Design of Bit Move Block using Bandwidth Arbitration for Master Slave Communication

Saranya Vasudevan and Lili He, *Member, IAENG*

***Abstract*— This paper illustrates the AMBA AXI4-Lite protocol bus model for multiple Master-Slave communication using bandwidth arbitration. A bit movement block is designed to perform read, modify, and write data into memory locations. Each bit move block consists of five configuration slave registers. The slave registers store source address, a destination address, and, a start bit. Bit move operation begins with start bit acknowledgment and the operation terminates with the done signal. Eight instances of this block act as masters, memory, and a total of 9 devices act as slaves. Master initiates the transfer by providing slave address, and slave acknowledges accordingly. Bandwidth allocation arbitrator is designed to select the master permitted to use the bus. An AXI-4 lite fabric is designed to establish connections between all these devices. The project aims to develop the best performing design which completes bit-move operation within a limited number of cycles. The project is designed in System Verilog.**

***Index Terms*— Bit movement, System Verilog, Arbitration, AMBA AXI4-Lite**

I. Introduction

The technological advancement in the semiconductor industry brought a plethora of devices on the same die without much increase in the size of the die. Most of the industries have incorporated SOC design as it includes more components to be embedded in a single die. System on a chip (SOC) is an integrated circuit that consists of all essential computer components like Central Processing Unit (CPU), Universal Asynchronous Receiver Transmitter (UART), Random Access Memory (RAM), Read-Only Memory (ROM), and various other peripherals [1]. With an increase in the number of components on a single chip performance degrades due to lack of proper interconnection, so proper connections between these components are essential for effective functionality. Advanced Micro Controller Bus Architecture (AMBA) introduced by ARM establishes the standards for proper communication between devices in a system. Advanced Micro Controller Bus Architecture (AMBA) is an open-source standard that provides the protocol for interconnecting several components in a chip. The AMBA protocol is popular and widely adopted in the semiconductor industry. The Advanced Micro Controller Bus Architecture (AMBA) has resolved problems in Soc by providing coherence in design, reusability of designs and it is extensively used for a design with high performance [2].

Manuscript received March 23, 2021, revised April 10, 2021. The authors are with San Jose State University, Department of Electrical Engineering, San Jose, CA, USA. (Corresponding author phone 408-924-4073; fax: 408-924-3925; e-mail: lili.he@sjsu.edu).

A bit movement block module has been designed and implemented using AMBA AXI Lite protocol. The bit move module performs read, modify, and write data into memory locations. Each bit move block consists of five configuration slave registers. The slave registers store source address, a destination address, and, a start bit. The bit move operation begins with start bit acknowledgment and the operation terminates with the done signal. Eight instances of this block act as masters, memory, and a total of 9 devices act as slaves. The master initiates the transfer by providing the slave address, and the slave acknowledges accordingly. Only one master at a time is permitted to use the bus. Initially, every master gains the bus using arbitration to talk to the slave, later based on the bandwidth master gains the arbitration. An AXI-4 lite fabric is designed to establish connections between all these devices. In this project, the bit movement block with AMBA AXI4-Lite model is designed in System Verilog.

II. The Proposed Bit Movement Block Design and Implementation

A. AMBA AXI-4 Lite bus protocol

Advance extensible Interface (AXI) protocol aims to give high performance in designs. The protocol provides great flexibility in interconnections. Advance extensible Interface (AXI) protocol operates in burst mode, has split transactions [3]. A Single bus is everything and works as five individual sub buses. Figure 1 depicts the block diagram of the AXI protocol design. Different portions of the Advance extensible Interface (AXI) are used for various operations. The operations are as below.

- Write address bus for sending address from master to slave.
- Write data bus to send data.
- Write response bus for giving back response to write from the slave.
- Read address bus to initiates the read.
- Read data bus to send data from slave to master.

The basic concept of an Advance extensible Interface (AXI) bus is to gain performance by splitting a single bus into five buses, thus allowing reads and writes overlaps [4].

