Journal of Multidisciplinary Engineering Science and Technology (JMEST)
ISSN: 3159-0040
Vol. 2 Issue 4, April - 2015

Design Architecture for Faster Ram Controller Using Internal Memory

Soumya Karnala
Department of Computer Science
University of Bridgeport
126 Park Avenue, Bridgeport, CT
skarnala@my.bridgeport.edu

Tarik El Taeib
Department of Computer Science
University of Bridgeport
126 Park Avenue, Bridgeport, CT
teltaieb@my.bridgeport.edu

***Abstract*—In this paper we are constructing a faster RAM by using inbuilt memory instead of external memory. In this era of fast processors and processors with many types, there is a need for more fast and much large memories. But the speed of fetching data from memories is not able to match up with speed of different processors. So there is am for a requirement for faster memory controller. The duty of the controller is to balance the speeds of the processor on one side and memory on the other. The controller is expected to synchronize data transfer between the processor and memory. To achieve this, the controller has to accept the requests from the processor side and convert them to a form of suitable to the memory and execute the requests. The processor has more speed compared with the memory. It is impossible to make the processor hold till each command is executed for it to give the next command, so that the controller has some kind of storage.**

In this proposed research work the memory controller has inbuilt memory, so that it can buffer multiple requests while the processor continues with other work. The memory we consider here is DRAM. Memory controller will have the logic to read and write to DRAM and also revive the DRAM. The memory controller has both processor interface logic, inbuilt memory and memory interface logic. The controller at processor side interface has to synchronize to the speed of the processor whereas the memory side interface has to run at the speed of memory. With this feature the design which 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 RAM fetch it. For inbuilt memory of the controller we are using FIFO. In this research work, the requests are kept in the controller even after the request has been serviced. The DRAM memory elements are arranged in an array of rows and columns. Search logic provides the logic for searching in the FIFO. The entire system construction will be architected using HDL language and simulation, synthesis and FPGA implementation will be done using various FPGA based EDA Tools.

INTRODUCTION

The controller is expected to synchronize data transfer between the processor and the memory. To obtain this, the controller has to acquire the requests from the processor side and convert them to a form suitable to the memory and execute the requests. As the processor is much faster than the memory, it is impossible to make the processor halt till each command is executed for it to give the coming next command. So the controller has to have any type of storage, 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 whereas the memory side interface has to run at the pace of memory. To get this, we manage to operate the controller with high frequency clock, but with wait states for the memory side interface. As the processor-memory performance gap creates a delay in the execution of the requested commands, so if we introduce a memory in order to speed up the execution.

DESIGN

General Block Diagram:

This is a general block diagram of the control which access the data transfer between processor and the memory. This controller consists of three major blocks. This are explained below.

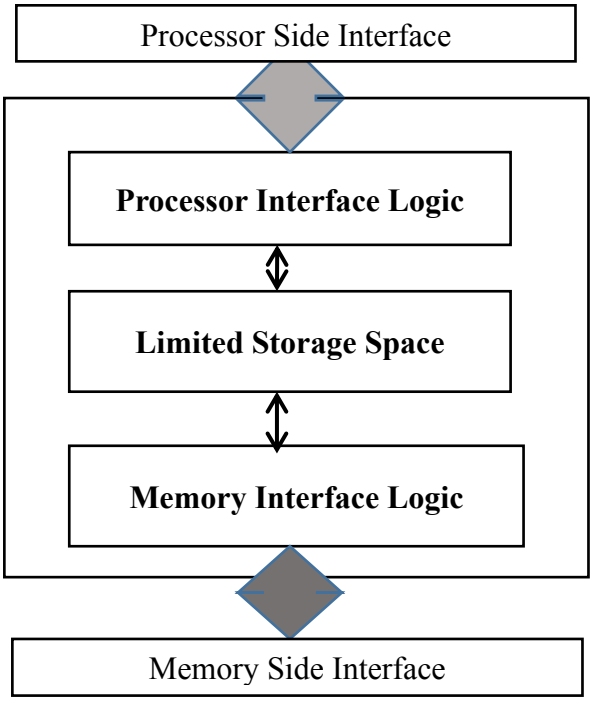
Processor side interface:

This is a first block which helps the controller to fetch the requests from the processor. After fetching the requests, it gives them to controller in order to execute the execution.

Controller: The controller contains three sub blocks in it. It has storage, interface logic of both processor and the memory.

Memory side interface:

Memory side interface gives the instructions which are fetched from the controller logic. It executes the requested commands and gives results.



General Block Diagram

Hardware Modules:

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.

