International Journal of Engineering Research in Electronic and Communication Engineering (IJERECE) Vol 3, Issue 4, April 2016

On-Chip Bus Architecture for Achieving Deadlock Free Communication

[1]Neethu Susan Alex, [2]Karthika Manilal
[1] PG Student [VLSI and ES] [2] Assistant professor,
Department of ECE, TKM Institute of Technology, Kollam
[1] neethuvaidyan@gmail.com [2] karthikamanilal@gmail.com

Abstract: — **In modern electronic systems, as the computing requirement increase, more and more intellectual property (IP) cores are embedded in System-On-Chip (SOC). On-chip communication architectures play an important role in determining the over-all performance of SOC designs. Communication architectures should be flexible so as to offer high performance over a wide range of traffic characteristics. The modern communication protocol, AXI (Advanced eXtensible Interface) supports advanced transactions like out-of-order transactions that improve communication efficiency. However a deadlock situation may occur if these transactions are not handled properly. The deadlock may get occurred if each master in a set of masters is holding a slave and waiting for another slave held by another master in the set. Hence a bus architecture avoiding deadlock condition, based on AXI protocol has to be developed. The masters and the slaves communicate each other through the AXI interconnect having different channels. The address bits should be transmitted initially through the address channel and data will be transmitted after that through the data channel. The encoding and decoding of the data bits are done using DMC (Decimal Matrix Codes). For avoiding deadlock, a waiting relation detector can be used to find whether any waiting relation exists. Thus multiple transactions can be handled without any deadlock occurrence. The designing language is VHDL and synthesis in Xilinx ISE Design Suite 13.2 and can be simulated in ModelSim SE 6.3f.**

Index Terms—**advanced extensible Interface, Decimal Matrix Code, On chip communication, System on Chip**

I. INTRODUCTION

On-chip communication architectures determine the overall performance of the System-on-Chip (SoC) designs. The communication architectures should be flexible to offer high performance over a wide range of traffic characteristics. Since the communication requirements of on-chip components can vary significantly over time, bus architectures that dynamically detect and adapt to such variations will substantially improve system performance. Modern on-chip communication protocols should support advanced transactions including burst transactions, pipelined transactions and out-of-order transactions to improve the communication efficiency. In out-of-order transactions the responses will return in an order different from their request order. The communication protocols in AXI support various advanced transactions, such as burst transactions, pipelined transactions, and out-of-order transactions [1]. Among these, the out-of-order transactions serve as a major key in improving system performance. Out-of-order transactions can be executed more efficiently when a master IP, like a processor, can handle out-of-order responses as it allows a slave core such as a dynamic random-access memory controller to service the requests in an order that is most convenient, rather than the order in which requests are sent by the master.

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-hold state, which is difficult to resolve. This situation may crash a bus system as none of the IP cores involved in the deadlock problem can continue its functions. A bus deadlock happens when each master in a set of masters is holding a slave and is waiting for another slave held by another master in the set. In these bus deadlocks, the relation between masters and slaves is similar to that between the processes and the 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 [2]. Deadlock will decrease robustness of the system.

A deadlock situation may occur in a bus if out-of-order transactions are not properly handled. This is because in general a bus system supporting out-of-order transaction also has to support transactions that will not allow out-of-order execution. For example, a read operation has to wait for a preceding write operation if they address to same memory location. Such read-after-write operations are commonly seen

in a bus system and they do not allow out-of-order execution [2]. As a tagged transaction must wait for the completion of the earlier issued transaction with the same ID, it may happen that a set of slaves are waiting for one another in a circular way and are all blocked by the bus, thereby resulting in a bus deadlock.

II. EXISTING TECHNIQUES FOR DEADLOCK PREVENTION

A. *Cyclic Dependency Schemes*

In any interconnect which is connected to a slave that reorders read data or write response signals, there is the potential for deadlock. To prevent this three cyclic dependency schemes that enables the slave interface to accept or stall a new transaction address is provided [6]. Each slave interface can be configured to one of the following cyclic dependency schemes:

- Single Slave Scheme
- Unique ID Scheme
- Hybrid Scheme

Single Slave Scheme: This configuration implements a deadlock prevention technique that accepts or stalls a new transaction address based on the following rules:

- A master can initiate a transaction to any slave if the master has no outstanding transactions
- If the master does have outstanding transactions then the master can initiate a transaction to the same slave as the current outstanding transactions [6].

