PULMONOLOGY

Development of an AXI Memory Controller IP Core for FPGA Implementation

Dr. P. V. Deshmukh*, Nitesh Kumar
* Professor, Department of Electrical Engineering, JSPM's Rajarshi Shahu College of Engineering, Pune, Maharashtra, India
Ph.D. Scholar (Electrical Engineering), Dr. Babasaheb Ambedkar Technological University, Lonere, Maharashtra, India

ABSTRACT
This paper proposes an implementation of AXI 2.0 protocol which removes the limitation of communication architecture, which would otherwise reduce the speed of data transfer in System on chip. We have also implemented DDR3 controller which was then interface with AXI 2.0 protocol. In comparison with earlier generations, DDR1/2 SDRAM, DDR3 SDRAM is a higher density device and achieves higher bandwidth due to the further increase of the clock rate and reduction in power consumption. Proposed protocol was synthesized on Xilinx 13.1 and simulated using Modelsim 6.5e.

KEYWORDS: AXI, DDR3, Modelsim, Xilinx.

INTRODUCTION
With the need of application, chip with a single processor can't meet the need of more and more complex computational task. We are able to integrate multiple processors on a chip thanks to the development of integrated circuit manufacturing technology Now as there are multiprocessing units and processors is getting faster, so compatibility with slow communication architectures a bit difficult furthermore this slow and conventional communication architecture limits the throughput.

To improve the performance we have to develop such efficient on chip Architecture which will be much faster system on chip solution which removes the limitation of communication architecture one of the solution is "AHB bus" but it can't give perfect parallelism as it can allow only one master to communicate at one slave only. While in our design there are five independent transfer channels which make multiple masters access multiple slaves at the same time and gain a perfect parallelism performance in MPSOC design.

As we have seen in AXI 1.0 protocol to achieve high speed communication between processor for on chip communication while we have developed AXI 2.0 Protocol. To improve the performance we have to develop such efficient on chip architecture which will be much faster system on chip solution which removes the limitation of communication architecture. One of the solution is "AHB bus" but it can't give perfect parallelism as it can allow only one master to communicate at one slave only. While in our design there are five independent transfer channels which make multiple masters access multiple slaves at the same time and gain a perfect parallelism performance in MPSOC design.

In this research work, AXI 2.0 protocol is implemented which removes the limitation of communication architecture, which would otherwise reduce the speed of data transfer in System on chip have also implemented DDR3 controller which was then interface with DDR3.

PROPOSED METHOD
The AMBA AXI protocol is targeted at high-performance, high-frequency system designs and includes a number of features that make it suitable for a high-speed submicron interconnects [1].

The objectives of the latest generation AMBA interface are to be suitable for high-bandwidth and low-latency designs Enable High Frequency operation using complex bridges meet the interface requirements of a wide range of components be suitable for memory controllers with high initial access latency provide flexibility in the implementation of interconnect architectures be backward-compatible with existing AHB and APB interfaces. The key features of the AXI protocol are:

- Separate address/control and data phases.
- Support for unaligned data transfers using byte strobes.
- Burst-based transactions with only start address issued.
- Separate read and write data channels to enable low-cost Direct Memory Access (DMA).
- Ability to issue multiple outstanding addresses.

- Out-of-order transaction completion.
- Easy addition of register stages to provide timing closure.

FLOW DIAGRAM FOR PROPOSED WORK

In this block diagram 1, we are showing that how our interconnect communicate with DDR3 RAM suppose master wants to send some data to the DDR3 RAM then first req is transferred to master interface which made valid signal high to pass req of processor to transfer the req to the interconnect and finely to the DDR3 controller. This required information of valid signal may be respond positively at slave itself by asserting the ready signal high. Note that, according to protocol the communication between master and slave may be established once both ready and valid signal are made high once when both the signal high then the communication between interfaces established and data transfer will be carried out. Here in this diagram the decoder has purpose of selecting the DDR3 controller as one of slaves.

