International Journal of Engineering Research and Development
e-ISSN: 2278-067X, p-ISSN: 2278-800X, www.ijerd.com
Volume 8, Issue 11 (October 2013), PP. 40-46

Design of Speedy RAM Controller Using Inbuilt Memory

R.Suneel Reddy[1], R.Sravanthi Reddy[2]
[1]*PG Scholar, Department of Electronics and Communication Engineering,PBR Visvodaya Institute of Technology & Sciences, Kavali, Andhra Pradesh, India.*
[2]*Associate Professor, Department of Electronics and Communication Engineering, PBR Visvodaya Institute of Technology & Sciences, Kavali, Andhra Pradesh, India.*

Abstract:- The main objective of the paper is synchronize the RAM controller. In general the speed of fetching the data from memories is not able to match up with processors. So there is a need to synchronizes the processor speed and memory speed with the help of a controller. Apart from achieving the synchronization between processor and memory a novel feature of in-built cache memory is also included in design, which could increases the overall efficiency of the controller. The responsibility of the controller is to match the speed of the processor on one side and memory on the other side so that the communication can take place seamlessly. Hear we have built a memory controller which is specifically targeted for DRAM. Certain novel features were included in the design which could increase the overall efficiency of the controller. Such as, searching the internal memory of the controller for the requested data for the most recently used data instead of going to the RAM to fetch it. The design was implemented on Xilinx ISE till the final simulation and synthesis.

Keywords:- DRAM, VLSI, FPGA, FIFO, FSM, Memory

I. INTRODUCTION

Memory controllers contain the logic necessary to read and write dynamic random access memory, and to "refresh" the DRAM. Without constant refreshes, DRAM will lose the data written to it as the capacitors leak their charge with in a fraction of a second (not less than 64 milliseconds according to JEDEC standards [2]). The paper is organised as follows: In Section II we have given brief introduction for *generic controller architecture* and DRAM *memory*. In section III we discussed existing architecture and motivation for internal search with principle of locality, in section IV we proposed our model for controller and in section V we showed some synthesis result for the controller on vertex 5 FPGA using VHDL

II. LITERATURE REVIEW

A. Generic architecture of controller

The controller is expected to synchronise data transfer between the processor and memory. To achieve this, the controller has to achieve this, the controller has to accept the requests from the processor side and convert them to a form suitable to the memory and execute the requests. Since the processor is faster than the memory, it is illogical to make the processor wait till each command is executed for it to give the next command. So the controller has to have some kind of storage as given in figure 1, so that it can buffer multiple requests while the processor continues with other work The interface at the processor side of the controller has to synchronize to the speed of the processor where as the memory side interface has to run at the speed of memory. To achieve this we operate the controller with high frequency clock, but with wait states for the memory side interface.

B. Dynamic Random Access Memory

DRAMs store data in cell that depend on capacitors, which need to be 'refreshed' continuously since they are not able to retain data indefinitely even if the device is continuously powered up [1]. A DRAM cell consists of only a single transistor that is paired with a capacitor. The presence of charge in the capacitor determines whether the cell contains a '1' or a '0'. This single-transistor configuration is commonly referred to a 1-T memory cell.

The memory elements of a DRAM are arranged in an array of rows and columns. Each row of memory cells share a common 'word' line, while each column of cells share a common 'bit' line. Thus, the location of a memory cell in the array is the intersection of its 'word' and 'bit' lines. The number of columns of such a memory array is known as bit width of each word [1]. During a 'write' operation, the data to be written ('1' or '0') is provided at the 'bit' line while the 'word line' is asserted. This terms on the access transistor and allows the capacitor to charge up or discharge, depending on the state of the bit line. During a 'read' operation, the 'word' line is also asserted, which terms on the access transistor. The enabled transistor allows the voltage on

the capacitor to be read by a sense amplifier circuit through the 'bit' line. This sense circuit is able to determine whether a '1' or '0' is stored in the memory cell by comparing the sensed capacitor voltage against a threshold.
.