Unique ID Scheme: This configuration implements a deadlock prevention scheme that accepts or stalls a new transaction address based on the following rules:

- A master can initiate a transaction to any slave if the master has no outstanding transactions
- If the master does have outstanding transactions then the master can initiate a transaction to any slave but only if the transaction ID of the current transaction is unique, relative to current outstanding transactions [6].

Hybrid Scheme: This configuration implements a deadlock prevention scheme that accepts or stalls a new transaction address based on the following rules:

- A master can initiate a transaction to any slave if the master has no outstanding transactions
- If the master does have outstanding transactions then the master can initiate a transaction to the same slave as the current outstanding transactions, or the master can initiate a transaction to any slave but only if the transaction ID of the current transaction is unique, relative to current outstanding transactions [6].

B. *Event Deadlock Detector*

It is an apparatus for detecting a bus deadlock in an electronic system [8]. It includes a bus tracker circuit to monitor the bus transactions to detect a condition that indicates the occurrence of a wait cycle or a retry cycle. The apparatus also includes a counter circuit to indicate that the tracker circuit has detected the condition a predetermined number of times.

Fig. 1 illustrates one embodiment of the apparatus for detecting a deadlock on a bus that follows the protocol in Peripheral Component Interconnect (PCI) bus. The method embeds a bus tracker in the buses to monitor bus transactions. It includes an address circuit which includes memory circuit to store an address of a bus agent involved in a transaction. Address circuit also includes a comparator circuit to compare an address of a bus agent involved in a transaction with an address stored in a memory circuit. If the tracker detects a condition indicating some transactions is waiting or retrying, a counter with a predetermined number starts to count down.

When the counter is decreased to zero, it is regarded as deadlock occurs. With this method, a bus deadlock can be detected with a few clock cycles. However the predetermined number is difficult to determine appropriately [8]. The main drawbacks of the method are,

- If the number is too small, designer may encounter the

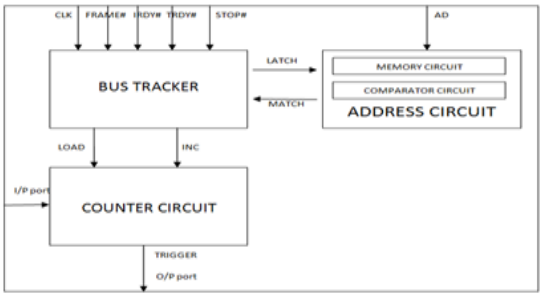

Fig. 1: Event deadlock detector

Problem of over detection, whereas if it is too large, the latency between the occurrence and the detection of deadlock will be long.

- For both a timer and bus tracker, after detecting the deadlock, some complicated mechanism is required to resume the bus system to state before the deadlock, which may require large hardware overhead.

III. SOFT ERRORS

The unexpected changes in the value of a bit (or bits) inside a memory is called as soft errors [4]. Due to the sudden change in the data, it gets stored inside the memory as

if it was a valid data. Soft errors cause changes in the data rather than changes in the hardware. By restoring the error, the value of the system starts its operation. There are usually two types of soft errors such as, Single bit upset and Multiple cell upset. In single-bit upsets the error occur when only one bit of given data unit is changed from 1 to 0 or from 0 to 1. Multiple cell upset occurs when two or more bits in the data changes from 0 to 1 or vice-versa. Such type of errors does not mean that error occurs in adjacent bits.

