

**Contents** [ [hide](#) ]

- [1 XILINX 63234 END FPGA Distributor](#)
- [2 Introduction](#)
- [3 Stimulus File Description](#)
- [4 Documents / Resources](#)
  - [4.1 References](#)

**XILINX 63234 END FPGA Distributor**

**Important Note:** This downloadable PDF of an Answer Record is provided to enhance its usability and readability. It is important to note that Answer Records are Web-based content that is frequently updated as new information becomes available. You are reminded to visit the Xilinx Technical Support Website and review (Xilinx Answer 63234)

for the latest version of this Answer.

## Introduction

Because of the way DDR2 and DDR3 memories are architected and the MIG 7 series controller is designed, performance is not straightforward. It requires an understanding of various Jedec Timing parameters and controller Architecture, and you will need to run simulations to get the estimates. The general principle for determining performance is the same, but this document provides an easy way to obtain efficiency using the MIG example design with the help of the test bench and stimulus files attached here.

## Effective Bandwidth

The DRAM data bus achieves near peak bandwidth only during bursts of read and write, and its overhead lowers the effective data rate.

$$\text{Efficiency (\%)} = \frac{\text{Clock cycles transferring data}}{\text{Total clock cycles}}$$

$$\text{Effective Bandwidth} = \text{Peak Bandwidth} * \text{Efficiency}$$

## A few examples of overhead are

- precharge time accessing rows in the same bank (Access address not in the same row-page hit)
- write recovery time to change from write to read access
- bus turnaround time to change from read to write access

## Clock cycles transferring data

- Efficiency (%) = \_\_\_\_\_

## Total clock cycles

$$\text{Effective Bandwidth} = \text{Peak Bandwidth} * \text{Efficiency}$$

## MIG Design Generation

- Refer to UG586 Chapter 1 for step-by-step details on MIG IP and example design generation.
- Before running the MIG 7 Series performance simulation, do the following to make sure your simulation environment is fine.
- Open the MIG example design and map the appropriate libraries, run the simulation, and ensure that you can see the message “test passed” in the transcript.
- To demonstrate the flow, I have generated an MIG IP for xc7vx690tffg1761-2 and invoked the example design.
- Two things that should be noted are memory address bits and memory address mapping selection.
- For example, I have selected MT41J128M8XX-125 under the memory part drop-down options.

**Options for Controller 0 - DDR3 SDRAM**

**Clock Period:** Choose the clock period for the desired frequency. The allowed period range(1112 - 3300) is a function of the selected FPGA part and FPGA speed grade. Refer to the User Guide for more information. 1500 ps 666.67 MHz

**PHY to Controller Clock Ratio:** Select the PHY to Memory Controller clock ratio. The PHY operates at the Memory Clock Period chosen above. The controller operates at either 1/4 or 1/2 of the PHY rate. The selected Memory Clock Period will limit the choices. 4:1

**Vccaux\_io:** Vccaux\_io must be set to 2.0V in the High Performance banks for the highest data rates. Vccaux\_io is not available in the High Range banks. Note that Vccaux\_io is common to groups of banks. Consult the 7 Series Datasheets and FPGA SelectIO Resources User Guide for more information. 1.8V

**Memory Type:** Select the memory type. Type(s) marked with a warning symbol are not compatible with the frequency selection above. Components

**Memory Part:** Select the memory part. Part(s) marked with a warning symbol are not compatible with the frequency selection above. Find an equivalent part or create a part using the "Create Custom Part" button if the part needed is not listed here. The "Create Custom Part" feature is not supported for RLD RAM II. MT41J128M8XX-125

**Memory Voltage:** Select the Voltage of the Memory part selected. 1.5V

**Data Width:** Select the Data Width. Parts marked with a warning symbol are not compatible with the frequency and memory part selected above. 8

**ECC:** MIG supports ECC for 72 bit data width configuration. To be able to select ECC, select a data width that has ECC supported. Disabled

**Data Mask:** Enable or disable the generation of Data Mask (DM) pins using this check box. This option can be selectable only if the memory part selected has DM pins. Uncheck this box to not use data masks and save FPGA I/Os that are used for DM signals. ECC designs (DDR3 SDRAM, DDR2 SDRAM) will not use Data Mask. ☒

