominates the overall performance of the system. In the early periods the popular communication used are the advanced peripheral buses and advanced high-performance bus. Both these buses use same technique, one master controlling only one slave. The communication protocols in AXI [1] and OCP [2], supports many advanced transaction. The advanced transaction is burst, pipeline and out-of-order transaction. Among these transaction out-of-order transaction is the more efficient transaction [4][5]. In Out-of-order transaction data does not wait for the previous transaction to complete. Though the out-of-order transaction is fast than others but it pays ways for deadlock problems. The deadlock problem is when the transaction stalls and forms a wait and hold state. The wait-and-hold is a situation when a cycle exit in the transaction and a loop occurs in the system. Thus the data are not transmitted and locked in the loop. This may crash the whole system [6]. The master when accessing a slave it sends a request to the slave. The slave in turn sends a response to the master. After the response are returned and accepted the transaction is complete. After the transaction is complete the master should release the slave. In AXI and OCP the master tags an ID to the transaction in such a way all the request and response transaction are accessed in ID order. This makes the transaction much more delay [6]. In this paper, we look over the deadlock problems. To overcome the deadlock problem first we develop a graphical representation of the bus. The bus status model has the model of the master and the slave transaction. If a cycle exists in the transaction then the system is called unsafe state that results in deadlock. Based on this problem we propose a technique to resolve the deadlock thus achieving greater communication efficiency .In the existing system, they addressed the deadlock problem in an on-chip bus system supporting out-of-order transactions. They presented a graphic model that can well represent the status of a bus system and showed that a cycle exists in the graph if and only if the bus system is in an unsafe state that may lead to a bus deadlock. Based on this model, they proposed a novel bus design technique that can efficiently resolve the bus deadlock problem..

II. RELATED WORK

The concept of memory access scheduling in which DRAM operations are scheduled, possible completing memory references out of order to optimize memory system performance was presented. A priority expression which considers three factors: wait time of a burst, burst length, priority of read or write accesses. The expression is used to select a burst from the write or read queue for bank arbitration. The proposed AXI bus possesses multiple independent channels to support multiple simultaneous address and data streams. A shared-link AXI interconnect can provide good performance while requiring less than half of the hardware required by a crossbar AXI implementation. The performance analysis of a shared-link AXI was presented. This paper proposes the issues and share experiences on using Open Core Protocol (OCP) as the standard interface protocol, defining reusable profiles to fit different IPs, on-chip interconnection design, verification, and SoC integration with them was presented. The usage of OCP as an interface Standard was given. The establishment of profiles is proposed for easy adoption and adaptation. Bus fabric design schemes are demonstrated to show the simplicity of interconnection IP design using these profiles. proposed for realizing high-performance SoCs, it is crucial for the communication architecture to be highly customized towards application traffic profiles. Since the communication requirements of SoC components can vary significantly over time, communication architectures that dynamically detect and adapt to such variations can substantially improve system performance. Thus the FLEXBUS, architecture capable of dynamically controlling both the communication topology, and the mapping of components to the communication architecture was presented.

III. NETWORK ON CHIP

Network on chip or network on a chip (NoC) is a communication subsystem on an integrated circuit typically between IP cores in a system on a chip (SoC). Network on chip is an emerging paradigm for communication within large VLSI systems implemented on a single silicon chip.