For detecting and correcting errors, extra bits should be added to every data byte of the memory. The extra bits are called as parity check bits. Parity bits are the simplest form of error detecting codes. These parity bits are used to determine whether the byte data that is stored in the memory has the correct number of 0s and 1s in it. If the count changes, it indicates that there is an error. There are two types of parity bits such as, Even parity bit and Odd parity bit. In even parity bit, if the number of ones in the given data is odd then the parity bit is set as one. If the number of ones is even then, parity bit is set as zero. In odd parity bit, if the number of ones in the given data is even then the parity bit is set as one. If the number of ones is odd then the parity bit is set as zero.

In some cases these parity bit itself may be erroneous and thus the error cannot be detected. Due to this error correcting codes were used. It is an algorithm which gives a sequence of numbers, such that the errors present can be easily detected and corrected. Error-correcting codes are more complex than the error detecting codes. Moreover it requires more redundant bits. Hence, the number of bits required to correct multiple bit errors will be very high.

IV. BUS ARCHITECTURE FOR DEADLOCK FREE COMMUNICATION

The bus architecture can be developed based on AXI protocol. AXI stands for Advanced eXtensible Interface.

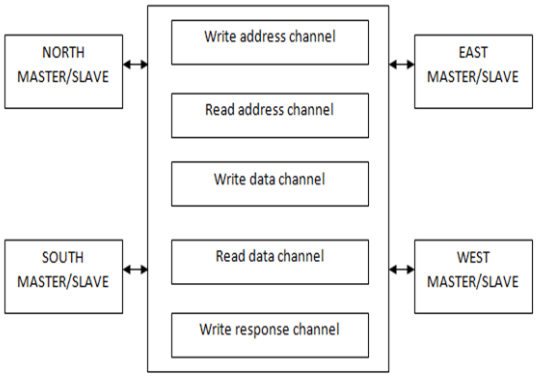

Fig. 2. Masters/Slaves in AXI Bus Architecture

From a technology perspective, AXI provides the means to perform low latency, high bandwidth on chip communication between multiple masters and multiple slaves. AXI Interconnect provides efficient connection between masters and slaves. The Interconnect is a highly configurable RTL component, which provides the entire infrastructure required to connect number of AXI masters to a number of AXI slaves. This infrastructure is an integral part of an AXI-based system. Architecture of the interconnect is highly modular with each of the routers and associated control logic partitioned on a per channel basis. It ensures, which bus master is allowed to initiate data transfers depending on highest priority or fair access.

Fig. 2 shows the basic block diagram of the bus architecture. The Master generates and drives transaction onto the bus, the Slave device accepts transaction from any master and the Interconnect routes the AXI requests and responses between masters and slaves. The interconnect consists of five channels; Read address channel, write address channel, read data channel, write data channel and write response channel. Every transaction has an address information and control information on the address channel that describes the nature of the data to be transferred. The data transfer takes place between master and slave using a write data channel to the slave or a read data channel to the master. In write transactions, all the data flows from the master to the slave and it has an additional write response channel to allow the slave to acknowledge the master about the completion of the write transaction. Then the architecture should allow, address information to be issued ahead of the actual data transfer, support for multiple outstanding transactions.

The masters/slaves are named as North, East, South and West should communicate each other through the interconnect. The read as well as write operations will takes place between these masters/slaves. The interconnect will provide efficient connection between the masters and slaves. Each one can act as either master or slave. The one who initiates the transaction will be the master and the other will be the slave. The master will give the request and the slave will respond according to it. A slave is a device or process that always responds according to the requirement of the master. The different channels included in the AXI protocol will be there in the interconnect. The address as well as the

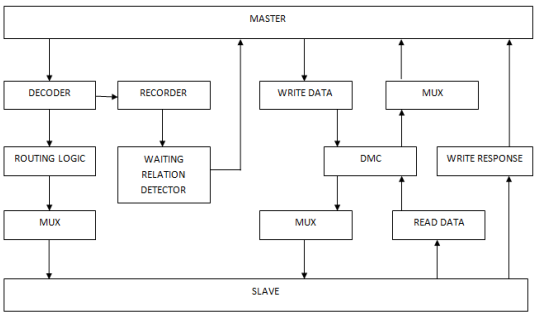

Fig. 3. Communication between Single Master and Single Slave

Control bits will be carried by the address channel. There will be separate channels for the read and write operation. The write response channel will give the acknowledgement for the operations.