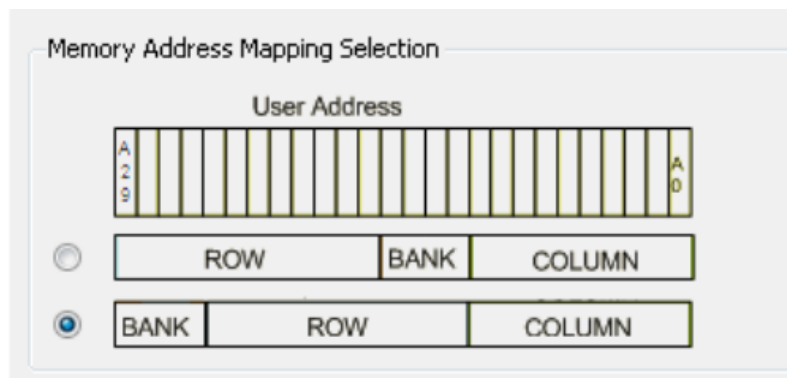
**ORDERING:** Normal mode allows the memory controller to reorder commands to the memory to obtain the highest possible efficiency. Strict mode forces the controller to execute commands in the exact order received. Normal

**Memory Details:** 1Gb, x8, row:14, col:10, bank:3, data bits per strobe:8, with data mask, single rank, 1.5V

Figure 1: Memory part details

For the selected memory part from Figure-1, row = 14, column = 10 and bank = 3, so  

$$\text{app\_addr\_width} = \text{row} + \text{column} + \text{bank} + \text{rank} = 28$$

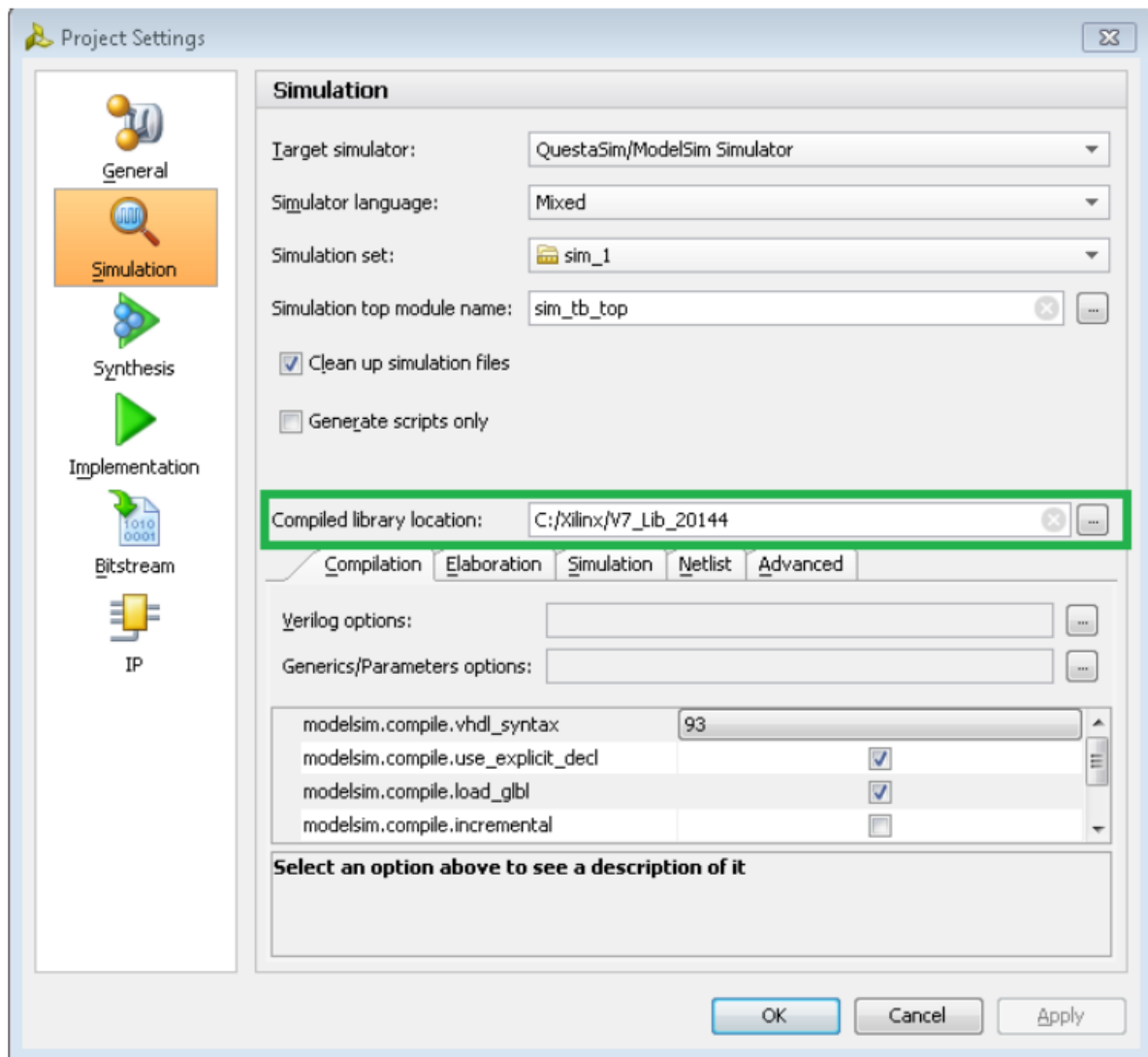


*Figure 2: Memory Address mapping*

You can select either BANK\_ROW\_COLUMN or ROW BANK\_COLUMN.  
I have left the ROW BANK Column, which is the default address mapping.