ISBN: 978-988-14049-2-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

The Advance extensible Interface AXI4-lite is a subset of Advance extensible Interface. Advance extensible Interface (AXI) is commonly used for lite which has no burst capability and no split transactions which is much better electrically. AXI4-lite interface is simpler when compared to the complete AXI4 interface. The communication between the AXI master and the AXI slave happens through any of the five communication channels. The five channels are the write address channel, write data channel, write response channel, read address channel, and the read data channel [5].

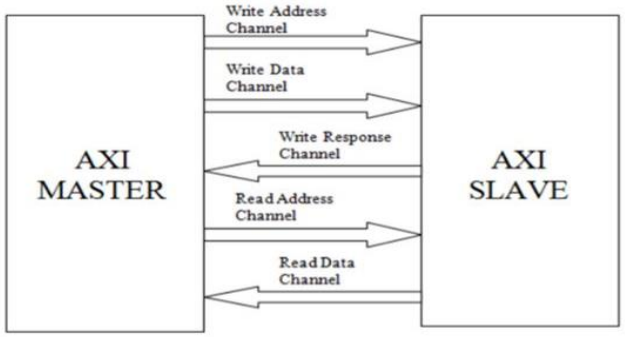

Fig.1. Block diagram for AXI master-slave.

B. Design of bit move block

The bit move module moves data from one part of the memory to another part of the same memory. Figure 2 depicts the block diagram of the bit-move module with slave and master interface signals. To perform the bit-move operation the bit move module uses a state machine. The state machine needs information like source address, a destination address, block size, and the condition to start the bit move operation [6]. The data is read from the source address and written into the destination address. The information about the source address, a destination address, block size, and the start bit are coded into the configuration registers.

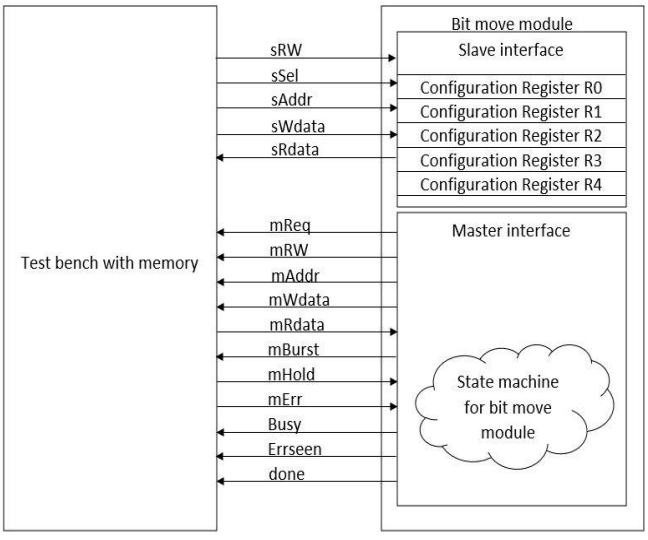

Figure.2. Bit-move module with slave and master.

The slave interface signals begin with the prefix‘s’ and the master interface signals begin with the prefix ‘m’. The test bench will program the configuration registers using the slave interface signals. Table I provides a list of slave interface signals. R0 and R1 provide the source address to read the data from. R2 and R3 registers provide the destination address to move the data to the required destination. R1 register provides the size of data to move several bits. The test bench writes ‘1’ to Start bit of R4 which initiates the bit movement operation. The bit-move module starts the bit-move operation when the start bit is asserted in the R4 configuration register. The bit-move module then begins to move the data inside the memory.

TABLE I
List of slave interface signals

Number	Signal Name	Function
1	sSel	The test bench has enabled the slave interface to perform a read or write to the registers.
2	sRW	The test bench mentions if it wants to read or write to the registers. 0 indicated read and 1 indicates write.
3	sAddr	This address mentions which register to read from or write into. It has the following registers written by test bench through the slave interface. R0: Source Address. R1: Portion of source address and block size. R2: Portion of destination address. R3: Portion of destination address. R4: Error, Busy, and Start bit.
4	sWdata	The test bench sends the data to write to the selected register through sWdata.
5	sRdata	The bit-move module reads the registers selected by the test bench using sRdata.