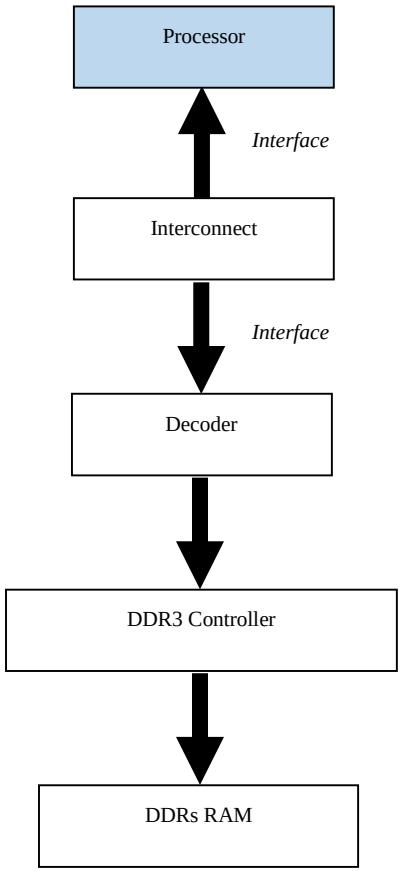

Figure 1: AXI Interface Protocol Interfaced with DDR3 Controller

Interconnect

A typical system consists of a number of master and slave devices connected together through some form of interconnect, as shown in Figure 2.

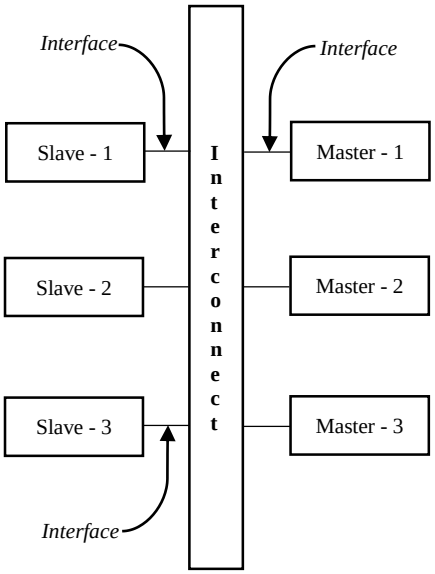

Figure 2: Interconnect

Interconnect is the main heart of this communication bus as it provides all the intelligence. It contents five individual channels for transferring the address and data and address generator. It is also able for transaction reordering mean suppose master1 wants to communicate the slave1 and slave 2 first it sends the address to the slave 1 and some no of bytes require to transfer.
The AXI protocol provides a single interface definition for describing interfaces:

- Between a master and the interconnect
- Between a slave and the interconnect
- Between a master and a slave

The interface definition enables a variety of different interconnect implementations. The Interconnect between devices is equivalent to another device with symmetrical master and slave ports to which real master and slave devices can be connected. Most systems use one of three interconnect approaches:

- Shared address and data buses.
- Shared address buses and multiple data buses.
- Multilayer, with multiple address and data buses.

In most systems, the address channel bandwidth requirement is significantly less than the data channel bandwidth requirement. Such systems can achieve a good balance between systems performance and interconnect complexity by using a shared address bus with multiple data to enable multiple data transfer.

DDR3 Controller

DDR3 was the next generation memory introduced in the summer of 2007 as the natural successor to DDR2. DDR3 increased the pre-fetch buffer size to 8-bits an increased the operating frequency once again resulting in high data transfer rates than its predecessor DDR2. In addition, to the increased data transfer rate memory chip voltage level was lowered to 1.5 V to counter the heating effects of the high frequency. By now one can see the trend of memory to increase pre-fetch buffer size and chip operating frequency, and lowering the operational voltage level to counter heat. The physical DDR3 is also designed with 240 pins, but the notched key is in a different position to prevent the insertion into a motherboard RAM socket designed for DDR2. DDR3 is both electrical and physically incompatible with previous versions of RAM. In addition to high frequency and lower applied voltage level, the DDR3 has a memory reset option which DDR2 and DDR1 do not. The memory reset allows the memory to be cleared by a software reset action [2]. Other memory types do not have this feature which means the memory state is uncertain after a system reboot. The memory reset feature insures that the memory will be clean or empty after a system reboot. This feature will result in a more stable memory system. DDR3 uses the same 240-pin design as DDR2, but the memory module key notch is at a different location [3].

