ISSN: 2456-043X

International Journal of Engineering Science and Generic Research (IJESAR)
Available Online at www.ijesar.in
Volume 2; Issue 5; September-October; 2016; Page No. 17-24

Design of Memory controller with AXI Bus interface

[1] K.KRISHNAIAH, [2] YELGAMONI RAVINDER

[1] M.Tech, Brilliant Institute of Engineering and Technology, Hyderabad, India

[2] Assistant Professor, Brilliant Institute of Engineering and Technology, Hyderabad, India

krishnaiah475@gmail.com

Abstract

The on-chip interconnection system known as advanced microcontroller bus architecture (AMBA) is a well-established open specification for the proper management of functional blocks comprising system-on chips (SOCs). An important feature for any of the SoC is based on how they interconnect. The AMBA AXI 4 protocol supports high performance, high-frequency system designs. It is suitable for high-bandwidth and low-latency designs and provides high frequency operation without using complex bridges. In the subject paper, the design and implementation details of AMBA bus (AXI) slave with memory controller (MC) interface are discussed. The AMBA-AXI does not require any bridge like AMBA-AHB. The realization of the control structure is based on the concept of conventional finite state machines (FSMs). The intellectual property (IP) blocks of AXI slave memory controller Designed in verilog and verified on Xilinx Isim simulator.

Keywords: *Advanced microcontroller bus architecture (AMBA) Advanced Extensible Interface (AXI), finite-state machines (FSMs), memory controller (MC), system-on-chips (SOCs).*

1. Introduction

Embedded real-time systems in particular and system-on-chips (SOCs) in general are typically comprised of various types of computational elements. In the realm of processing, these elements perform different tasks in order to realize an overall solution. Consider a set-top box for television (TV) sets as an example [1]. A set-top box must generate inputs for a particular television channel from the received digital satellite signal. The entire process incorporates several phases. The first is to split the incoming digital signal into its component video and audio data streams. The next phase is to convert the video data stream into its actual television signal. Also, the audio data stream has to be changed into audio signal for the TV set. Besides, an additional job to look after will be to handle the user input for changing the channel when the remote control will be pressed. All these operations have to be completed not only simultaneously but within a certain time frame. The cost of not accomplishing these within the stipulated time frame or deadline will be manifested in the form of blank screen or audible noises. This is obviously unacceptable and hence, it is necessary to always deliver the data within real-time deadlines. The needed computational elements here will be either general-purpose or special-purpose processors (such as signal processors). Nowadays, multitudes of these devices are integrated in the form of an SOC. A processor needs to interact with other processors, memories or input/output (I/O) devices to complete a task. Currently, bus systems are used to interconnect the intellectual property (IP) blocks. The current research in this field suggests that using instead network-on-chips (NOCs) to interconnect these IP blocks might allow more flexibility than buses [1]. However, to get NOCs acceptable to the industry is still an open question. A major problem in the present context is to deal with large amount of data storage. Usually, higher storage volumes are handled using on-chip dynamic memory. The manufacturing process for the memories is different from that for standard logic. The dynamic memories have high data rates and large storage capabilities. Notwithstanding, the dynamic memories lack compliance in accessing different locations efficiently at random and also necessitate periodic refreshing to keep the storage content intact. In order to allow efficient communication between the IP blocks and a shared memory requires the use of a memory controller

Corresponding author: K.KRISHNAIAH

(MC). The IP blocks, which communicate with the memory, are named hence forth as requestors. The MC arbitrates among the requestors and manages the memory accesses. The target of this paper is to design an MC that will provide guarantee in real-time for efficient communication between the IP blocks and memory. In the following, we first present an overview of the MC in general and AMBA system in particular.

2. RELATED WORK

2.1. Discussion on memory controller (MC) And Advanced microcontroller bus architecture(AMBA) system

The performance of microprocessors has been improving rapidly over the years. In contrast, the memory latencies and bandwidths have progressed relatively little. As a consequence, the memory access has been a real bottleneck in the context of improving the system performance. The MC should be efficiently combined with the interconnections. The MC is designed as a master that operates on different protocols and interacts with the memory by sending or receiving data with the help of its slave.
The biggest challenge in an SOC design is the integration of processors and memory modules. An increasing numbers of companies have adopted the advanced microcontroller bus architecture (AMBA) system, which has rapidly evolved as the de facto standard for the SOC interconnections and IP library development. The AMBA enhances a reusable design methodology by defining a common backbone for the SOC modules [4]. Figure 1 shows the AXI interconnect.