The slave address is controlled only by the test bench and the Bit-move module can only respond to the slave interface. The master interface is controlled only by the bit-move module. Test bench can only receive/respond to the master interface transfers by bit-move module. The test bench can never initiate a transfer on the master interface. Table II provides a list of master interface signals.

After the data has been moved, the module sends the moved data and their respective addresses to the testbench through the master interface signals. The slave interface signals are single cycle operations. Therefore, the slave interface signals cannot have any pipeline or additional delay. The signals mReq, mRW, mAddr, mWdata, mRdata, mBurst, mHold, mErr, Busy, ErrSeen, Done are the master interface signals. The master interface signals are two-cycle operations and support pipeline. The bit-move module asserts the mReq, mRW, mAddr signals in the first cycle,

ISBN: 978-988-14049-2-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

and the data is sent in the following cycle. The bit-move operation is performed with 32- bit blocks of data. The source and destination addresses programmed in the configuration registers are bit addresses. The bit addresses point to the respective starting bit and need not point at the 32-bit word border.

TABLE II
List of master interface signals

Number	Signal Name	Function
1	mReq	The output from bit-move module would like to initiate an operation to the test bench.
2	mRW	The output from the bit-move module to the test bench indicating bit-move module would like to perform a read or write.
3	mAddr	The output from the bit-move module to the test bench denoting the word address of the data the module writes to the test bench.
4	mWdata	The output from bit-move module to test bench denoting the data it sends to the test bench.
5	mRdata	The input to the bit-move module from the test bench denotes the read data from the test bench sends to the module.
6	mBurst	The output from the bit-move module to the test bench indicating bit-move sends bursts of data
7	mHold	The input to the bit-move module from the test bench indicating test bench is not ready to accept transaction request from the bit-move module.
8	mErr	The input to the bi-move module from the test bench that it has detected an error in the data sent by the bit-move module.
9	Busy	The output from the bit-move module to the test bench indicating bit-move module is busy with the moving operation.
10	Errseen	The output from the bit-move module to the test bench indicating an error has been encountered.
11	done	The output from the bit-move module to the test bench indicating the bit-move module has completed the bit-move operation.

The pointer can be anywhere in the 32-bit word address. Hence, the bit-move module performs the Read-Modify-Write while moving data to the destination. The read happens on the source and the Read-Modify-Write operation happens on the destination addresses. Read-Modify-Write is necessary to complete bit move operation in limited number of cycles.

C. Finite state machine for bit movement operation.

The finite state machine (FSM) is developed to decide the bit movement based on certain conditions [7]. The conditions are coded into the states namely reset, read the source, read destination, write destination address, write destination address and read the source, write destination data and read the source, and write destination data states. Table III depicts the finite state machine for bit movement operation.

TABLE III
States in the finite state machine.

S.No	Current state	Next state
1	Reset	Read source if start is true.
2	Reset	Read source if start is false.
3	Read source	Read destination is hold is false, enough data available in destination, source and destination address match.
4	Read source	Write destination address if hold is false and enough data available in destination.
5	Read source	Read source if enough data not available in destination or hold is true.
6	Read destination	Write destination address if hold is false.
7	Read destination	Read destination if hold is true.
8	Write destination address	Write destination data if source read is done.
9	Write destination address	Write destination address if source read is not done.
10	Write destination data	Read destination is hold is false and destination address matches.
11	Write destination data	Write destination data if hold is true.

The reset state decides if the bit move engine will be in the reset state or move to the read source state based on the start signal. When the start signal is asserted, the bit move module moves out of the reset state and enters into the read source state. When the hold is false and if enough data is available for destination in the read source state the bit move engine progresses to the write destination

ISBN: 978-988-14049-2-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

address state. When enough data is not available for the destination in the read source state, the bit move engine retains in the read source state. When the hold is true in the read source state, the bit-move engine retains in the read source state. When the hold is false in the read destination state, the bit move engine progresses to the write destination address state. When the hold is true in the read destination state, the bit move engine retains in the write destination address state. When the hold is false and source read is done in the write destination address and read source state the state machine progresses to the read destination state. When the hold is false and source read is not done in the write destination address and read source state the state machine progresses to the write destination address state. When the hold is true in the write destination address and the read source state, the bit move engine retains in the write destination address and read source state. When the hold is false in the write destination state and all data has been read, the bit move progresses to the read destination state else bit move progresses to the reset state. When the hold is false in the write destination state bit move engine progresses to the write destination data state.