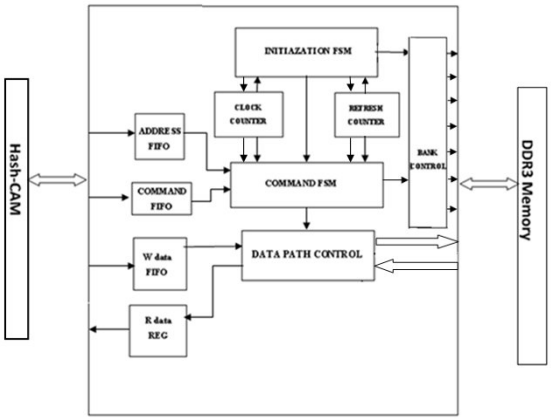

Figure 3: Functional block diagram of DDR3 controller

The architecture of DDR3 SDRAM controller consists of Initialization FSM Command FSM, data path, bank control, clock counter, refresh counter, Address FIFO, command FIFO, Wdata FIFO and R_data reg. Initialization FSM generates proper i-State to initialize the modules in the design. Command FSM generates c-State to perform the normal write, read and fast write, read operations. The data path module performs the data latching and dispatching of the data between Hash CAM unit and DDR3 SDRAM banks. The Address FIFO gives the address to the Command FSM so the bank control unit can open particular bank and address location in that bank. The Wdata FIFO provides the data to the data path module in normal and fast write operation [4]. The R_data reg gets the data from the data path module normal and fast read operation. The DDR3 controller gets the address, data and control from the HASH CAM circuit in to the Address FIFO. Write data FIFO and control FIFO respectively [5].

Address FIFO

DDR3 SDRAM controller gets the address from the Address FIFO so that controller can perform the read from the memory or write in to the memory address location specified by the Address FIFO. Here the Address FIFO width is 13 bit and stack depth is 8.

Write data FIFO

DDR3 SDRAM controller gets the data from the Write data FIFO in write operation into the memory address location specified by the Address FIFO. Here the Address FIFO width is 64 bit and stack depth is 8.

Control FIFO

DDR3 SDRAM controller gets the command from the Control FIFO controller can perform the read from the memory or write in to the memory address location specified by the Address FIFO. Here the Control FIFO width is 2 bit and stack depth is 8. If the control FIFO gives the "01" DDR3 controller performs the Normal read operation. If the control is "10" DDR3 controller performs the Normal read operation and if control is "11" DDR3 controller performs the Fast read operation [6].

Read Data Register

When DDR3 controller performs Normal read or Fast read operation Read data register gets the data send to the Hash Cam circuit.

Finite State Machine

Different states of Initial FSM:

Idle: When reset is applied the initial FSM is forced to IDLE state irrespective of which state it is actually in when system is in idle it remains idle without performing any operations.

No Operation (NOP): The NO OPERATION (NOP) command is used to instruct the selected DDR SDRAM to perform a NOP (CS# is LOW with RAS#, CAS#, and WE# are HIGH). This prevents unwanted commands from being registered during idle or wait states. Operations already in progress are not affected.

Precharge (PRE): The PRECHARGE command is used to deactivate the open row in a particular bank or the open row in all banks as shown in Figure 4. The value on the BA0, BA1 inputs selects the bank, and the A10 input selects whether a single bank is precharged or whether all banks are precharged[3][7].

Auto Refresh (AR): AUTO REFRESH is used during normal operation of the DDR SDRAM and is analogous to CAS#-before-RAS# (CBR) refresh in DRAMs. This command is nonpersistent, so it must be issued each time a refresh is required. All banks must be idle before an AUTO REFRESH command is issued.