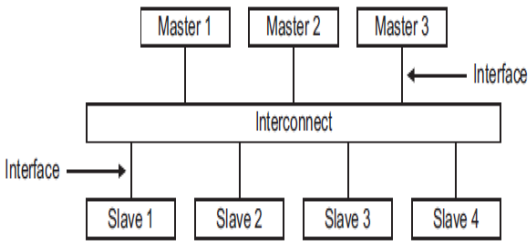

Figure 1: ***AXI interconnect with interfaces***

The first AMBA buses were Advanced System Bus (ASB) and Advanced Peripheral Bus (APB) [1&2]. In its second version, AMBA2, ARM added AMBA High-performance Bus (AHB) that is a single clock-edge protocol [3-5]. In 2003, ARM introduced the third generation, AMBA 3 [6&7], including AXI to reach even higher performance interconnects and the Advanced Trace Bus (ATB) as part of the Core Sight on chip debugs and trace solution. In 2010 the AMBA 4 specifications were introduced starting with AMBA 4 AXI4, then in 2011 extending system wide coherency with AMBA 4 ACE with a re-designed high-speed transport layer and features designed to reduce congestion [8-10]. Advanced microcontroller bus architecture (AMBA) protocol family provides metric-driven verification of protocol compliance, enabling comprehensive testing of interface intellectual property (IP) blocks and system-on chip (SoC) designs. The AMBA advanced extensible interface 4 (AXI4) update to AMBA AXI3 includes the following: support for burst lengths up to 256 beats, updated write response requirements, removal of locked transactions and AXI4 also includes information on the interoperability of components. AMBA AXI4 protocol system supports 16 masters and 16 slaves interfacing.

The AMBA-AXI does not require any bridge like AMBA-AHB. This AMBA-AXI compliant memory controller can be used with the latest ARM processors as well. This memory controller is mainly designed for interfacing the memories like SRAM and ROM and matches well with the memory latencies. Here the Memory controller is designed to improve the system performance. As the Memory Controller is On-chip, the memory accessing time is decreased so that performance is increased.

In the following section, AXI with application of MC is shown; also, the efficiency in terms of area overhead and speed is discussed. The control structure is designed with conventional finite state machines (FSMs). The IP of an AXI slave interface is realized and its interface with the MC is designed and tested.

The Advanced Microcontroller Bus Architecture (AMBA) is a protocol that is used as an open standard; on-chip interconnects specification for the connection and management of functional blocks in a system-on-chip (SoC). The AMBA bus is applied easily to small scale SoCs. Therefore, the AMBA bus has been the representative of the SOC market though the bus efficiency.

Three distinct buses are defined within the AMBA specification:
1. Advanced Peripheral Bus (APB).
2. Advanced System Bus (ASB).
3. Advanced High performance Bus (AHB).

© 2016 IJESAR. All Rights Reserved.

4. Advanced extensible Interface Bus (AXI).

The AMBA specification defines all the signals, transfer modes, structural configuration, and other bus protocol details for the APB, ASB,AHB, and AXI buses.

The AMBA APB is used for interface to any peripherals which are low bandwidth and do not require the high performance of a pipelined bus interface. APB peripherals can be integrated easily into any design flow, with the following specification:

- Peripheral bus for low-speed devices
- Synchronous, non multiplexed bus
- Single master (bridge)
- 8, 16, 32-bit data bus
- 32-bit address bus
- Non-pipelined

The AMBA ASB sits above the current APB and implements the features required for high-performance, with the following specification:

- burst transfers
- pipelined transfer operation
- multiple bus master.

AMBA AHB is a new level of bus which sits above the APB and implements the features required for high performance, high clock frequency systems, with the following specification:

- Burst transfers
- Split transactions
- Single cycle bus master handover
- Single clock edge operation
- Wider data bus configurations (64/128 bits)

AXI extends the AHB bus with advanced features to support the next generation of high performance SoC designs. The goals of the AXI bus protocol include supporting high frequency operation without using complex bridges, flexibility in meeting the interface, and performance requirements of a diverse set of components, and backward compatibility with AMBA AHB and APB interfaces. The features of the AXI protocol are:

- Separate address/control and data phases
- Support for unaligned data transfers
- Ability to issue multiple outstanding addresses
- Out-of-order transaction completion.