a) NOC topology

i) 2D Mesh

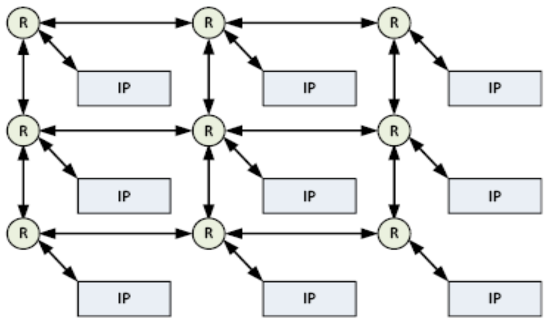


Fig 1. 2D-Mesh topology

ii) Star topology

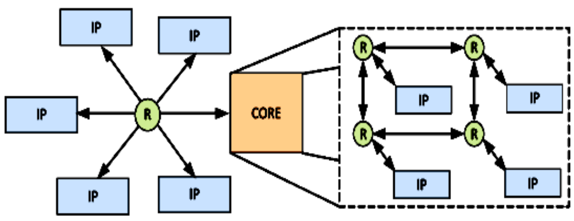


Fig 2. Star topology

b) Bus transaction

The basic bus transactions are single and burst transactions, where a single transaction is one that requests only one response whereas a burst transaction is one that requests multiple responses. These basic transactions can be either pipelined or no pipelined. As commonly recognized, a pipelined transaction is one that can be issued before its preceding transaction is completed. As an out-of-order transaction is one that may be completed without waiting for its preceding transactions, all out-of-order transactions can be considered as pipelined ones. In AXI or OCP, the pipelined transactions can be further divided into tagged and untagged ones, where all untagged transactions must be executed in order, whereas the execution orders of the tagged ones depend on the IDs they are tagged. In this paper, we comply with the following order constraints.

1) All untagged transactions must be executed in order.

2) All tagged transactions with the same tag ID must be executed in order.

3) Two transactions tagged with different IDs can be executed out of order.

4) There is no order restriction on the execution between one tagged transaction and an untagged transaction

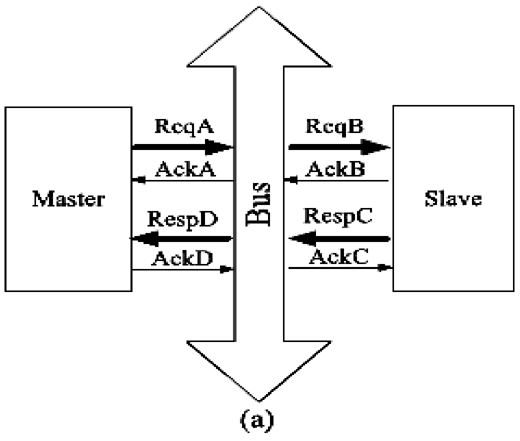


Fig 3. Basic Bus Transaction

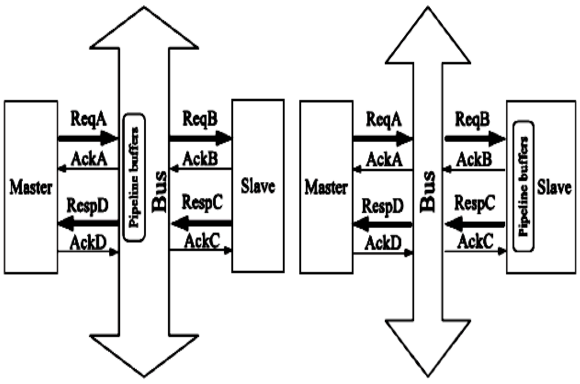


Fig 4. Pipelined Transactions with Buffers in the Bus, And and Pipelined Transactions with Buffers in the Slave.