Load Mode Register (LMR): The mode registers are loaded via inputs A0–An. The LOAD MODE REGISTER command can only be issued when all banks are idle, and a subsequent executable command cannot be issued until tMRD is met.

Read/Write Cycle: The Figure 5 shows the state diagram of CMD_FSM which handles the read, write and refresh of the SDRAM. The CMD_FSM state machine is initialized to c_idle during reset. After reset, CMD_FSM stays in c_idle as long as sys_INIT_DONE is low which indicates the SDRAM initialization sequence is not yet completed. Once the initialization is done, sys_ADSn and sys_REF_REQ will be sampled at the rising edge of every clock cycle. A logic high sampled on sys_REF_REQ will start a SDRAM refresh cycle. This is described in the following section. If logic low is sampled on both sys_REF_REQ and sys_ADSn, a system read cycle or system write cycle will begin. These system cycles are made up of a sequence of SDRAM commands [7].
Initial FSM State Diagram:

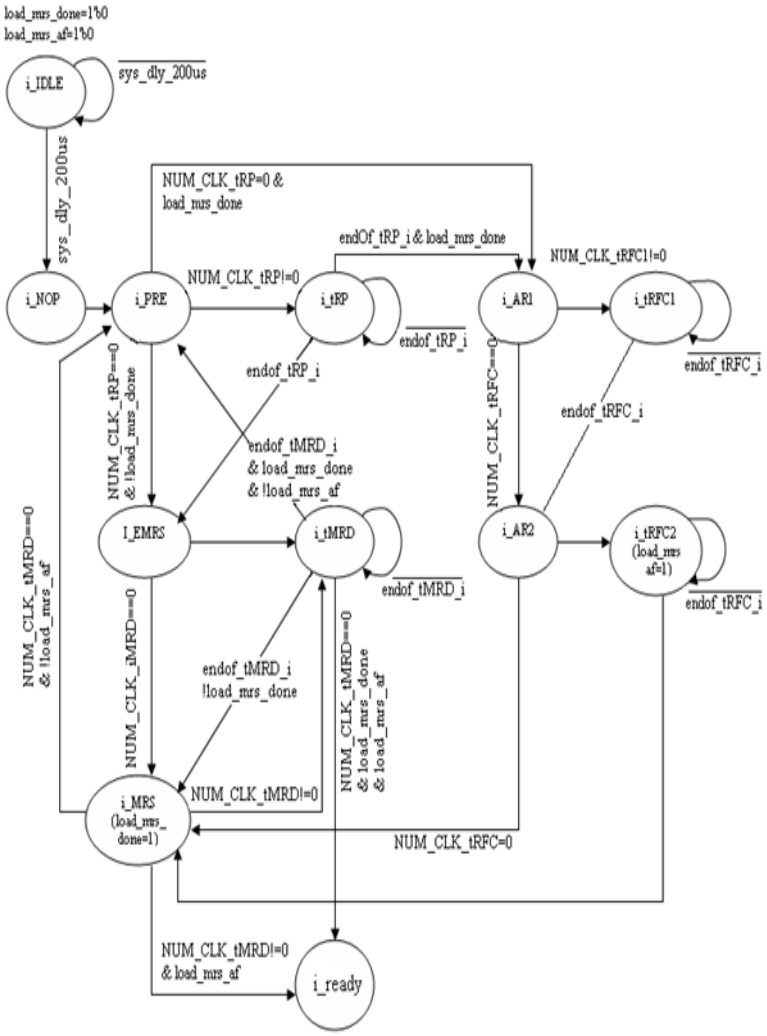

Figure 4: Initial FSM State Diagram

Command FSM State Diagram:

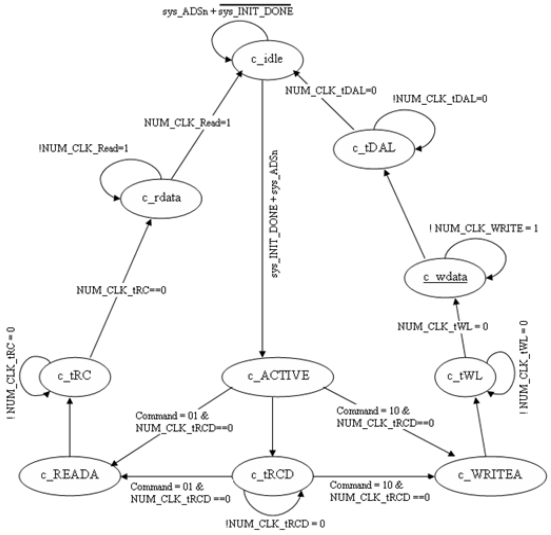

Figure 5: Command FSM State Diagram

BLOCK DIAGRAM OF WRITE ADDRESS CHANNEL

Figure 6 shows the block diagram of Write address channel. Here idappender has responsibility of adding the additional bit to the wid to let the slave known that from which master the particular data or address being transferred then six bit id is transferred to the idstorage block then form id storage, id is transferred to the Multiplexer form. Address storage address is transferred to Multiplexer. Here in the project we have used 4 Masters, those 4 masters are simultaneous to the Slave. Masters are giving their different control signal to the Multiplexer. All Masters are sending the valid signal to Arbiter. Arbiter can select one of the master on fixed priority basis which is known as Round Roubin method. Here in our project, arbiter is selecting the particular Master on the basis of fixed priority, hence arbiter selects the master by issuing the grant to particular master then grant signal is given to the encoder. On the basis of the grant signal, it provides the SEL signal to Multiplexer. Hence Multiplexer decides or selects the Master. Multiplexer provide all signal of selected master to the decoder and finally decoder selects the slave to which address needs to be send. Decoder selects the slave on the basis of the address sent by the slave, hence address is passed from master to slave. The function of write data signal is identical to that of write address channel so only address.

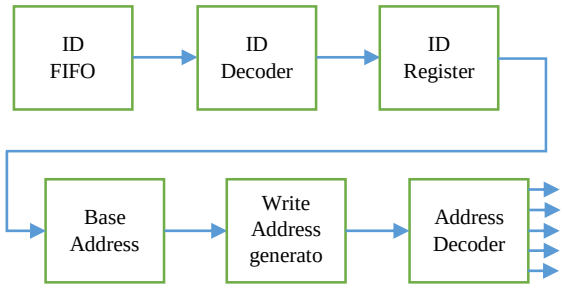

Figure 6: Block diagram of Write Address channel

BLOCK DIAGRAM OF READ ADDRESS CHANNEL

The function of read data signal is identical to that of write address channel. First of all six bit id from id storage would come on to the port from slave, four signal such as rlast, rready data, rvalid would come on to the slave port. From slave port, six bit id would go to the Id separator and decoder. Id separator/decoder gives four bit Id to the Encoder. Encoder, according to grant Id, will activate select (sel) line. Then four output form slave port will go to demultiplexer. According to selection line, demultiplexer will send the id for particular Master.

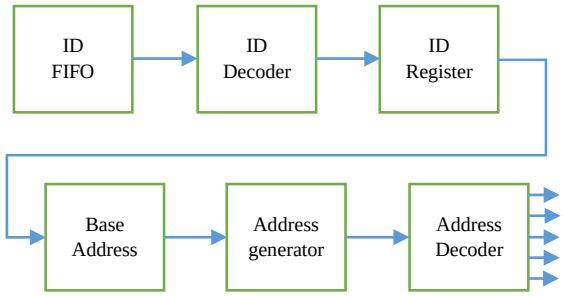

Figure 7: Block diagram of Read Address channel

BLOCK DIAGRAM OF WRITE RESPONSE CHANNEL

In AXI interconnect write operation needs to be responded, this response is not given for signal transfer. Response operation is started when completion signal occurs for a burst. The completion signal occurs only after completion of signal burst not for signal transfer that means after completion of every burst, completion signal is generated and this response is given through response channel. Here in this block diagram 1, 2, 3 and 4 are respectively bvalid, bid1, bvalid and bready and 5, 6, 7 and 8 are bvalid1, bid1, bvalid and bready1. In this protocol, four types of responses are occurs; OK, Exclusive OK, SLERR and DECERR.