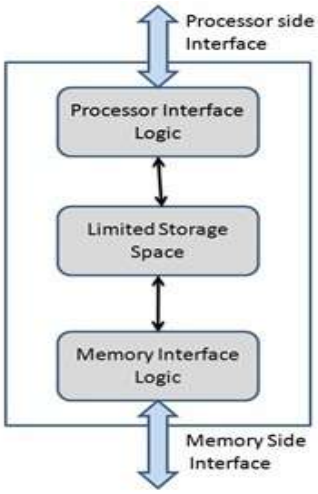

Figure 1: Generic block diagram of a DRAM Controller

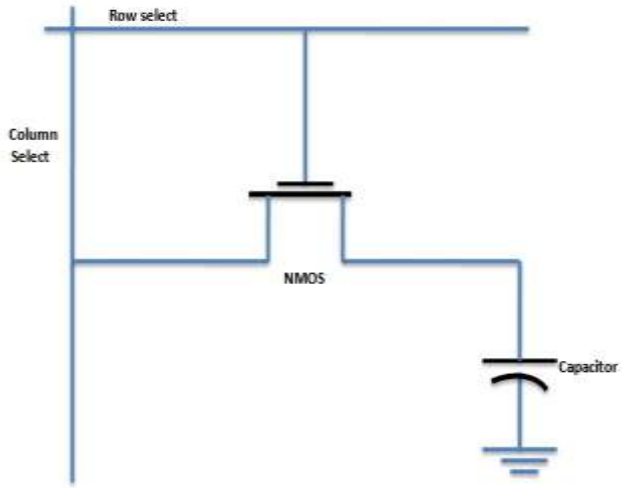

Figure 2: DRAM Cell

For DRAMs, the simple operation of reading the data of a memory cell is destructive to be stored data. This is because the cell capacitor undergoes discharging every time it is sensed through the 'bit' line. In fact the stored charge in a DRAM cell decays over time even if it doesn't undergo a 'read' operation. Thus, in order to preserve the data in a DRAM cell, it has to undergo what is known as a 'refresh' operation [6]. A refresh operation is simply the process of reading a memory cell's content before it disappear and then writing it back into the memory cell. Typically it is done every few milliseconds as per word [2]. Aside from its memory array, a DRAM device also needs to have the following support circuitries to accomplish its functions :1)a decoding circuit for row address and column address selection;2) a counter work tracking the refresh operation sequence;3)a sense amplifier for reading and restoring the charge of each cell; and 4) a write enable circuit to put the cell in 'write' mode. ie., make it ready to accept a charge[3][6].

III. MOTIVATION AND DESIGNING WORK

Existing controller [3][4] have two different FIFO one to store the write requests and one to store to read requests. As soon as the request is serviced it discards the corresponding data from the FIFO. But all processors follow the principle of locality which is described below. This paper tries to exploit the behaviour of the processors. So we have proposed a method where in the requests are stored in the controller even after the request has been serviced. This enables us to search the internal FIFO before accessing the RAM, to check if the data already exists in the internal memory. This reduces the turn-around time for requests significantly if the data was found internally.

Principle of Locality: Locality of reference, also known as the principle of locality, is the phenomenon of the same value or related storage locations being frequently accessed. There are two basic types of reference locality – *temporal locality* and *spatial locality.* Temporal locality refers to the reuse of specific data and/ or resources within relatively small time durations. If at one point in the time a particular memory location is referenced, then it is likely that the same location is referenced, and then it is likely that the same location will be referenced again in the near future. In this case it is advisable to store a copy of the referenced data in the special small memory, which can be accessed faster. Spatial locality refers to the use of data elements within relatively close storage locations. If a particular memory location is referenced at a particular time, then it is likely that near by memory location will be referenced in the near future. These two properties motivated us to use internal search module in inbuilt memory(FIFO in our case) inside the controller, so that memory can act as a cache for faster data access. So a cache like behaviour in the controller which can useful for some embedded processors which don't have in-built cache. This increases the overall efficiency of the controller.