D. Design of Bandwidth arbitrator

An arbiter decides whose turn it is to use a resource such as a bus. Various types of arbitrators include bandwidth arbitrator, round-robin arbitrator, priority arbitrator, priority round robin arbitrator, etc.

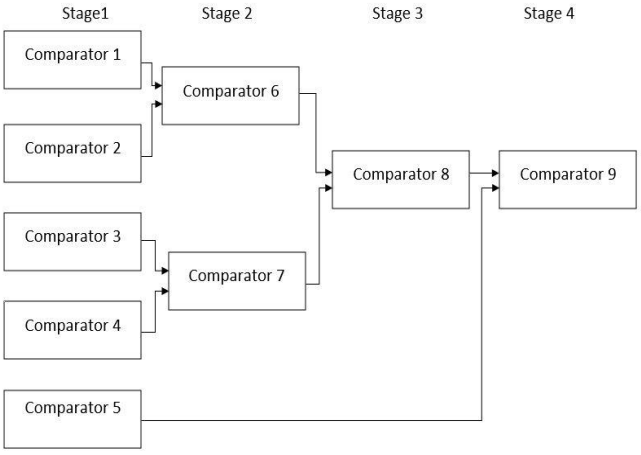

Fig.3. Bandwidth arbitrator implementation.

In the current project, a bandwidth arbitrator is designed to choose the bus master. Each master requesting bus access has a certain amount of bandwidth. A bandwidth allocation arbitrator selects the master having the highest bandwidth and allows that master to have bus access. A bandwidth allocation arbitrator is designed to identify the master that has access to the AXI-4 Lite bus. Figure 3 depicts the implementation of the bandwidth arbitrator.

The bandwidth arbiter needs information about the allocation, allocation amount, maximum amount, and burst size to arbitrate to the correct master. The request amount Bandwidth allocation arbitrator is used to select the master having the maximum bandwidth. Comparators are used to implement the bandwidth arbiter in four stages. In each stage, multiple comparisons are made to select the winner from each stage. Each comparator compares any two master's bandwidth amounts to identify the winner at each stage. The overall winner of the arbitration is determined in the fourth or the final stage.

E. Interface the Bit-move Module with AXI4-Lite Bus.

The final stage of the project is to implement eight bit-move modules with the bandwidth arbitrator along with the AXI4-Lite fabric. The test bench is also one of the masters connected to the fabric. The arbitrator arbitrates between the masters and provides read or write access to the winning masters. Anyone of the eight masters can request the bus independently or at the same time. When multiple masters request bus access at the same time, the bandwidth allocation arbitrator decides which master gets access to use the bus. When multiple masters requesting the bus, access have the same bandwidth amount, the bandwidth allocation arbitrator designed will select the master that has not used the bus recently. For instance, consider both bit-move module 0 and bit-move module 1 have the same bandwidth amount. If bit-move module 0 has the bus access recently, the current bus access is given to bit-move module 1.

The final stage of the project is to implement eight bit-move modules with the bandwidth arbitrator along with the AXI4-Lite fabric. Figure 4 depicts the block diagram for interfacing eight bit-move modules with the AXI4-Lite bus. The test bench is also one of the masters connected to the fabric. The arbitrator arbitrates between the masters and provides read or write access to the winning masters. The performance of the AXI4-Lite bus in this stage is analyzed and reported.