3. ARCHITECTURE OF AMBA AXI MEMORY CONTROLLER (MC)

The AMBA-AXI memory controller is mainly divided into three parts like: AXI Slave interface, FIFO, External Memory Interface. Figure 2 shows the architecture of AXI-MC.

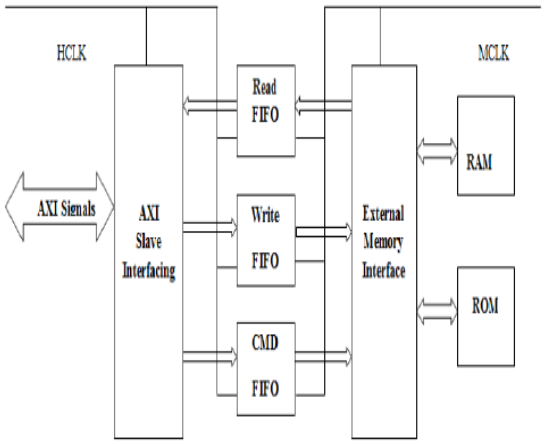

Figure 2: ***Architecture of AXI- MC***

3.1. AXI Slave Interface

AMBA AXI supports data transfers up to 256 beats and unaligned data transfers using byte strobes. In AMBA AXI system 16 masters and 16 slaves are interfaced. Each master and slave has their own 4 bit ID tags. AMBA AXI system consists of master, slave and bus.

The system consists of five channels namely write address channel, write data channel, read data channel, read address channel, and write response channel.

The AXI protocol supports the following mechanisms:

- Unaligned data transfers and up-dated write response requirements.
- Variable-length bursts, from 1 to 16 data transfers per burst.
- A burst with a transfer size of 8, 16, 32, 64, 128, 256, 512 or 1024 bits wide is supported.

Each transaction is burst-based which has address and control information on the address channel that describes the nature of the data to be transferred. The data is transferred between master and slave using a write data channel to the slave or a read data channel to the master. Table 1[3] gives the information of signals used in the complete design of the protocol. The write operation process starts when the master sends an address and control information on the write address channel as shown in fig 1. The master then sends each item of write data over the write data channel. The master keeps the VALID signal low until the write data is available.

© 2016 IJESAR. All Rights Reserved.

The master sends the last data item, the WLAST signal goes HIGH. When the slave has accepted all the data items, it drives a write response signal BRESP [1:0] back to the master to indicate that the write transaction is complete. This signal indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR. After the read address appears on the address bus, the data transfer occurs on the read data channel. The slave keeps the VALID signal LOW until the read data is available. For the final data transfer of the burst, the slave asserts the RLAST signal to show that the last data item is being transferred. The RRESP[1:0] signal indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR. The protocol supports 16 outstanding transactions, so each read and write transactions have ARID[3:0] and AWID [3:0] tags respectively. Once the read and write operation gets completed the module produces a RID[3:0] and BID[3:0] tags. If both the ID tags match, it indicates that the module has responded to right operation of ID tags. ID tags are needed for any operation because for each transaction concatenated input values are passed to module

The block diagram of the AXI Generic Mater Controller has the following blocks and is shown in the fig 3:

1) Master
2) AMBA AXI4 Interconnect
3) Slave

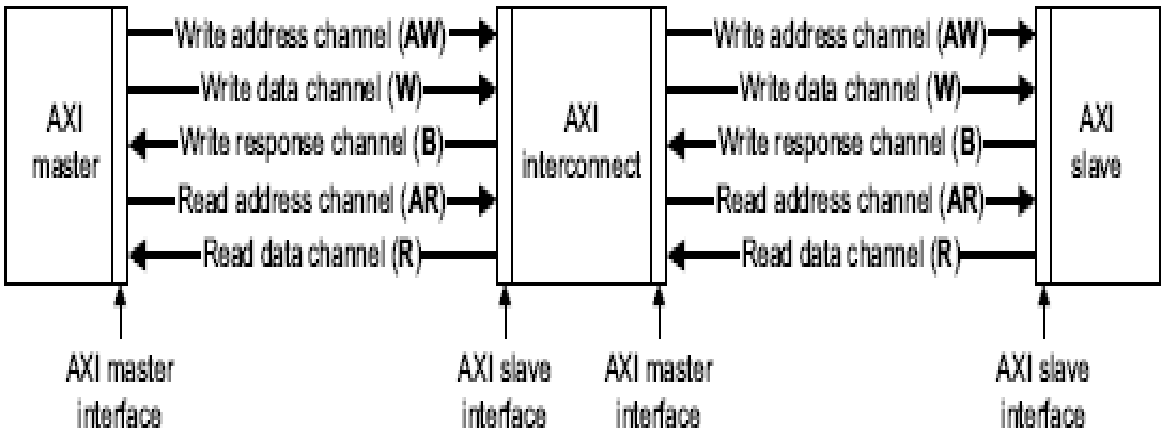

Figure 3: The Block Diagram of AXI Generic Master

Table 1: Signal descriptions of AMBA AXI protocol.

Signal	Source	Input/ Output	Description
Aclk	Global	Input	Global Clock Signal
Aresetn	Global	Input	Global Reset Signal
AWID[3:0]	Master	Input	Write address ID
AWADDR[31:0]	Master	Input	Write address
AWLEN[3:0]	Master	Input	Write burst length
AWSIZE[2:0]	Master	Input	Write burst size
AWBURST[1:0]	Master	Input	Write burst type
AWLOCK[1:0]	Master	Input	Write lock type
AWCACHE[1:0]	Master	Input	Write cache type
AWPROT[2:0]	Master	Input	Write protection
WDATA[31:0]	Master	Input	Write data
ARID[3:0]	Master	Input	Read address ID
ARADDR[31:0]	Master	Input	Read address

© 2016 IJESAR. All Rights Reserved.

ARLEN[3:0]	Master	Input	Read burst length
ARSIZE[2:0]	Master	Input	Read burst size
ARLOCK[1:0]	Master	Input	Read lock type
ARCACHE[3:0]	Master	Input	Read cache type
ARPROT[2:0]	Master	Input	Read protection
RDATA[31:0]	Master	Input	Read data
WLAST	Master	Input	Write last
RLAST	Slave	Output	Read last
AWVALID	Master	Output	Write address valid
AWREADY	Slave	Output	Write address ready
WVALID	Master	Output	Write valid
RVALID	Slave	Output	Read valid
WREADY	Slave	Output	Write ready
BID[3:0]	Slave	Output	Write response ID
RID[3:0]	Slave	Output	Read response ID
BRESP[1:0]	Slave	Output	Write response
RRESP[1:0]	Slave	Output	Read response
BVALID	Slave	Output	Write Response valid
BREADY	Master	Output	Response Ready
RVALID	Slave	Output	Read valid

3.2 External Memory Interface

Figure 4 describes the state diagram of the MC. It is also an FSM implementation; the initial condition is reset state which is an idle state when no operation is there. When start signals arrive, the FSM triggers; depending upon the instruction, its operation is decided on by command (CMD) control state. Based on the instruction, it moves to RAM read, RAM write and ROM read operations. Two important tasks for the MC are to handle all the scheduling among the different commands and to keep track of which rows are presently activated. The activation of rows is time-consuming and therefore, the MC has a look-ahead functionality where the arbitrator can report which command is going to be executed after the current one has been completed. The arbitrator makes it possible to activate the row in advance if the next command is not retrieving the same bank or chip as the current command.

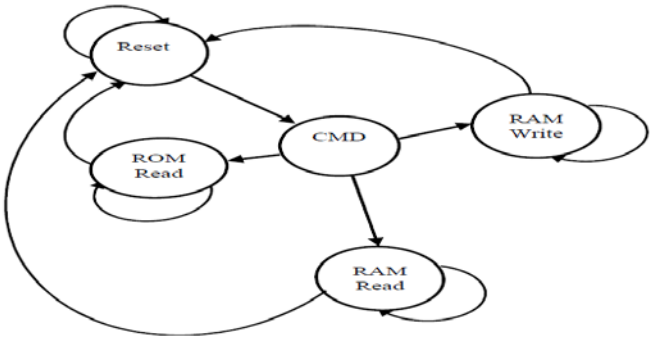

Figure 4: ***State diagram of memory controller (MC).***

3.3 First-in first-out (FIFO)

The FIFO is a method of processing and retrieving data. In a FIFO system, the first items entered are the first ones to be removed. In other words, the items are removed in the order they are entered. Figure 4 shows a FIFO buffer consisting of two modules (called source and sink) connected to one another. When data are being passed from module to module, the source is the module that is outputting data. The sink is the module that is receiving that data. In Figure 6, three signals are shown between the two modules A