c) Bus Dead lock

Bus deadlock is a problem that occurs when a set of IP cores communicating through a bus system is involved in a circular wait-and-old state that cannot be resolved. This problem may crash a bus system as none of the IP cores involved in the deadlock can continue its functions. In, the authors invest the bus deadlock problem of a system that allows a master to execute a process only if the master is granted to access all required slaves of the process and each master will hold the slaves granted to it until all its required slaves are granted to it. A bus deadlock happens when each master in a set of masters is holding a slave and waiting for another slave held by another master in the set. In this type of bus deadlocks, the relation between masters and slaves is similar to that between processes and resources in an operating system (OS) where a deadlock occurs when there is a circular wait-and-hold relation among a set of processes and resources. A resource allocation graph (RAG) is commonly utilized to represent the status of resource allocation in an OS. A vertex in a RAG represents a process or a resource. A directed edge from a process vertex to a resource vertex denotes that the process is requesting the resource, and one from a resource vertex to a process vertex denotes that the resource is being held by the process. In an OS, a deadlock may occur when a cycle exists in the RAG. To formally describe the deadlock problem considered, throughout this paper we make the following assumptions about the bus systems. 1) All components in a bus-based system, including masters, slaves, and buses, are compatible with OCP or AXI protocols. In particular, we assume that a slave must return responses of transactions with the same tag ID in order as mentioned before. 2) An arbiter grants a master to access a slave only when the master has the highest priority among the masters being requesting to access the slave and the slave is available to process the request. As a result, when a slave accepts a request from a master, it will return the response after some finite latency. 3) When a response returned from a slave violates any order constraint, the bus is responsible for avoiding the violation by taking some appropriate action such as buffering or not accepting the response. Buffering the responses that cannot be accepted may require large area overhead and thus in this paper, we assume that the bus will not accept any response that violates any order constraint.

IV. ON CHIP BUS DESIGN

Compared to previous bus designs, the supporting of various advanced transaction types in OCP and AXI has emphasized. As shown in Fig.5 our bus supports single, burst, pipelined (outstanding), and tagged transactions. We divide the components of the bus system into three parts:
1) The components for each master interface;
2) The components for each slave interface; and
3) The components in the center that will be shared by all masters and slaves. Fig. 3.3 shows a bus system with one master and one slave. If l masters (m slaves) are to be employed, l (m) copies of the components in the master (slave) interface should be employed, whereas only one copy of the components in the center is needed. A tagged transaction starts with a master issuing a request with an ID to the bus. In the request phase, if the Request Buffer in the bus is not full, the bus acknowledges the master and the request is stored in the Request Buffer. The Decoder then decodes the transaction address of the request, and the Arbiter arbitrates whether the request can be granted to access the target slave. If it can be granted, the Arbiter forwards the request to the slave by controlling the corresponding multiplexors, and the index of the target slave is recorded in one of the Recorders, in the way that the transactions with the same tag are recorded in the same Recorder The number of Recorders is equal to the number of IDs that the corresponding master can assign. Also the size of each Recorder is equal to the pipeline depth such that it is just enough to record all transactions that are not completed. The Request Busy Checker checks whether the request is completed or not to assist the arbitration. For a write request, the corresponding write data are stored in the Write Data Buffer, and the Tagged Transaction Access Controller controls the multiplexors to decide from which master the write data are to be provided. After the slave accepts the request, it acknowledges the bus, and the acknowledgment is forwarded to the Request Buffer under the control of the Arbiter. Finally, the response is sent to the master when the master is able to receive it. In the bus model, the masters and slaves are connected by the bus in a crossbar manner and thus parallel transactions can be executed as long as no contention occurs. When more than one request from different masters to access the same slave arrives simultaneously, the arbiter will determine whether the request with the highest priority can be granted or not. If it cannot be granted, the arbiter will start a new arbitration with possibly some priority updating (such as round robin). If it can be granted, the request will be forwarded to the corresponding slave. Once a request is granted, it can be processed in parallel with other granted requests. As a result, high communication parallelism can be achieved by this bus design.

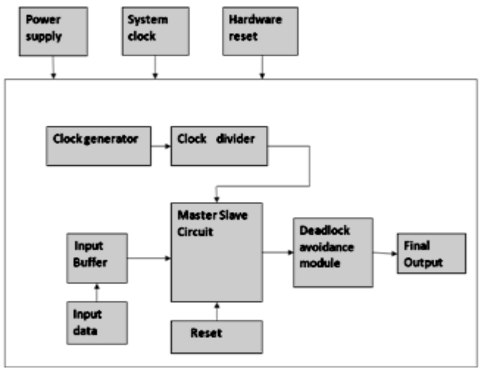


Fig 5. Bus Supporting Tagged Transactions

V. DEADLOCK AVOIDANCE CONCEPT

We will use the designs with two IDs and two slaves in the following description to illustrate the various deadlock avoidance schemes. The single slave scheme only allows tagged requests to access the same slave.