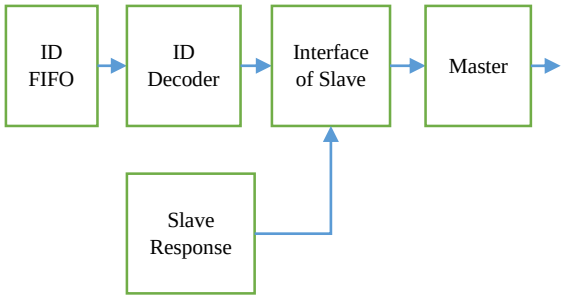

Figure 8: Block diagram of Write Response Address channel

The operation starts as completion signal is generated, which leads to generation of Bvalid signal by slave. In response to the assertion of Bvalid signal, Bready signal also gets high then signal of slave port (Bvalid bready bid bresponse) also gives 6 bit Id signal to Id decoder/separator as result of which Id decoder/separator. According to 6 bit Id, grant signal is provided to the encoder, then encoder issues grant for a particular master to whom response needs to be transferred. In Demultiplexer there are 4 signals is coming which on the basis of sel signal chooses master. In demultiplexer 4 bit Id is coming from Id decoder/separator which tell that for which transaction response has been given.

BLOCK DIAGRAM OF READ DATA CHANNEL

The read data channel conveys both the read data and any read response information from the slave back to the master. The read data channel includes:

- The data bus that can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide.
- A read response indicating the completion status of the read transaction.

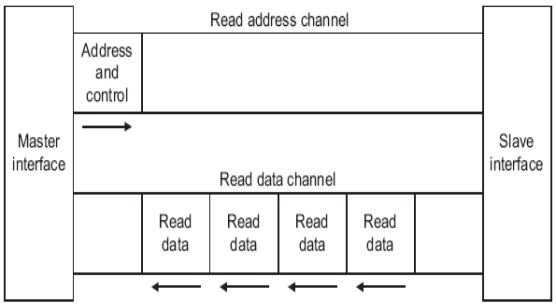

Figure 9: Read address and data channel

SIMULATION AND RESULTS

Simulation is carried out on Modelsim 6.5e simulator and syntheses is done using Xilinx 13.1.