© 2016 IJESAR. All Rights Reserved.

and B: data, FIFO full and FIFO empty. The line marked data represents the wire that actually passes the data from the source to sink. The FIFO full and FIFO empty are known as handshaking signals, which allow the source and sink to communicate when it is time to pass the data. The FIFO full signal indicates that the FIFO is full and puts valid data on the data line. FIFO full is what is called a state signal – it is high only when data are valid. If data are not valid on the data line during a particular cycle, valid should be low during that cycle.

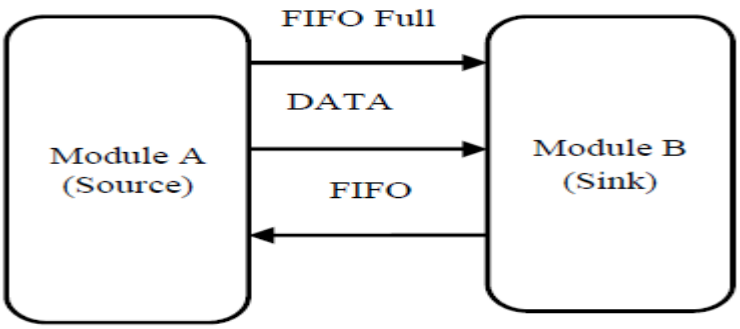

Figure 5: ***First-in first-out***

4. SIMULATION EXPERIENCE

The simulation results of the AXI-MC operations: signals high or low, depending on the read or write control signals are shown in Figure 5. If a write operation is to be executed, then, the FIFO is empty (default) and once the entire data are written, the FIFO turns to full. The MC outputs depend upon CMD control. There are three outputs: RAM_RD, RAM_WR and ROM_RD. Furthermore, the read or write operations are forwarded to the MC by the slave interface.