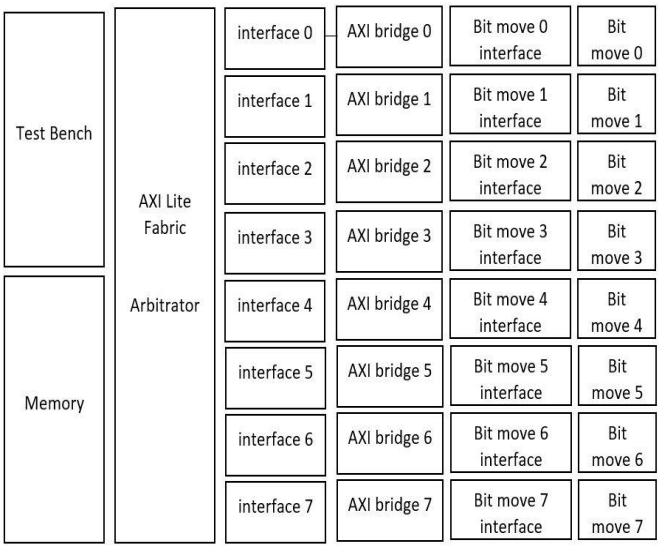

Fig.4. Block diagram for eight bit-move modules and test bench.

F. Simulation and Results

The bit-move module starts the bit move operation as per the state machine. The source is read once or twice based on the address. The first read is source read, then destination

ISBN: 978-988-14049-2-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

read based on address, source read, destination writes, source read, destination writes and operation continues until the entire bit movement operation is complete. Figures 5, 6, and 7 depict the simulation results with the AXI4 Lite bus for the initial bit move operation. The waveforms are shown for the first few cycles of the grant. Initially, the bit move module does not have the grant. The fabric has not yet issued the grant to this bit move module. The bit move master module has not won the arbitration yet. The bit-move module sends its request, mReq = 1, mRw = 0 to read. The ARREADY signal is still zero indicating the grant has not yet been provided to the bit move module. Until the mHold signal is one, the bit move module keeps holing its request. The bit move can begin the operation only when the mHold signal is zero.

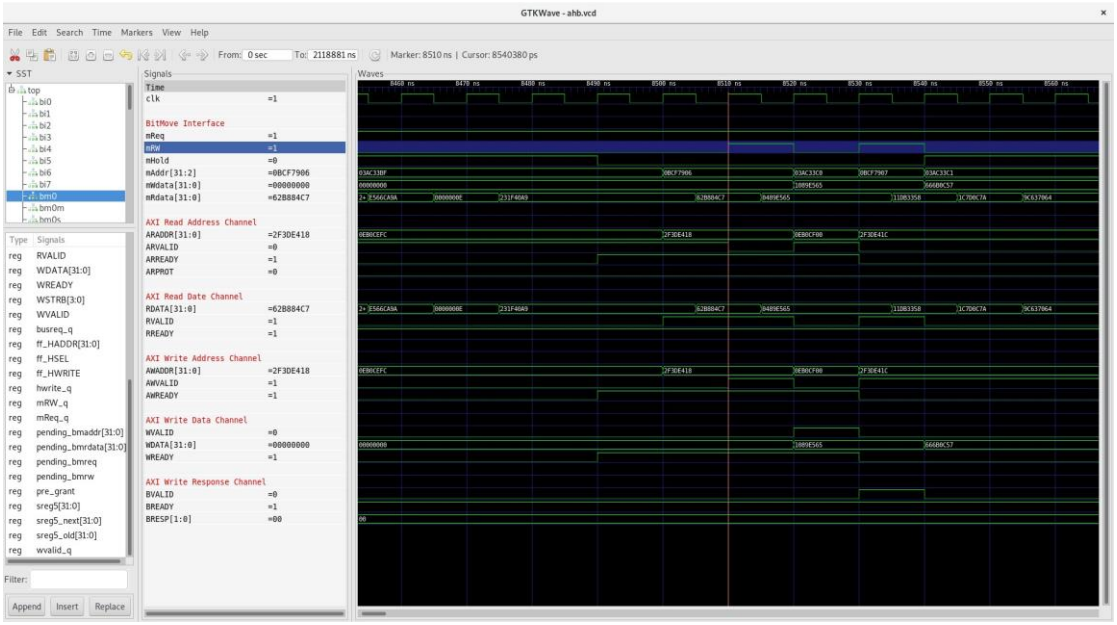

Fig.5. Simulation results with AXI4 Lite bus for the initial bit move operation.