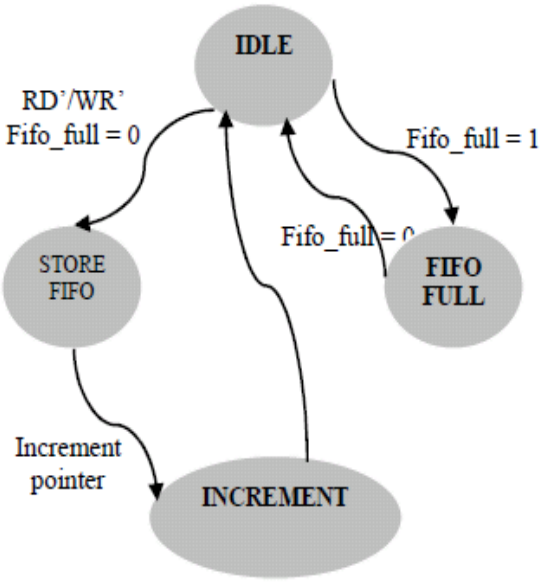
3. Circular FIFO: This unit is used to store the requests coming from the processor. It acts as an interface between the processor interface module and memory interface module.

4. Search Unit: This module provides the logic for searching within the FIFO.

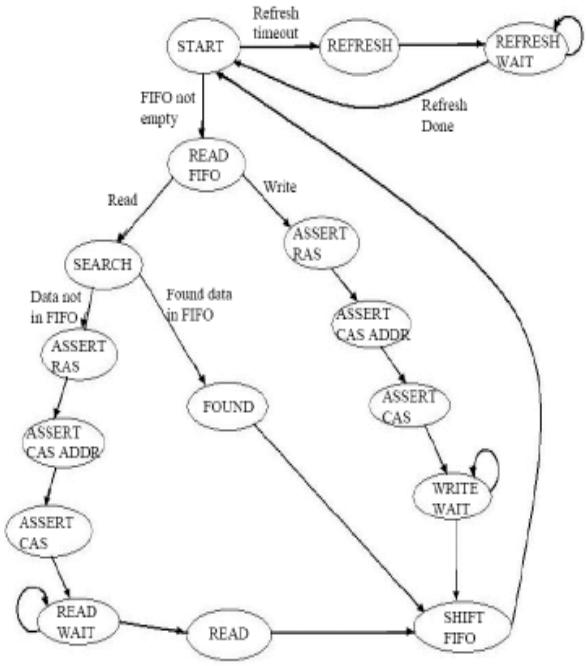
Finite State Machine:

A finite-state machine (FSM) is a mathematical abstraction sometimes used to design digital logic. It has finite internal memory, an input feature and an output feature, which is in the form of a user interface, once the model is developed. The number of states and names of the states depends on the different possible states of the memory, e.g. if the memory is 3 bits long, there are 8 possible states i.e 2^k, where k equals to the number of bits . The state that reads the first symbol of the input sequence is called the start state and the state which signifies the successful operation of the machine is called the accept state.

Depending on the number of states an input can lead to, finite-state machines can be divided into two categories: deterministic and nondeterministic. The nondeterministic machine may have any number of following states for a given input symbol or none at all, whereas the deterministic machine has exactly one following state for a symbol. An important property of finite-state machines is that every nondeterministic machine has a corresponding deterministic one that accepts the same words. This is not true for pushdown automata. Finite-state machines can solve a large number of problems. The controller had two state machines, one for the processor side interface and one for the memory side interface, the memory side state machine had 16 states whereas the processor side interface had 4 states. The given below figures shows the different states of the execution.



Processor Side State Machine



Memory Side State Machine

Proposed Architecture:

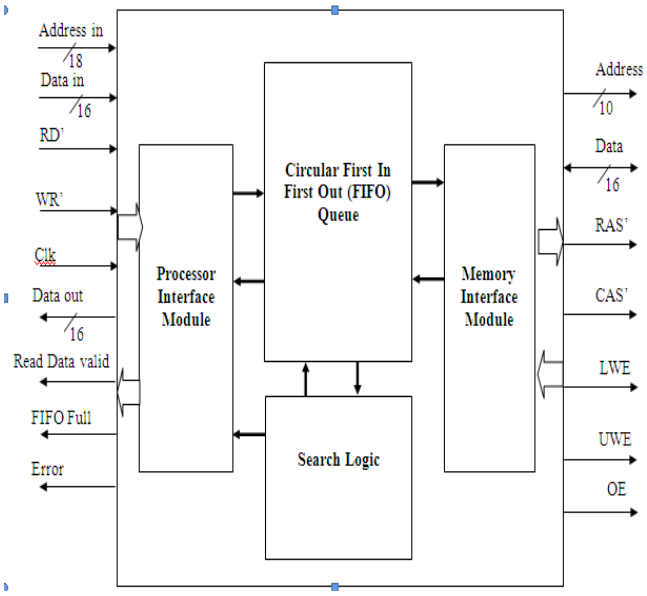
Existing controller have two different FIFO one to store the write requests and one to store the read requests. As soon as the request is serviced it

discards the corresponding data from the FIFO. But all processors follow the principle of locality which is described below. This paper tries to exploit this behavior of the processors. So we have proposed a method wherein 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, it is the phenomenon of the same value or related storage locations being frequently accessed. There are 2 basic types of principle of locality, they are - *Temporal locality* and *spatial locality*. Temporal locality means 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 the same location is referred for the next time. In this case it is advisable to store a copy of the referenced data in special small memory, which can be accessed faster. Spatial locality refers to the use of data elements within relatively close storage locations. 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 behavior in the controller which can be useful for some embedded processors which don't have in-built cache. This increases the overall efficiency of the controller.