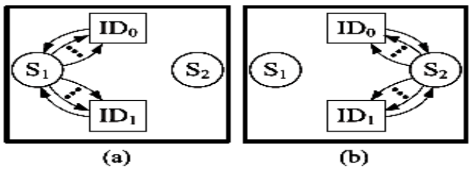
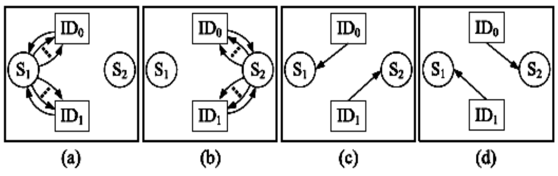


Fig 6 Legal requests under single slave scheme.



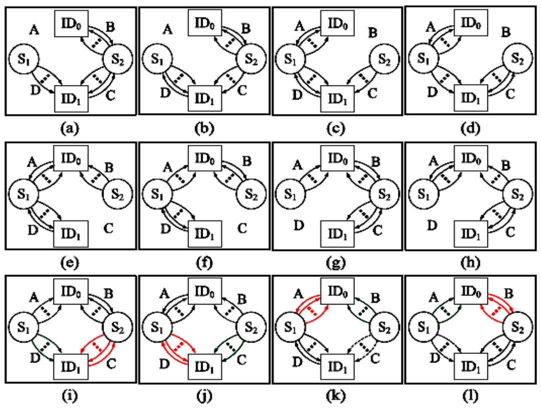


Fig 7 Legal requests under our DALS

VI. HARDWARE IMPLEMENTATION OF DALs

The hardware design of our DALS attempts to stall the least number of transactions. In addition, we also target to complete the decision of whether to stall a transaction in one clock cycle. We propose a hardware implementation that contains a number of waiting relation detectors and a number of unsafe state predictors which altogether requires only one clock cycle for the stall decision. Each waiting relation detector determines whether a pair of slaves has a waiting relation between them. According to the results of these detectors, the unsafe state predictors predict whether forwarding a request will result in an unsafe state. These designs are detailed next..

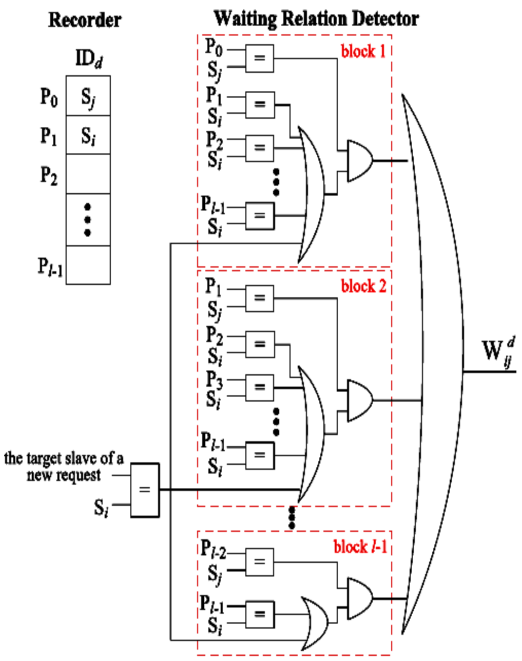


Fig 8 hardware *implementation of DAL*

1) Waiting Relation Detectors: As mentioned in Section III, we allocate a recorder for each ID in the bus system, and the index of a target slave of a tagged transaction is recorded in a corresponding recorder when the transaction is accepted by the slave. Responses from different slaves with the same ID must be returned in the order that they are recorded in the recorder. The hardware implementation of the waiting relation detector detecting Wdij is shown in Fig. 3.18. The left side shows a recorder to record the transactions with tag IDd . The recorder is implemented using a shift queue with the following features.