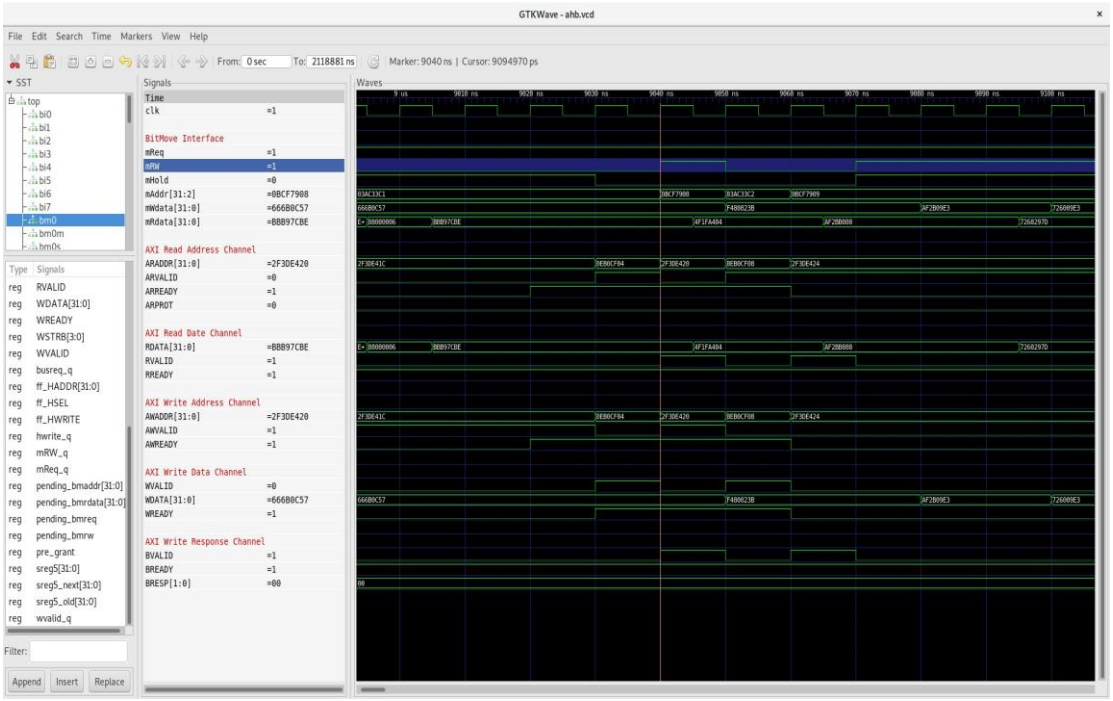

Fig.6. Simulation results with AXI4 Lite bus for the intermediate bit move operation.