If at one point in the time a particular memory location is referenced, 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 special small memory, which can be accessed faster.



Proposed Architecture

States and it's Descriptions:

Processor side interface:

Signal name	Description
Address	This is the multiplexed address bus to the memory, SIZE IS 13 BIT, ROW ADDRESS IS 10 BIT, and COLOUMN 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

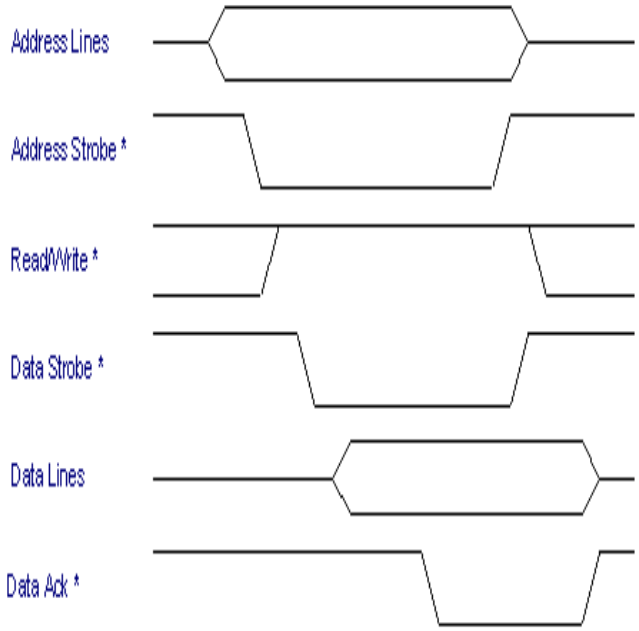
Memory Side Interface:

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 16 BIT
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

Signal (RD'):

This is a Read control signal (Active Low). This signal indicates that the selected I/O or memory device is to be read and data are available on the data bus.

Read bus cycle:

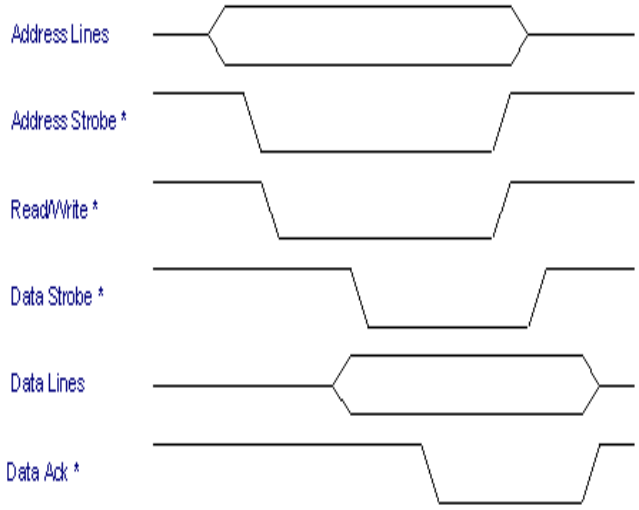


Read cycle

Write (WR'):

This is a write control signal (Active Low). This signal indicates that the data on the data bus are to be written into a selected memory or I/O location.

Write Bus Cycle:



Write Cycle

FIFO Full:

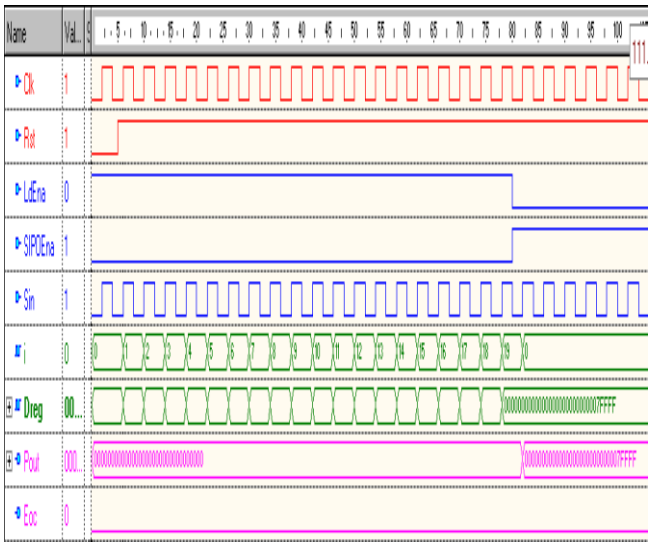
•When the write address register reaches the read address register, the FIFO triggers the FULL signal.