The communication between a single master and slave will happen as shown in the Fig. 3. The data transfer between a master and slave will be completed through the different channels between the master and the slaves.

A. Read and Write Address Channels

These channels are used to read or write the address information from master to slave and vice versa. The address channel will contain the address of the corresponding slave to be communicated, the tagged ID of the transaction and the read or write information. The address will be given to the master and decoding is done to get address, tagged ID and read or write information. Then it checks whether any waiting relation exists. If any waiting relation exists, that transaction will be stalled. If there is no waiting relation, the corresponding slave is identified and it will be enabled for the data transfer.

Handling Out-of-Order Transactions: Out-of-order transactions allow the responses to be returned in an order different from request order. It is based on the tagged ID. The execution order depends on the IDs they are tagged. That is, all tagged transactions with the same tag ID must be executed

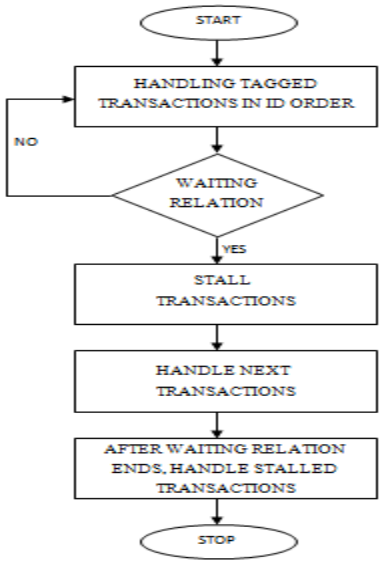

Fig. 4. Handling Out-of-Order Transactions

In order and two transactions tagged with different IDs can be executed out of order.

If the out-of-order transactions are not properly handled, the deadlock situation may get occurred. It should have to check whether any waiting relation happen in the bus while handling each transactions and in such cases, it have to be avoided. The waiting relation can be avoided by stalling such transactions. Such transactions can be stalled until the waiting relation ends. When those transactions are stalled, other transactions can be handled.

B. Read and Write Data Channels

The read data channel and the write data channel will carry the data to be transferred between the masters and slaves. After the selection of the slave to be communicated by the address channel, the data transfer will takes place. The read data channel conveys the read data from the slave back to the master. The write data channel conveys write data information from the master to the slave.

There will be buffers in the sending side and receiving side. While writing data, the data that is to be transferred will be initially stored into a buffer. Then it will be handled. In the receiving side, the output will be stored in another buffer and it will be obtained at the required slave. The same process will takes place while reading data also.
Here Decimal Matrix Code is used for the encoding and decoding purpose. DMC are mainly used to avoid soft errors. DMC make use of decimal algorithm that is decimal integer addition and decimal integer subtraction to detect errors. In this the input bits are provided to the DMC encoder during the encoding process. As a result, the horizontal redundant bits, vertical redundant bits and the information bits are obtained. Once encoding process gets completed, the obtained codeword is stored to the memory. If multiple cell upset occurs in memory information block, this can be corrected during the decoding process.

DMC Encoder: At first an N-bit data is divided into k symbols of m bits such that N=k*m. Then the symbols are arranged as k=k1*k2. Here k1 represents the number of rows and k2 represents the number of columns. The horizontal redundant bits (H) are calculated by using decimal integer addition of selected symbols per row and the vertical redundant bits (V) are calculated by binary operation of the bits per column. Here a 32 bit word is chosen and it is divided into 8 symbols of 4 bit, such that k1=2 and k2=4.

Here D0-D31 are the information bits, H0-H19 are the horizontal redundant bits and V0-V15 are the vertical redundant bits. The maximum correction capability and the number of redundant bits are different for different values of k and m. Hence k and m should be properly chosen to maximize the error correction capability and to reduce the number of redundant bits.