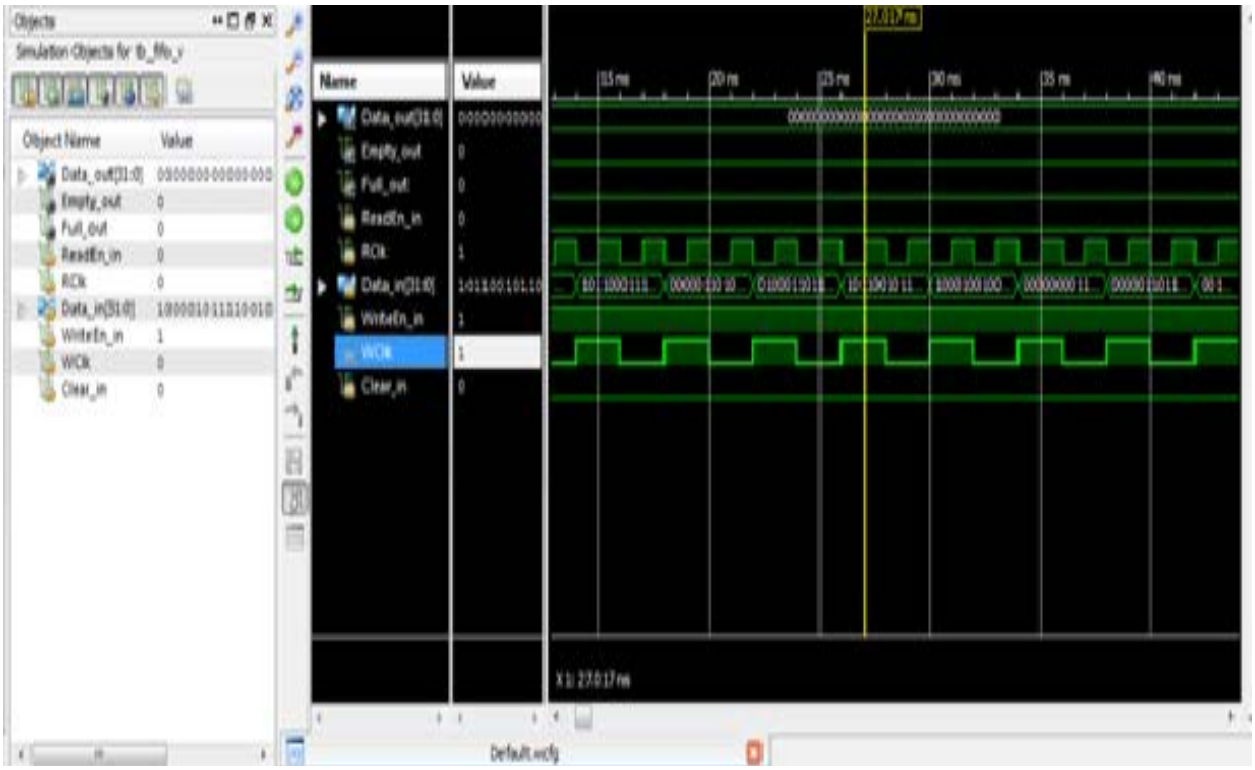

Figure 6: Simulation result of first-in first-out (FIFO)

© 2016 IJESAR. All Rights Reserved.