•When the read address LSBs equal the write address LSBs and the extra MSBs are different, the FIFO is full.

Simulation:

The purpose of simulation is to verify that the circuit works as desired. The Active-HDL simulator provides two simulation engines.

- Event-Driven Simulation
- Cycle-Based Simulation

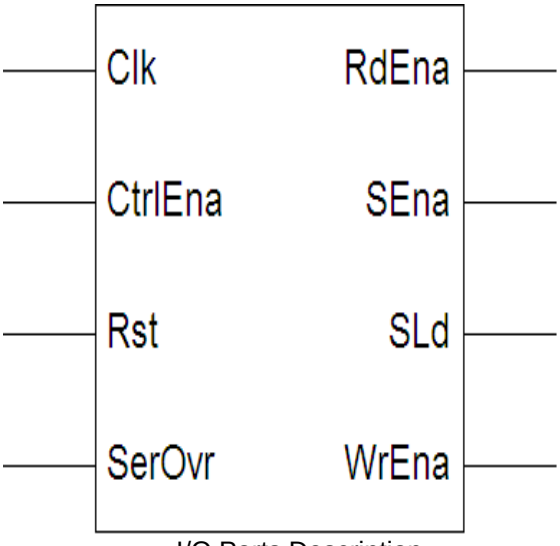


Simulation Result

MODULES:

RESET, and READY accept the externally initiated signals as inputs. The READY signal is used to delay the microprocessor READ or WRITE cycles until a slow-responding peripheral is ready to send or accept data. When this signal goes low, the microprocessor waits for an integral number of clock cycles until it goes high. Lastly, when the RESET IN signal goes low, the program counter is set to zero, the buses are tri-stated, and the MPU is reset and the RESET OUT signal indicates that the MPU is being reset and used to reset other devices.

To respond to the HOLD request, the 8085 Microprocessor has one signal, called HLDA (Hold Acknowledge). It acknowledges the HOLD request.



I/O Ports Description

STATE DESCRIPTION:

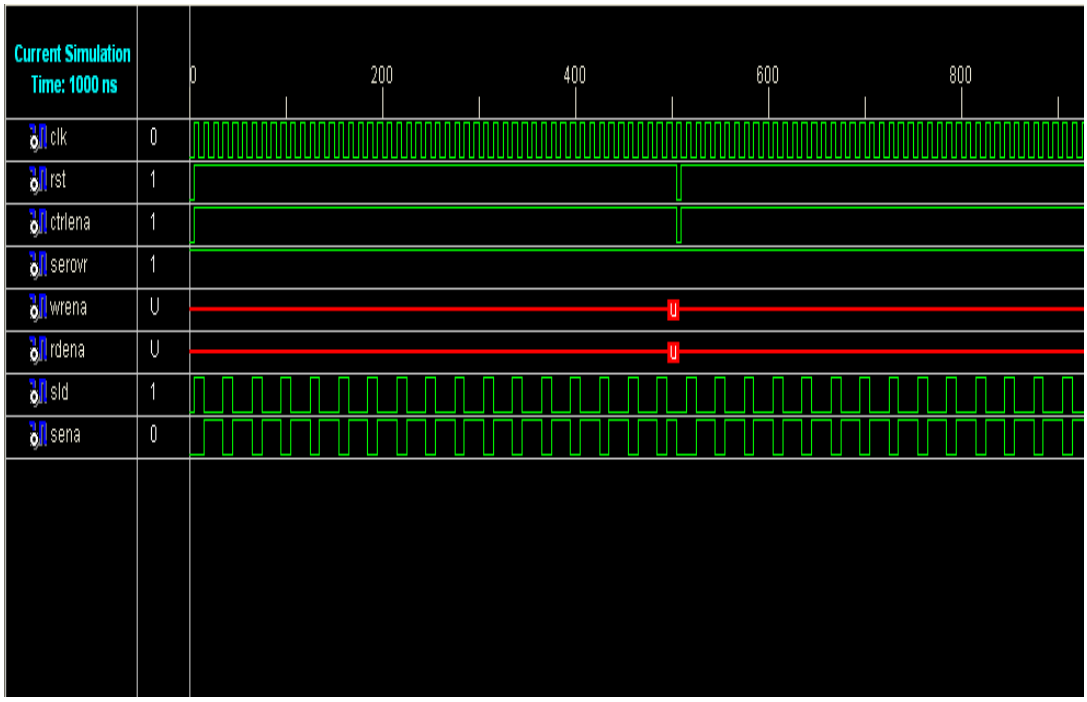
S.N0	Port Name	Mode	Size	Port Description
1	CLk	In	-	Entity function synchronized for Rising edge of the Clk
2	Rst	In	-	When asserted Entity is initiliased to default values
3	Ctrl Ena	In	-	Enables of smc
4	Serovr	In		1bit data which input to smc
5	WrEna,RdEna	Out	-	Output signals
6	SEna	Out	-	Output signal
7	Sld	Out	-	Output signal
8	Empty	Out	-	Output signal