The horizontal bits are calculated by decimal integer addition as follows:

$$H4H3H2H1H0 = D3D2D1D0 + D11D10D9D8 \quad (1)$$

$$H9H8H7H6H5 = D7D6D5D4 + D15D14D13D12 \quad (2)$$

$$H14H13H12H11H10 = D19D18D17D16 + D27D26D25D24 \quad (3)$$

$$H19H18H17H16H15 = D23D22D21D20 + D31D30D29D28$$

(4)

Where + indicates decimal integer addition.
The vertical redundant bits are calculated as follows:

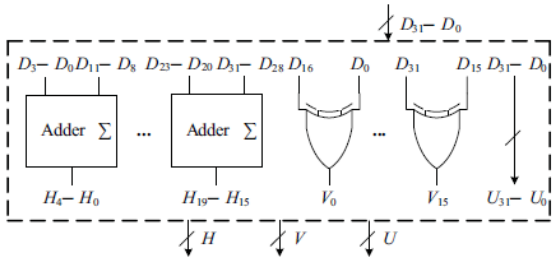

Fig. 5. DMC Encoder

Fig. 6. DMC decoder

V 0 = D0 xor D16 (5)
V 1 = D1 xor D17 (6)
V 2 = D2 xor D18 (7)
V 3 = D3 xor D19 (8)

Accordingly all the vertical redundant bits are calculated. Thus at the output of the encoder horizontal redundant bits, vertical redundant bits and the original bits (U0- U31) are obtained.

DMC Decoder: The decoding process takes place in two steps: Firstly the received redundant bits [H4H3H2H1H0] and [V0- V3] are calculated from the information bits obtained from the memory information block. Secondly the horizontal syndrome bits ΔH4H3H2H1H0 and vertical syndrome bits S3-S0 are calculated.

To calculate the horizontal syndrome bits, decimal integer subtraction is performed between the received horizontal redundant bits and the horizontal redundant bits obtained from memory redundant block and is as shown below:

ΔH4H3H2H1H0 = [H4H3H2H1H0]' - H4H3H2H1H0 (9)
ΔH9H8H7H6H5 = [H9H8H7H6H5]' - H9H8H7H6H5 (10)
ΔH14H13H12H11H10 = [H14H13H12H11H10]' - H14 H13H12H11H10 (11)
ΔH19H18H17H16H15 = [H19H18H17H16H15]' - H19H18H17H16H15 (12)

Where - indicates decimal integer subtraction. To calculate vertical syndrome bits, XOR operation is performed between the received vertical redundant bits and the vertical redundant bits obtained from memory redundant block and is as shown below:

S0 = V 0' xor V 0 (13)
S1 = V 1' xor V 1 (14)
S2 = V 2' xor V 2 (15)
S3 = V 3' xor V 3 (16)

Accordingly all the vertical syndrome bits, are calculated. Suppose if all the bits of ΔH4H3H2H1H0 and S3-S0 are zero then there is no error. This indicates that there is no error in symbol zero. If ΔH4H3H2H1H0 and S3-S0 is non-zero, it indicates that there is error in symbol 0 and this error can be corrected as follows:

D0correct = D0' xor S0 (17)
D1correct = D1' xor S1 (18)
D2correct = D2' xor S2 (19)
D3correct = D3' xor S3 (20)

C. Write Response Channels

The write response channel provides a way for the slave to respond to write transactions. When the data reaches the exact destination, the slave acknowledges the master that the data is received. After this the transaction gets completed.

V. EXPERIMENTAL RESULTS

The transactions happen between multiple Masters/Slaves such as NORTH, EAST, SOUTH and West. Read as well write operations are performed between the Masters/Slaves. The one who initiates the transaction will be the master. The address as well as tagged ID is transferred through the address channel and thus the corresponding slave can be identified. Then the data can be transferred though the data channel.