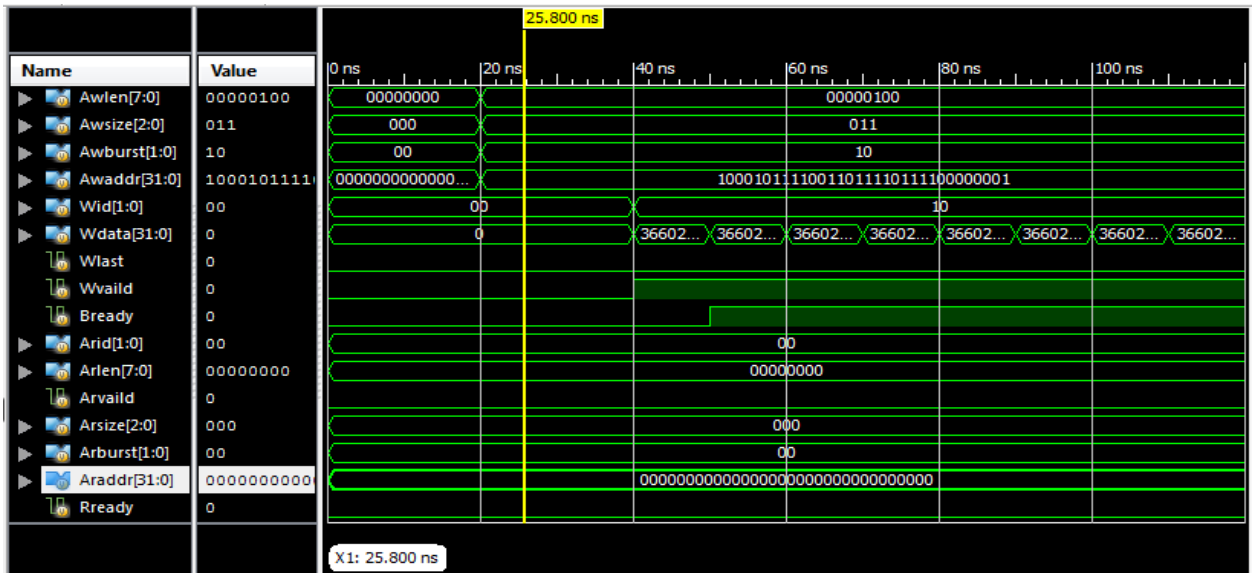

Figure 7: ***AXI MC write operation***

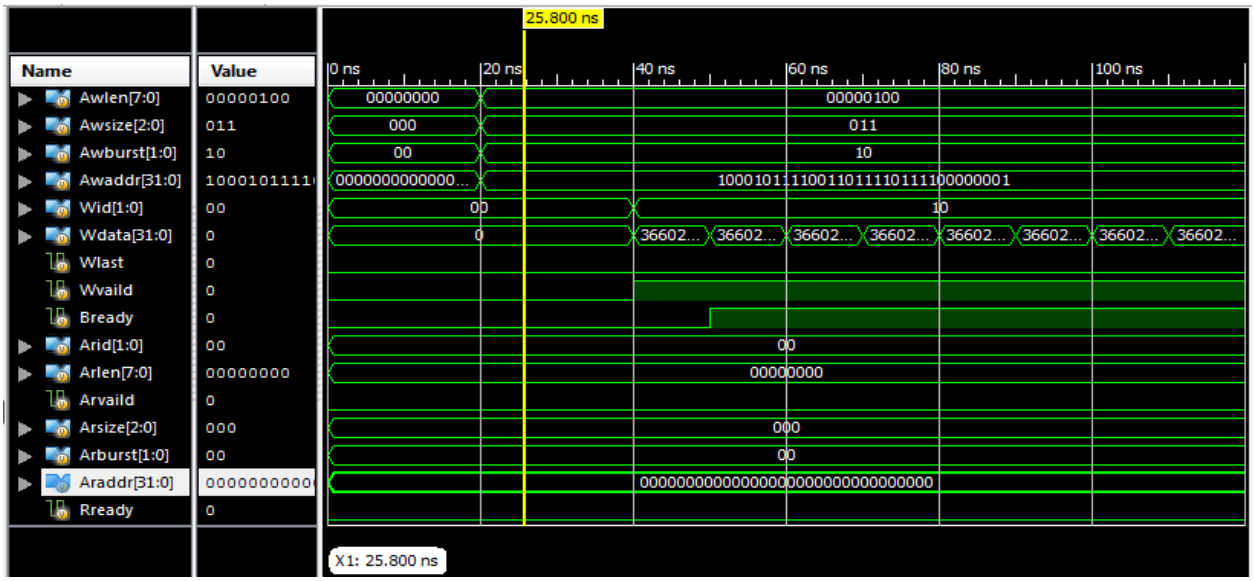

Figure 8: *AXI-MC read operation*

5. CONCLUSIONS

An AXI-based MC is designed, realized and tested in the present paper. The implementation was carried out using Verilog HDL for SOC solutions. The design improves the performance and it removes the bridges like in AHB bus architecture. To avoid handshaking complexity, we have used the FIFO for the MC and slave interface. Even though this increases the latency, it makes design simple and free of bottleneck. The design has been synthesized on Xilinx13.2 Spartan3E and simulated in ModelSim.

6. REFERENCES

1. K. Goossens, O. P. Gangwal, J. Rover, and A. P. Niranjan, “Interconnect and memory organization in SOCs for advanced set-top boxes and TV – evolution, analysis, and trends”, in J. Nurmi, H. Tenhunen, J. Isoaho, and A. Jantsch, Editors, Interconnect-Centric Design for Advanced SoC and NoC, Chapter 15, pp. 399–423, Kluwer, 2004.
2. Y. Hu and B. Yang, “Building an AMBA AHB compliant memory controller”, Proceedings of the Third International Conference on Measuring

© 2016 IJESAR. All Rights Reserved.

Technology and Mechatronics Automation , Vol. 01, 2011, pp. 658–661.

3. S. Rao and A. S. Phadke, "Implementation of AMBA compliant memory controller on a FPGA", International Journal of Emerging Trends in Electrical and Electronics, Vol. 2, April 2013, pp. 20–23.
4. AMBA Specification (Rev 2.0), ARM Inc., 1999.
5. AHB Example – AMBA System, Technical Reference Manual, ARM Inc., 1999.
6. Arm amba 2 speci_cation. This research was supported in part by the Department of Computer Science, College of Arts and Sciences, Troy University, Montgomery, AL 36103, USA.
7. AMBA® AXI Protocol, v1.0 Specification, ARM IHI 0022B, 2003

AUTHORS DETAILS:

	**K.KRISHNAIAH** has completed his B.Tech in Electronics and Communication Engineering from Sree Dattha Institute of Engineering & Science, J.N.T.U.H Affiliated College in 2013. He is pursuing his M.Tech in VLSI System Design from Brilliant Institute of Engineering and Technology, J.N.T.U.H Affiliated College.
	**YELGAMONI RAVINDER** an Assistant Professor at Brilliant Institute of Engineering and Technology, Hyderabad in ECE Department. He received his B.Tech degree in Electronics and Communication Engineering from Visvesvaraya institute of Engineering & Technology, J.N.T.U.H Affiliated College in 2006. He has completed M.Tech in VLSI System Design from New Nethaji institute of Engineering & Technology J.N.T.U.H Affiliated College. His research interest is VLSI Technology and Design and Communication Systems.

© 2016 IJESAR. All Rights Reserved.