IV. DESIGN DETAIL

This section discusses the designing part of the controller. In sub- section A we described different hardware modules and input/output signals, sub-section B explains the state machines of interfaces (one is processor/user side and other is DRAM side). Sub-section C and D gives internal memory structure and search engine logic respectively.

A. Proposed Architecture

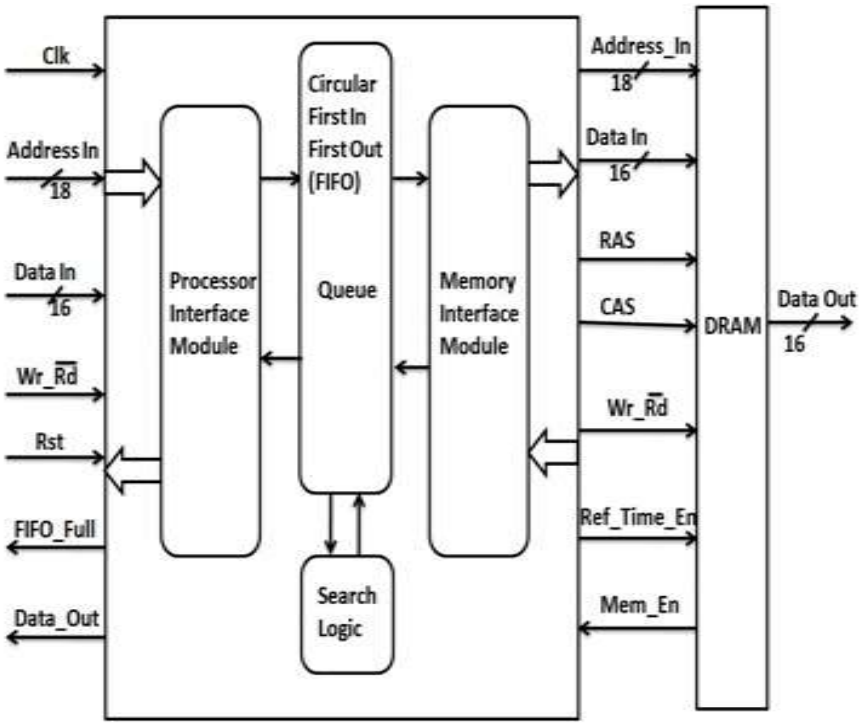

Figure 3: Proposed Architecture for DRAM Controller

Table 1: Processor Side Interfacing Signals

Signal name	Description
Address In	This is the memory address input to the controller, SIZE IS 18 BIT
Data In	This is the data input for the corresponding memory address, SIZE IS 16 BIT
RD'	This is an active low signal to indicate whether the operation is a read
WR'	This is an active low signal to indicate whether the operation is a write
Clk	Clock input to the controller
Data out	The data which is read from the memory is given back to the processor through this port SIZE IS 16BIT
Read data Valid	This signal indicates to the processor that the data on the bus is valid
FIFO Full	This indicates that the internal FIFO is full and the controller cannot accept any more requests
Error	An error operation occurred

In the above table 1 all signals at user/processor side is explained.

Table 2: Memory Side Interface

Signal name	Description
Address	This is the multiplexed address bus to the memory, SIZE IS 13 BIT, ROW ADDRESS IS 10 BIT, COLUMN ADDRESS IS 8 BIT.
Data	This bidirectional bus carries data to and from the memory, SIZE IS 16 BIT
RAS'	This is the Row Address Strobe signal to the DRAM
CAS'	This is the Column Address Strobe signal to the DRAM
OE	This signal enables the DRAM data output
LWE, UWE	Indicate which part of a word is being written (for future enhancement)

In the above table 2 all signals at the memory side is explained.