Port Description

5.1.2 FUNCTIONAL DESCRIPTION:

State machine control controls all the modulesof architecture by clk,enable and rst signal on it.whens rst is '0' it is in idle state and for every rising edge of clock all modules are enableded and controls fifo,ram,ptos.

Simulation Result:

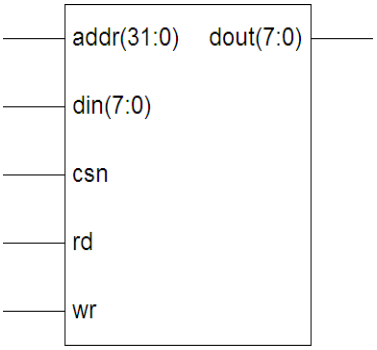


Simulation Behavior

RAM:

Functional Description:

Random-access memory (RAM) is a form of computer data storage. A random-access memory device allows data items to be read and written in roughly the same amount of time regardless of the order in which data items are accessed.



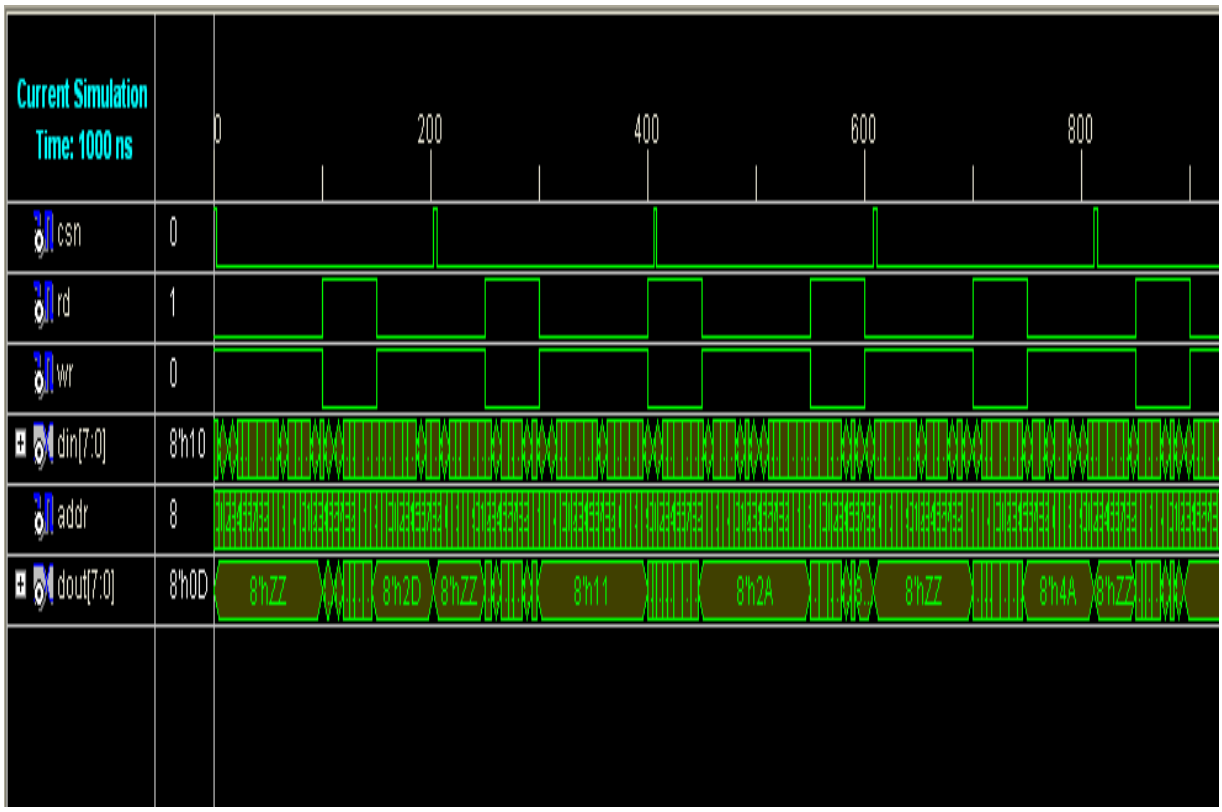
I/O Ports Description for RAM

State Description:

S.N0	Port Name	Mode	Size	Port Description
1	Addr	In	[31:0]	Address input signal
2	Csn	In	-	Chip selection input signal
3	Rd	In	-	Read request signal of memory
4	Din	In	[7:0]	8 bit data which input to top module
5	Wr	In	-	Write request signal of memory
6	Dout	Out	[7:0]	Output signal
7	Dout	Out	[7:0]	8 bit data which is output from FIFO

Port Description for RAM

Simulation Results:



Simulation Behavior for RAM

FIFO:

Functional Description:

FIFO [First in First Out] is a memory that stores information. An information can be either written into memory or read from memory. Here, we are using a 8X8 bit memory. The depth of the memory is 8 bit [i.e; number of locations] and width of memory is 4 bit. Let's, explain this with an example when, uses it according to her requirement. In the similar fashion the FIFO works, where we store incoming 8bit data in a memory, But to store data in a specific location we are using write pointer as well as read pointer. These are nothing but house address. The process consists of two parts, one is write logic and other is read logic. The process starts in this way, When reset is inserted then the internal register i.e, Wrptr or Rdptr is cleared. Then for every rising edge of clk whenever fifo enable is one, wrEna and rdEna are one and zero respectively, then we process the right logic. Where data is written into memory and wrptr is incremented parallel. When wrEna and rdEna are zero and one respectively, we read the data from memory to output (Dataout) using read pointer that specifies the location.But as memory is fixed, more and more data comes that leads to overlap of existing data. In order to avoid this problem we are using FULL and EMPTY signal, where full shows high when memory is full by which we stop writing the data to begin reading the data from memory. In the same way empty signal tells whether memory is empty or not, by which we begin reading the data from memory.

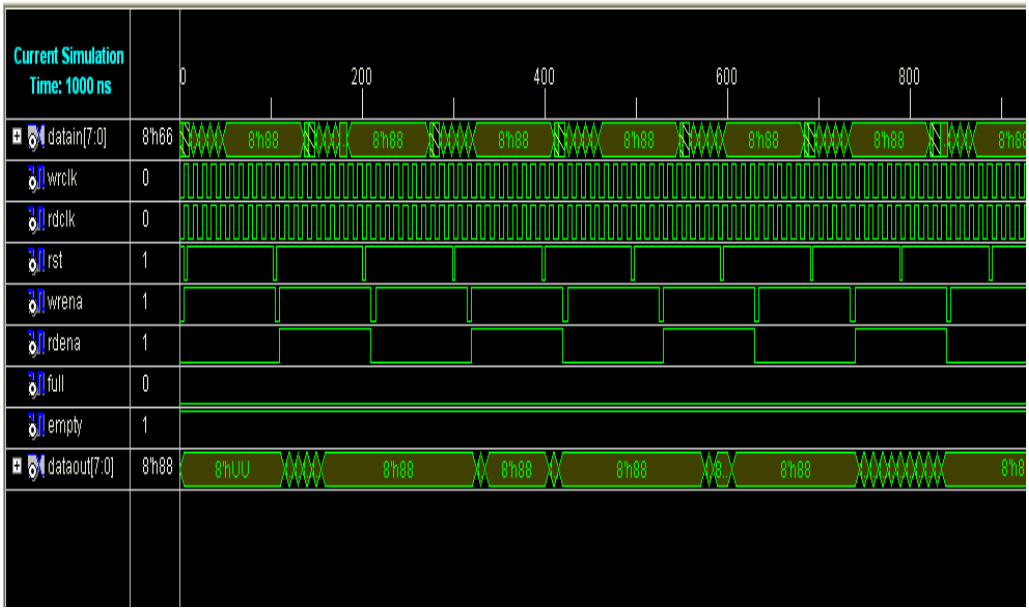
State Description:

S.N0	Port Name	Mode	Size	Port Description
1	WrClk,RdCLk	In	-	Entity function synchronized for Rising edge of the Write Clk and read Clk.
2	Rst	In	-	When asserted Entity is initiliased to default values
3	rdEna,wrEna	In	-	Enables of FIFO
4	Datain	In	[7:0]	8 bit data which input to FIFO
5	Dataout	Out	[7:0]	8 bit data which is output from FIFO
6	Full	Out	-	Signal Status of memory is full
7	Empty	Out	-	Signal Status of memory is ready to read.

State Description for FIFO

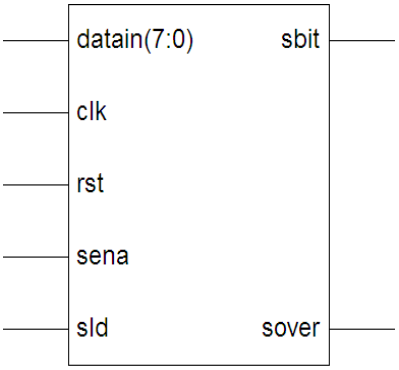
First In, First Out, FIFO is a method of processing data where the data first received is the first to be sent out after processed.