1) The indices of the slaves accepted are put in the recorder in a first-in-first-out manner.

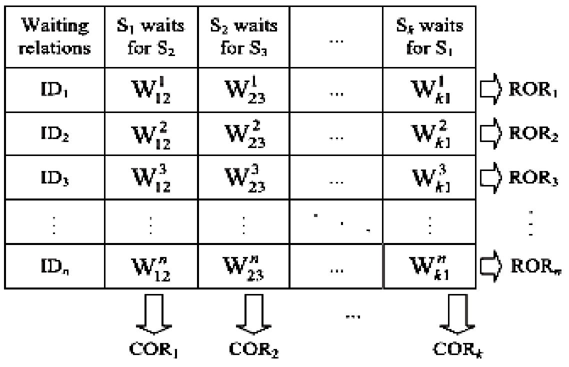
2) The first entry (P0) always contains the index of the slave that accepts the earliest transaction among all the accepted but not completed transactions with tag IDd , which is corresponding to the prime edge associated with IDd in the corresponding BSG.

3) When the response from the slave recorded in P0 is returned, a shift operation is performed to remove the index of the slave from the recorder and all the indices of the slaves in the remaining entries of the buffer are shifted toward P0 by one position.

The P0 and P1 entries of the IDd recorder in Fig. 13 shows that a request to Sj with IDd is first accepted by Sj and then a request to Si with IDd is accepted by Si. Now a new transaction also tagged with IDd is requested. The proposed waiting relation detector will detect the waiting relations of already accepted requests as well as the waiting relations if the new request is accepted as described below.

TABLE I

A) Waiting table



Waiting relations	S_1 waits for S_2	S_2 waits for S_3	...	S_k waits for S_1	
ID_1	W^1_{12}	W^1_{23}	...	W^1_{k1}	ROR_1
ID_2	W^2_{12}	W^2_{23}	...	W^2_{k1}	ROR_2
ID_3	W^3_{12}	W^3_{23}	...	W^3_{k1}	ROR_3
⋮	⋮	⋮	⋱	⋮	⋮
ID_n	W^n_{12}	W^n_{23}	...	W^n_{k1}	ROR_n
	COR_1	COR_2	...	COR_k	

TABLE II

B) *Comparison of bus performance with various bus deadlock techniques under various numbers of slaves*

COMPARISON OF BUS PERFORMANCE WITH VARIOUS BUS DEADLOCK TECHNIQUES UNDER VARIOUS NUMBERS OF IDs

Bus Performance Index (Number of Stalled Requests)	4 IDs and 8 Slaves					6 IDs and 8 Slaves					8 IDs and 8 Slaves				
	0 (10000)	2000 (151593)	4000 (269146)	6000 (451497)	8000 (556877)	0 (10000)	2000 (154168)	4000 (277314)	6000 (411974)	8000 (553741)	0 (10000)	2000 (155976)	4000 (270438)	6000 (424569)	8000 (541837)
DALS	1 (2703)	1 (52017)	1 (80412)	1 (156318)	1 (182315)	1 (2627)	1 (50811)	1 (75431)	1 (121873)	1 (165913)	1 (2597)	1 (51993)	1 (78914)	1 (134912)	1 (146007)
Single slave [9]	0.700 (7534)	0.676 (118327)	0.637 (203148)	0.677 (347114)	0.658 (443187)	0.419 (8147)	0.427 (124357)	0.408 (212407)	0.409 (344711)	0.400 (473201)	0.287 (8591)	0.272 (140741)	0.276 (228476)	0.291 (357701)	0.282 (471253)
Unique ID [9]	0.942 (4614)	0.912 (74135)	0.892 (135471)	0.930 (220417)	0.950 (289123)	0.667 (6011)	0.690 (88125)	0.681 (150736)	0.684 (240871)	0.713 (332478)	0.462 (6787)	0.479 (104048)	0.453 (176044)	0.476 (278137)	0.465 (360074)
Hybrid [9]	0.987 (4124)	0.957 (64298)	0.953 (117521)	0.972 (191428)	0.984 (246947)	0.700) (4997)	0.713 (73110)	0.695 (12801)	0.703 (209140)	0.709 (280114)	.518 (5813)	0.546 (87761)	0.532 (146977)	0.519 (237114)	0.528 (304155)