Hardware Module

1. *Processor Interface Module:* This module provides the necessary logic to handle the processor requests.
2. *Memory Interface Module:* This module provides the necessary logic to interface with the memory side

signals.

3. *Circular FIFO:* This unit is used to store the requests coming from the processor. It acts as an interface between the processor interface module and memory interface module.
4. *Search Unit:* This module provides the logic for searching within the FIFO.

B. State Machine

The controller had two state machines ,one for the processor interface shown in figure 4 and one for the memory side interface shown in figure 5,the memory side state machine had 16 states where as the processor side interface had 4 states and is designed by FSM. As in the figure 4 the IDLE state it check whether that memory controller FIFO is empty or full. If it is full the next state FIFO FULL state else it will go to the STORE FIFO. In this state storing the all processor side requests (i.e address, data and control bit) into the memory controller. Then the next state will be the increment state here increment the write pointer and go to the IDLE state. The Memory Interface Module is the one which accepts the requests, executes then and finally write back into the FIFO. This operates on a state machine which is shown in figure 5.This module reads only when the FIFO is not empty. The outputs of the Memory Interface Module are the signals for DRAM and it contains refresh logic to refresh the DRAM. This works based on the timer. Once the timer is run out the refresh signal is asserted.

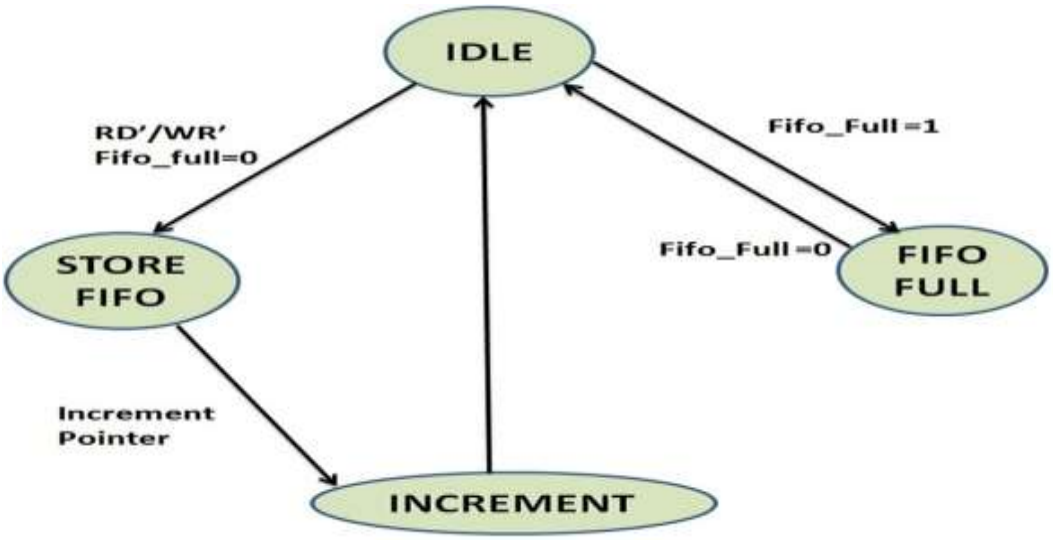

Figure 4: Processor side state machine

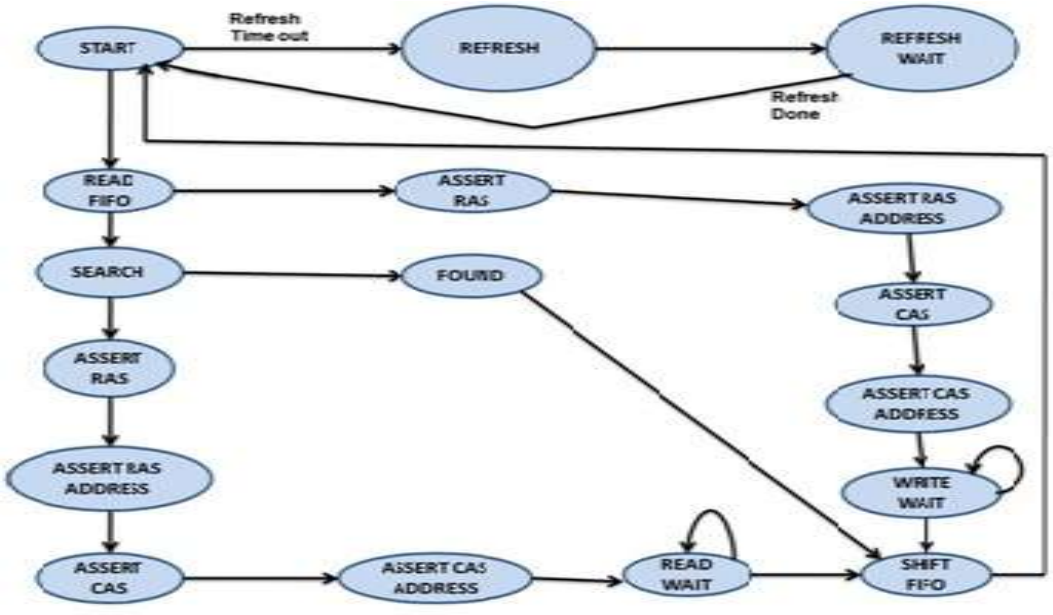

Figure 5: Memory side state machine

C. FIFO Structure

The controller uses a first in first out (FIFO) queue to store the requests coming from the processor. This FIFO is 36 bits wide, i.e. 18 bits for address, 16 bits for data and 2 flag bits. The two flag bits are Read/Write and executed/not executed (ex), as shown in figure 6.The depth of the FIFO depends on the following factors, one is the difference in speed between memory and processor and other is amount of area overhead acceptable.

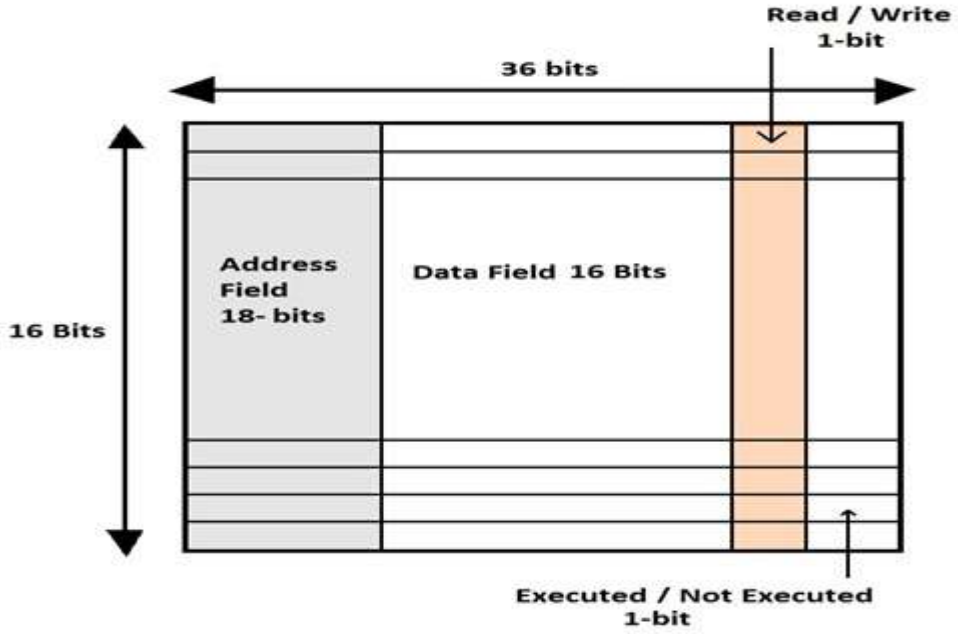

Figure 6: FIFO Structure

In this paper we have chosen the FIFO depth to be 16 locations for the ease of testing the functionality. This FIFO is a circular FIFO. That is, the contents of the FIFO are always read from the top of the FIFO (location0). Once this command is executed, the FIFO is shifted circularly, i.e. the contents of the first location moves into the last location and all other contents get shifted up by one location.