Simulation Results:



Simulation Results for FIFO

Seriliazer:



I/O Ports Description for Serializer

State Description:

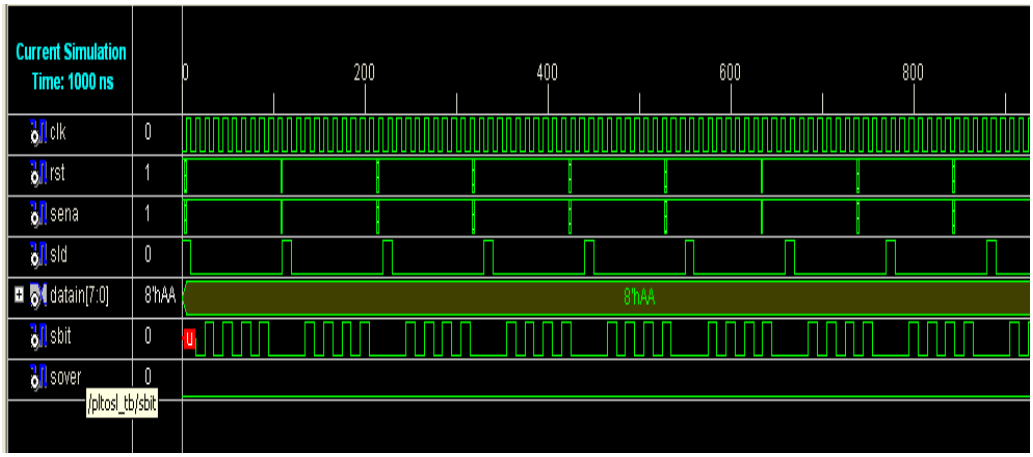
s.no	Port name	Mode	Size	Port description
1	SEna	Input	-	Input select signal
2	Clk	Input	1 bit	Input system clock signal
3	Datain	Input	[7:0]	Input data signal
4	Rst	Input	1 bit	On board reset signal
5	Sbit	Out	-	Output signal
6	SLd	Input	-	Input signal
7	SerOvr	Out	-	Output Signal

State Description for Serializer

Functional Description:

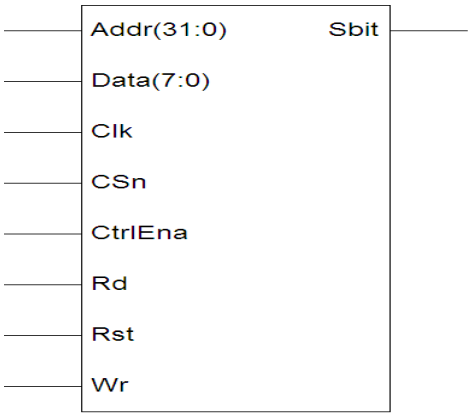
This is used to convert the data from parallel form to serial form. Here it is used to satisfy the basic requirement of the vlsi technology (i.e) utilization of less power and area with high speed. When bits are delivered to the system serially the power and area utilization reduces and speed of operation increases. Hence parallel to serial converter is incorporated in this project to deliver serialized data to the system .

Simulation Results:



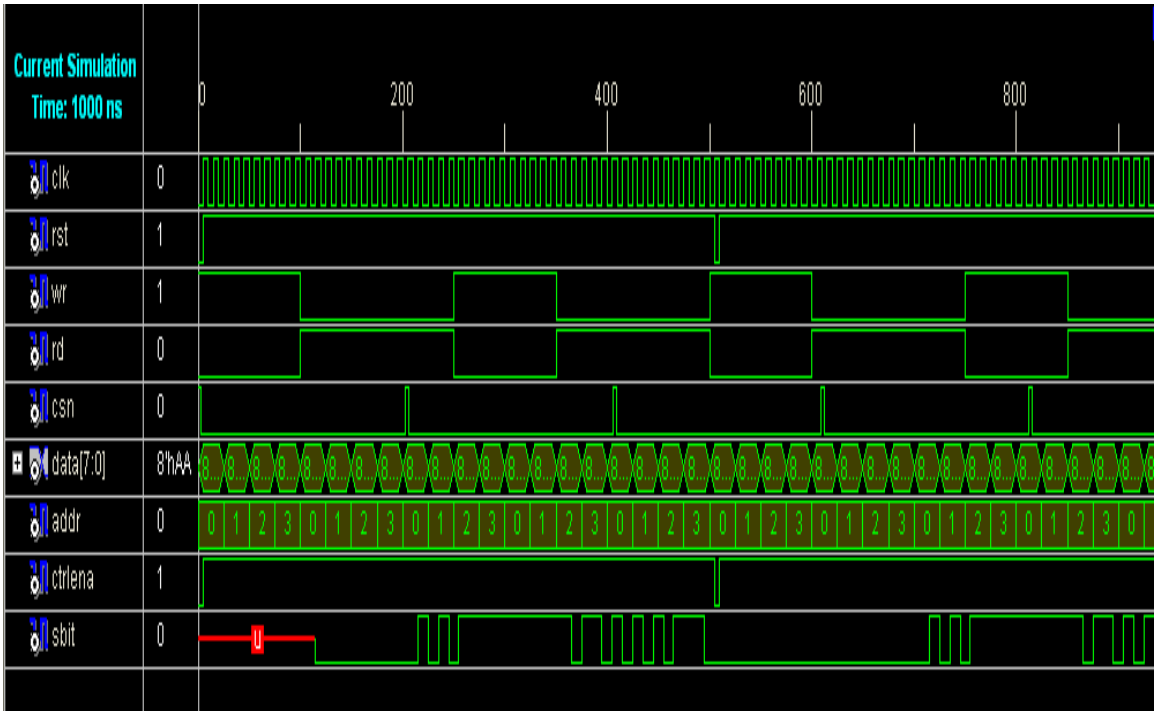
Simulation Result for Serializer

Top Module:



I/O port description

Simulation Results:



simulation result for final output

Synthesis Report:

==========
* Final Report *
==========
Final Results
RTL Top Level Output File Name: TOPMODULE.ngr
Top Level Output File Name : TOPMODULE
Output Format : NGC
Optimization Goal : Speed
Keep Hierarchy : NO
Design Statistics
IOs : 47
Cell Usage :
BELS : 589
GND `: 1
INV : 5
LUT1 : 62
LUT2 : 79
LUT3 : 83
LUT4 : 162
MUXCY :78
MUXF5 : 32
MUXF6 : 16
MUXF7 : 8
VCC : 1
XORCY : 62
FlipFlops/Latches : 228
FDC : 3
FDCE :79
FDE : 9
LD : 9
LDE :120
LDE_1 : 8

RAMS : 8
RAM16X1D :8
Clock Buffers : 1
BUFGP :1
IO Buffers : 17
IBUF : 16
OBUF : 1
===
Device utilization summary:

Selected Device : 3s100evq100-5
Number of Slices: 220 out of 960 22%
Number of Slice Flip Flops: 228 out of 1920 11%
Number of 4 input LUTs: 407 out of 1920 21%
Number used as logic: 391
Number used as RAMs: 16
Number of IOs: 47
Number of bonded IOBs: 18 out of 66 27%
Number of GCLKs: 1 out of 24 4%

Partition Resource Summary:

No Partitions were found in this design.

The simulation and synthesis is carried out on Xilinx ISE 8.2i, The synthesis has been successfully completed on vertex 5 XC5VLX30 with speed grade -1 produced following results.

Clock report: maximum combinational path delay or Minimum clock period allowed is 25.658 ns.

5.7 Device utilization summary:

Slice Logic Utilization	Used	Available
Number Of Slice Flip Flops	791	19,200
Number Of Slice LUT's	1,091	19,200
Number Used As Logic	1,088	19,200
Number Using 06 Output Only	1,088	
Number Used Exclusive Route-Thru	3	
Number Of Route –Thrus	3	38,400
Number Using 05 And 06	3	
Slice Logic Distribution		
Number Of LUT-Flop Pairs	1,207	19,200
IO Utilization		
Number Of Bonded IOB's	88	220
Other Utilization		

Table 5.5:Synthesis Report

CONCLUSION

We proposed a novel way i.e. cache like behavior 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 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.

Parameters	Existing Model	Proposed Model
Power Supply	1.8 ± 0.1 V	1.7 ± 0.1 V
Area	29mm	27mm
Delay	2.5ns	2.4ns
Gate Count	64 million transistor	68 million transistor

REFERENCES

1. B. Keeth and R. J. Baker, "*DRAM Circuit Design*", IEEE Press Series on Microelectronic Systems", New York, 2000
2. *"Jedec standard: double data rate (ddr) sdram specification"*, (revision of jesd79b), jedec solid state technology association, march 2003,
3. J. Hassoun, "*Virtex Synthesizable High Performance SDRAM Controller*", Xilinx (white paper),may 7,1999
4. K. Palanisamy and R. Chiu, "*High-Performance DDR2 SDRAM Interface in Virtex-5 Devices*", xilinx, may 8,2008
5. A. Cosoroaba, "*Memory Interfaces Made Easy with Xilinx FPGAs and the Memory Interface Generator*",xilinx,February 16, 2007
6. A. S. Sedra and K. C. Smith, "*Microelectronic Circuits*", Oxford Series in Electrical Engineering ,4th edition.