RTL Schematic

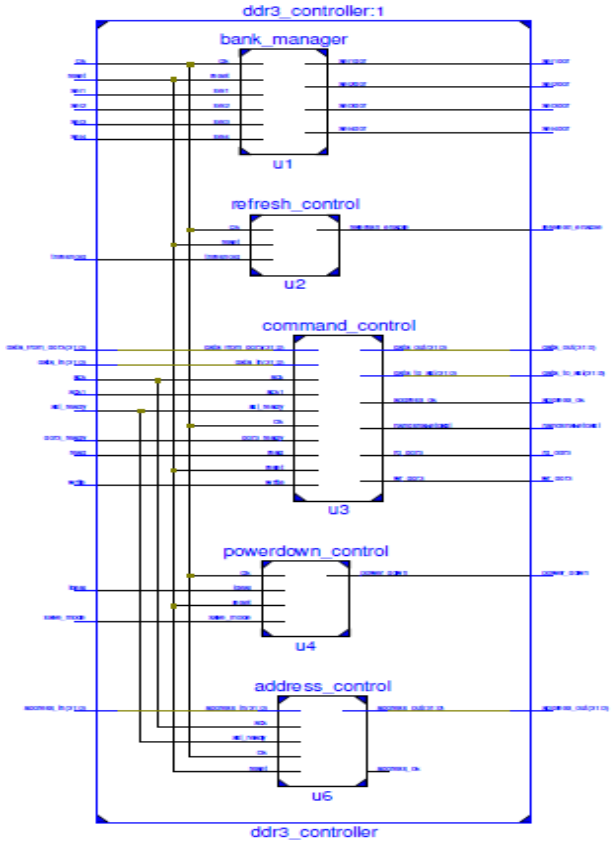

Figure 10: RTL schematic of DDR3 controller top entity

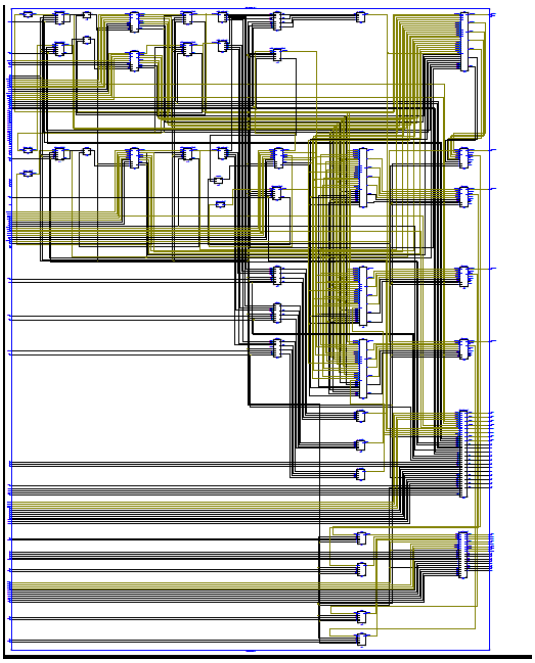

Figure 11: RTC schematic of address Storage block

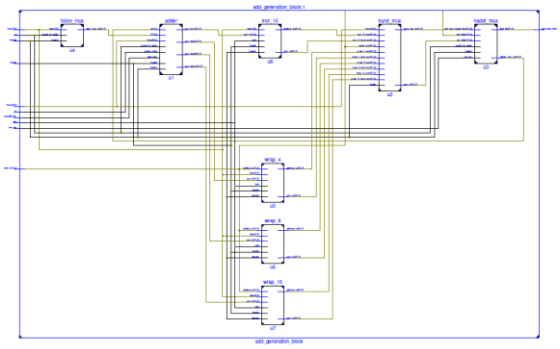

Figure 12: RTL schematic of address generation block

Simulation Waveforms

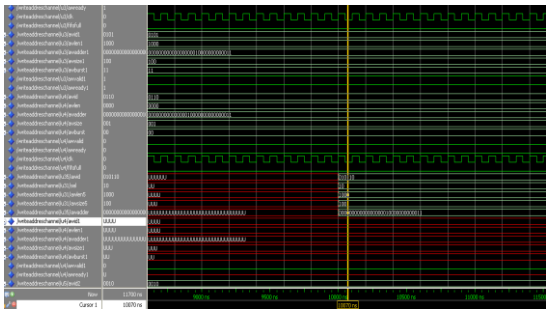

Figure 13: Simulation waveform for Write Address Channel

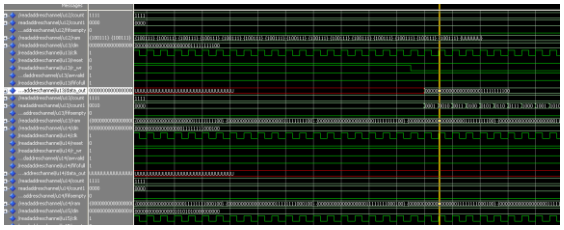

Figure 14: Simulation waveform for Master

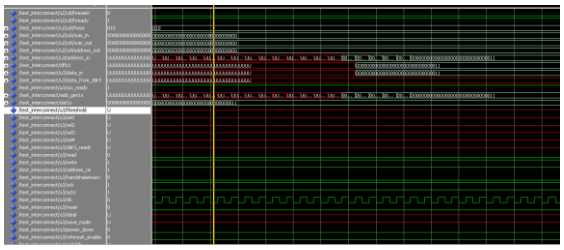

Figure 15: Simulation waveform for write operation with DDR3

Here in the simulation u3 component is master from where address needs to be transferred input signals of so address which was given to the master (u3) is "00000000000000000010000000000011" is and awid is "0101". Both of this awid and awaddress signals are being transferred to u31 component as shown in figure, hence address output signal of slave port is receiving the a awaddress signal as "00000000000000000010000000000011" and awid as "0101".