D. Search Module

One of the novel features of this controller is the internal search which is carries out before giving the request to the DRAM. When the controller encounters a read command at the head of the FIFO queue, it takes this command and searches the FIFO to see if the required data is already in the FIFO.

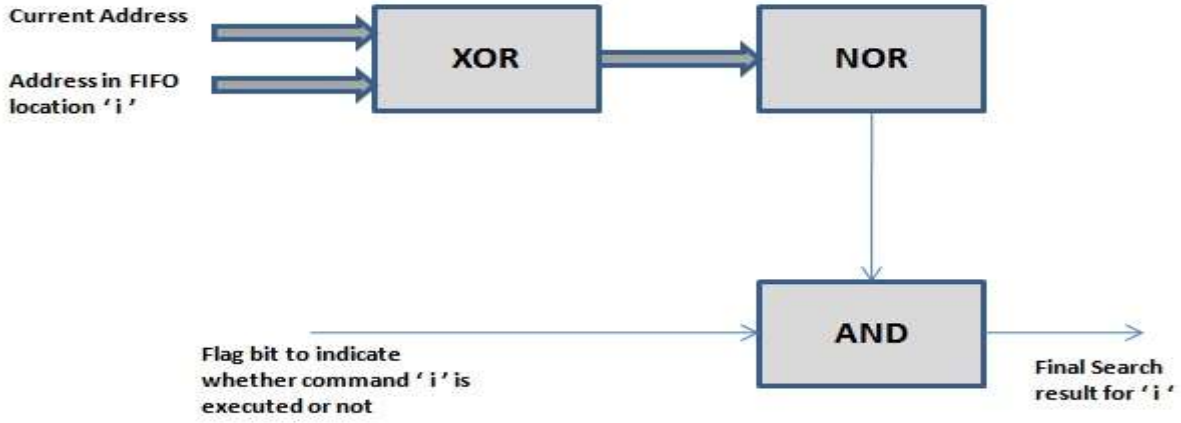

Figure 7: Search Logic

This search is carried out in parallel and all the FIFO locations are searched in one clock cycle. Hence the searching time is not dependent on the depth of the FIFO but at the cost of hardware overhead. The flag bit (executed / not executed) aids in the search. Once the data is found in the FIFO, which was the result of a previous operation, it is directly routed to the output. Hence saving the time and burden of fetching it from the

DRAM memory. If the data is not found in the FIFO then the controller executes the request on the memory as usual. Thus if the data is found internally then we save a lot of clock cycles, but even if the data is not found internally then we just lose one clock cycle. The searching is carried out only for read requests as there is no meaning in doing it for write requests.

During the search state every bit of the address of the current request is XORed with every bit of all the other addresses in the FIFO which have the same parity. If a match of the addresses occurs then the output for that operation will be all zeros. So all the bits of each of the results is NORed. So if a match is found, the output of this NOR operation will be a logic 1, else logic 0. Now this bit is ANDed with the flag bit which says whether the command has been executed or not, thus giving the final search result as shown in figure 7. If multiple addresses match then the most recently executed one is considered as a match. This is one by feeding this output bits to a priority encoder. The output of the priority encoder logic is the FIFO address which contains the required data. This is then accessed in the next clock cycle and routed to the output. Here 'i' ranges from 0 to FIFO_DEPTH – 1