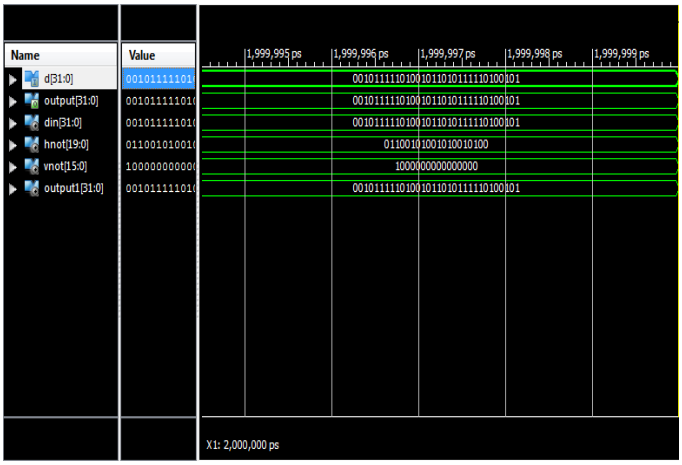

Fig. 7. DMC Output

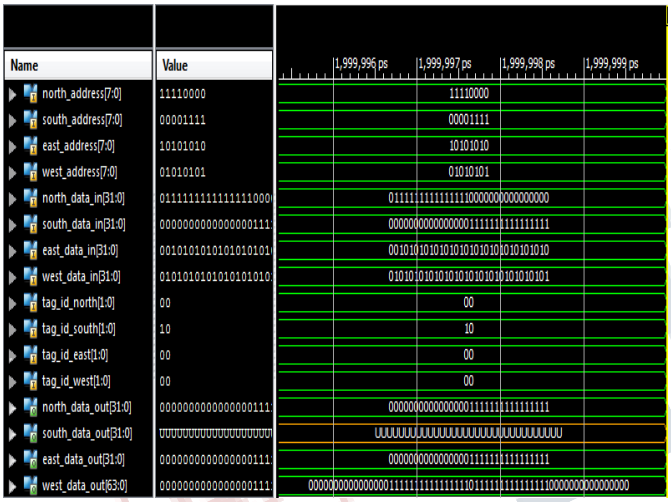

Fig. 8. Output Obtained without Deadlock Occurrence

The data bits are encoded and decoded using DMC to avoid unexpected or unusual errors. Fig. 7 shows the simulation result of the DMC.

Fig. 8 shows the transactions between multiple masters/slaves. The transactions happen based on the tagged ID. While handling out-order-transactions, the chances of occurring deadlock situation are avoided. This is done by checking the waiting relation between the transactions. If any waiting relation occurs, the transaction should be stalled and will go for handling next transactions. The stalled transactions can be handled after the waiting relation ends. Hence deadlock is avoided. VHDL language is used to develop the bus architecture and it is synthesized in Xilinx ISE Design Suite 13.2 and simulated in ModelSim SE 6.3f.

REFERENCES

[1] Advanced Microcontroller Bus Architecture Specification (1997) [Online]. Available: http://www.arm.com

[2] Chin-Yao Chang, Student Member, IEEE, and Kuen-Jong Lee,"On Deadlock Problem of On-Chip Buses Supporting Out-of-Order Transactions ", IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 22, no. 3, pp. 484-496, March 2014.

[3] Mayank Rai Nigam, Mrs Shivangi Bande, "AXI Interconnect Between Four Master and Four Slave Interfaces", Int. Jou. of Engg. Res. and Gen. Sci., vol. 2, issue 4, pp. 432-446, June-July 2014.

[4] Jing Guo, Liyi Xiao, Zhigang Mao, and Qiang Zhaoz," Enhanced Memory Reliability Against Multiple Cell Upsets Using Decimal Matrix Code", IEEE Trans. Very Large Scale Integr. (VLSI) Syst.,vol. 22, No. 1, January 2014.

[5] AXI Reference Guide, XILINX, UG761 (v13.4) January 18, 2012.

[6] Technical Reference Manual of PrimeCell AXI Configurable Interconnect (PL300), ARM, Cambridge, U.K., 2010.

[7] K. Lahiri, A. Raghunathan, and G. Lakshminarayana, "The LOTTERYBUS on-chip communication architecture", IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 14, no. 6, pp. 596608, Jun. 2006.

[8] T. S. Cummins, "Method and apparatus for detecting a bus deadlock in an electronic system", U.S. Patent 6 292 910, Sep. 18, 2001.