ISBN: 978-988-14049-2-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

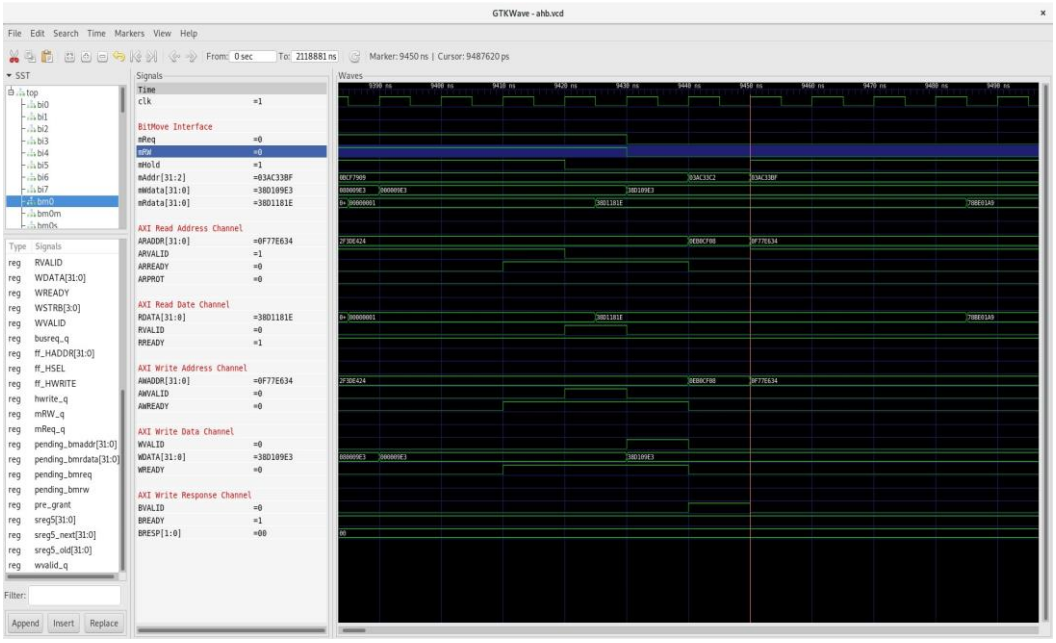

Fig.7. Simulation results with AXI4 Lite bus for the final bit move operation.

The working of signals in the AXI read address channel is explained below. In the next cycle, the destination read happens. In the destination read state, ARVALID is one and ARREADY is one, meaning destination read address request has been sent as one. The first read data is source read data and the second read data is the destination.

The address from the bit move is a thirty-bit word address. The AXI address is a 32-bit address. Therefore, two zeros are appended in the LSB of the AXI address to change the word address into a bit address. This is the reason for the difference in the value between mAddr and ARADDR shown in the waveform. However, the addresses are the same as zeros are appended. When bit move is requesting a read, ARVALID is set as one. ARVALID is not performing write and AWVALID is zero. Since no read data is available, RVALID is also zero. Once a grant is obtained, the grant is used as ARREADY and ARREADY becomes 1. When ARREADY is available, mHold becomes 0, which means the first source read address has reached the memory. ARREADY is available and ARVALID is available. So AXI transfer has happened successfully.

III. Conclusion

The AMBA AXI4-Lite specification by AMBA provides the architecture for interconnects in ASICS. The project implements the design of the bit movement module, designing bandwidth arbitrator, designing AMBA AXI4-Lite fabric using System Verilog. The master-slave interaction using bandwidth arbitration happens in the bit-move block. The project designed a bit-move module master which initiates the transfer by providing a slave address, and slave acknowledge accordingly. In any instance, only one master has permission to access the bus. The AXI4-Lite fabric establishes the connections between the devices available on the AMBA AXI4 Lite bus. Eight bit-move blocks compete for bus access and the bandwidth arbitrator selects the appropriate bit master. The simulation results verify the functionality of the bit move block on the AXI interface. It has been proved that the developed design completed the bit-movement operation within the specified number cycle as the design passed all the test cases successfully. The designed project can be used with direct memory access controllers to move data from one address or memory to another address or memory. The AXI fabric can be used in applications having multiple masters communicating with the memory. These applications of the designed project help to achieve advancements in the technological industry. The designed project can be incorporated for the design of high speed and high-performance applications owing to the high performance and speed at which bit-move transactions are completed.

References

[1] L.Benini, D. Bertozzi, "Network-on-chip architectures and design methods", IEE Proc.-Computer. Digital. Tech., March 2005 vol. 152, no. 2, pp. 166-176.
[2] Raghunathan A, Lahir. K, Lakshmi Narayana G., “LOTTERYBUS: a new high-performance communication architecture for system-on-chip designs,” in Proceedings of Design Automation Conference, May 2005, pp 1-58113-297-2.
[3] GruruPrasad S.P, Sudharshan K.M “Design, and Analysis of Master module for AMBA AXI4”, September 2011, pp NCECS-2011.
[4] Anurag Vatsav. G, Tomar, Ashutosh Kumar Singh “Performance Comparison of AMBA Bus-Based System-On-Chip Communication Protocol”, 29 July 2011, pp. 449-454.
[5] Pradeep S, Laxmi C, “Design and verification environment for amba AXI protocol for soc integration”, International Journal of Research in Engineering and Technology, May 20114, ISSN: 2319-1163, vol. 3, issue 3.
[6] M. Jones, Design specification for bit move. San Jose, 2019, pp.1-2.
[7] E. Clifford and Cummings, "Coding And Scripting Techniques for FSM Designs with Synthesis-Optimized Glitch-Free Outputs", SNUG (Synopsys Users Group Boston MA 2000) Proceedings, September 2000.

ISBN: 978-988-14049-2-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)