V. SIMULATION AND SYNTHESIS RESULTS

The simulation and synthesis is carried out on Xilinx ISE 9.2i

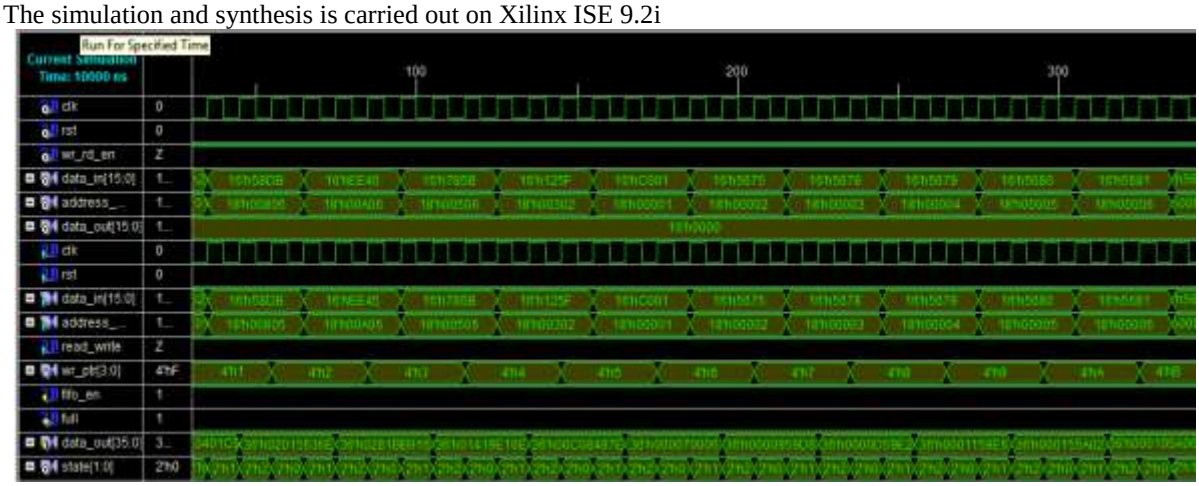

Figure 8: Simulation Results

The Synthesis has been succesfully completed on vertex5 XC5VLX30 with speed grade-1 produsedfollowing results.

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	822	19200	4%
Number of Slice LUTs	1172	19200	6%
Number of fully used Bit Slices	808	1186	68%
Number of bonded IOBs	53	220	24%
Number of BUFG/BUFGCTRLs	2	32	6%

Figure 9: Device Utilization

Clock report: Maximum combinational path delay is 25.568ns

VI. CONCLUSION AND FUTURE WORK

We proposed a novel way i.e. cache like behaviour inside controller to improve its performance and hence the bandwidth. We used FIFO to store the Read/Write commands coming from processors/user side along with corresponding write data and included a search engine to search recently read/write data inside the FIFO in order to reduce the clock cycles of fetching data from DRAM. The methodology provided good lab results and synthesized well. This concept will be useful mainly in embedded processors which may not have cache in them and also do not access the memory in blocks. Future work can include refining the design for a specific memory module and a target processor. The design can also be extended for SDRAM and DDR SDRAM.

REFERENCES

[1]. B. Keeth and R. J. Baker, "DRAM Circuit Design", IEEE Press Series on Microelectronic Systems",
[2]. New York, 2000

[3]. "Jedec standard: double data rate (DDR) SDRAM specification", (revision of jesd79b), jedec solid state technology association,march 2003,

[4]. J. Hassoun, "Virtex Synthesizable High Performance SDRAM Controller", Xilinx (white paper), may 7,1999.

[5]. K. Palanisamy and R. Chiu, "High-Performance DDR2 SDRAM Interface in Virtex-5 Devices", xilinx, may 8,2008

[6]. A. Cosoroaba, "Memory Interfaces Made Easy with Xilinx FPGAs and the Memory Interface Generator",xilinx,February 16, 2007

[7]. A. S. Sedra and K. C. Smith, "Microelectronic Circuits", Oxford Series in Electrical Engineering, 4th edition.