V. SIMULATION RESULT

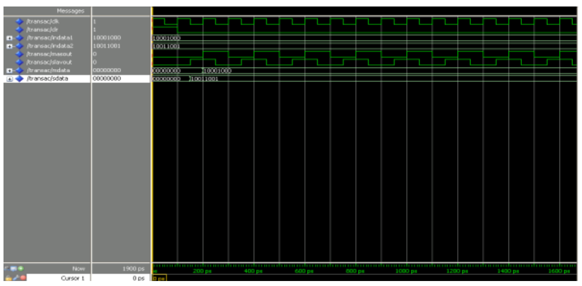


Figure 5.1 A Master To Slave Transaction

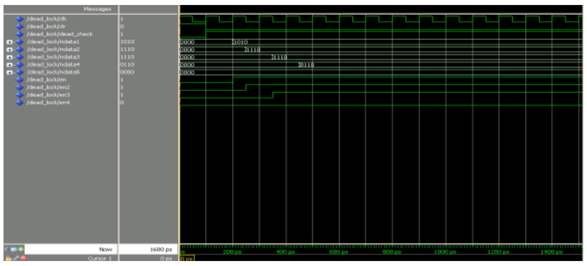


Figure 5.2 A Deadlock In The Transaction

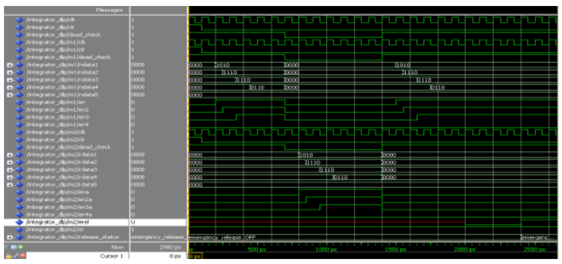


Figure 5.3 Deadlock release

VI. CONCULSION

In this project, we had designed a Master – Slave transaction module which transmits the data when the corresponding requests and responses are performed. The slave works when the master makes a request. Likely, we had designed a deadlock occurrence module and a method to release that deadlock is also designed. This deadlock release module which clears the condition for the occurrence of deadlock. Thus all these designs are designed and verified successfully using Modelsim Simulator.

REFERENCES

[1] Advanced Microcontroller Bus Architecture Specification. (1997)
[Online]. Available: http://www.arm.com
[2] Open Core Protocol Specification. (2006)
[Online]. Available:http://www.ocpip.org/home

[3] A. T. Tran and B. M. Bass, "RoShaQ: High-performance on-chip routerwith shared queues," in Proc. IEEE 29th Int. Conf. Comput. Design, Oct. 2011, pp. 232–238.

[4] J. Shao and B. T. Davis, "A burst scheduling access reordering mechanism," in Proc. IEEE 13th Int. Symp. High Perform. Comput. Archit.,Feb. 2007, pp. 285–294.

[5] J. Pang, L. Yang, L. Shi, T. Zhang, D. Wang, and C. Hou, "A priorityexpression- based burst scheduling of memory reordering access," in Proc. Int. Conf. Embedded Comput. Syst., Archit., Model., Simul., Jul. 2008, pp. 203–209.

[6] X. Xiao and J. J. Lee, "A true O(1) parallel deadlock detection algorithm for single-unit resource systems and its hardware implementation," IEEE Trans. Parallel Distrib. Syst., vol. 21, no. 1, pp. 4–19, Jan. 2010.

[7] A. Silberschatz, P. B. Galvin, and G. Gagen, Operating System Concepts, 7th ed. New York, USA: Wiley, 1993.