Device Utilization Summary
Selected Device: 6slx100tfgg900-3

Slice Logic Utilization
Number of Slice Registers: 1122 out of 126576 0%
Number of Slice LUTs: 868 out of 63288 1%
Number used as Logic: 644 out of 63288 1%
Number used as Memory: 224 out of 15616 1%

Number used as RAM : 224

Slice Logic Distribution
Number of LUT Flip Flop pairs used: 1472
Number with an unused Flip Flop: 350 out of 1472 23%
Number with an unused LUT: 604 out of 1472 41%
Number of fully used LUT-FF pairs: 518 out of 1472 35%
Number of unique control sets: 95

IO Utilization
Number of IOs: 833
Number of bonded IOBs: 727 out of 498 145% (*)
IOB Flip Flops/Latches: 128

Specific Feature Utilization
Number of BUFG/BUFGCTRL/BUFHCEs: 1 out of 16 6%

CONCLUSION

AXI bus specification and a technology independent methodology for designing of IP Core Read and Write operation is satisfied. We have implemented AXI 2.0 protocol which removes the limitation of communication architecture, which would otherwise reduce the speed of data transfer in System on chip. We have also implemented DDR3 controller which was then interface with AXI 2.0. DDR3 controller improves power consumption and heat generation, as well as enabling more dense memory configurations for higher capacities. Proposed approach was synthesized with Xilinx 13.1.

REFERENCE

[1] "AMBA AXI Protocol specification". www.arm.com/armtech/AXI
[2] E. G. T. Jaspers, et al, "Bandwidth Reduction for Video Processing in Consumer Systems", IEEE Transactions on Consumer Electronics, Vol. 47, No. 4, Nov. 2001, pp. 885- 894.
[3] "High-Performance DDR3 SDRAM Interface in Virtex-5 Devices", Xilinx, XAPP867 (v1.0), Sept 24, 2007.
[4] www.altera.com/literature/ug/ug_altmemphy.pdf, External DDR Memory PHY Interface Megafunction User Guide (ALTMEMPHY) accessed on 23 Feb. 2009.
[5] G. Allan, "The Love/Hate Relationship with DDR SDRAM Controllers", MOSAID Technologies Whitepaper.
[6] T. Mladenov, "Bandwidth, Area Efficient and Target Device Independent DDR SDRAM Controller", Proceedings of World Academy of Science, Engineering and Technology, Vol. 18, De. 2006, pp. 102-106.
[7] DDR3 SDRAM Specification (JESD79-3A), JEDEC Standard, JEDEC Solid State Technology Association, Sept. 2007.
[8] Vijaykumar, R K Karunavathi, Vijay Prakash, "Design of Low Power Double Data Rate 3 Memory Controller with AXI compliant", International Journal of Engineering and Advanced Technology (IJEAT), ISSN: 2249 – 8958, Volume 1, Issue 5, June 2012.
[9] Osborne, S., Erdogan, A.T. Arslan, T., Robinson, D., "Bus encoding architecture for low- power implementation of an AMBA-based SoC platform", IEEE Proceedings on Computers and Digital Techniques, Vol. 149, Issue 4, July 2002.
[10] Fu-ming Xiao, Dong-sheng Li, Gao-Ming Du, Yu-kun Song, "Design of AXI bus based MPSoC on FPGA", 3rd International Conference on Anti-counterfeiting, Security, and Identification in Communication (ASID), 2009.
[11] Terry Tao Ye, LUCA BENINI, Giovanni De Micheli, "Packetized On-Chip Interconnect Communication Analysis for MPSoC," Proceeding of the DATE, Messe Munich, Germany, pp. 344-349, 2003.
[12] Sudeep Pasricha, Nikil Dutt, "On-Chip Communication Architectures: System on Chip Interconnect", Morgan Kaufmann, 2010.