### **Example design Simulation with synthesizable test bench**

- Under Simulation settings, select QuestaSim/ModelSim Simulator and browse to the compiled libraries location.
- For details on pointing to the third-party tools install path, selecting the target simulator, and compiling and mapping libraries, you can refer to (UG900) Vivado Design Suite User Guide Logic Simulation.



**Figure 3: Simulation Settings**

Simulate the GUI (Click the Run Simulation Tab in the project manager) and make sure you see the “test passed” message in the transcript.

### **Performance Simulation RTL modifications**

1. Right click the sources tab, select “add or create simulation sources”, browse to the mig7\_perfsim\_traffic\_generator.sv file and click finish to add it.
2. Right click the sources tab, select “add or create simulation sources”, browse to perfsim\_stimulus.txt, and click finish adding it.
3. Comment out the example\_top instantiation in the sim\_tb\_top.v file.
4. Add the below RTL lines to sim\_tb\_top,v

```

localparam APP_ADDR_WIDTH = 28;
localparam APP_DATA_WIDTH = 64;
localparam APP_MASK_WIDTH = APP_DATA_WIDTH / 8;
localparam MEM_ADDR_ORDER = "BANK_ROW_COLUMN";
localparam BANK_WIDTH = 3;
localparam RANK_WIDTH = 1;

```

```

wire [APP_ADDR_WIDTH-1:0] c0_ddr3_app_addr;
wire [2:0] c0_ddr3_app_cmd;
wire c0_ddr3_app_en;
wire [APP_DATA_WIDTH-1:0] c0_ddr3_app_wdf_data;
wire c0_ddr3_app_wdf_end;
wire [APP_MASK_WIDTH-1:0] c0_ddr3_app_wdf_mask;
wire c0_ddr3_app_wdf_wren;
wire [APP_DATA_WIDTH-1:0] c0_ddr3_app_rd_data;
wire c0_ddr3_app_rd_data_end;
wire c0_ddr3_app_rd_data_valid;
wire c0_ddr3_app_rdy;
wire c0_ddr3_app_wdf_rdy;
wire c0_data_compare_error;
wire ui_clk;
wire ui_clk_sync_rst;
wire app_sr_req = 0;
wire app_ref_req = 0;
wire app_zq_req = 0;
wire c0_app_wdf_mask = 0;

```

```

//=====
//          FPGA Memory Controller instantiation
//=====

```

```

mig_7series_0_mig u_mig_7series_0_mig (

```

```

// Memory interface ports
.ddr3_addr      (ddr3_addr_fpga),
.ddr3_ba        (ddr3_ba_fpga),
.ddr3_cas_n     (ddr3_cas_n_fpga),
.ddr3_ck_n      (ddr3_ck_n_fpga),
.ddr3_ck_p      (ddr3_ck_p_fpga),
.ddr3_cke       (ddr3_cke_fpga),
.ddr3_ras_n     (ddr3_ras_n_fpga),
.ddr3_reset_n   (ddr3_reset_n),
.ddr3_we_n      (ddr3_we_n_fpga),
.ddr3_dq        (ddr3_dq_fpga),

```

```

        .ddr3_dqs_n      (ddr3_dqs_n_fpga),
        .ddr3_dqs_p      (ddr3_dqs_p_fpga),
        .init_calib_complete (init_calib_complete),

        .ddr3_cs_n      (ddr3_cs_n_fpga),
        .ddr3_dm         (ddr3_dm_fpga),
        .ddr3_odt        (ddr3_odt_fpga),

        // Application interface ports
        .app_addr        (c0_ddr3_app_addr),
        .app_cmd          (c0_ddr3_app_cmd),
        .app_en           (c0_ddr3_app_en),
        .app_wdf_data     (c0_ddr3_app_wdf_data),
        .app_wdf_end      (c0_ddr3_app_wdf_end),
        .app_wdf_wren     (c0_ddr3_app_wdf_wren),
        .app_rd_data      (c0_ddr3_app_rd_data),
        .app_rd_data_end  (app_rd_data_end),
        .app_rd_data_valid (c0_ddr3_app_rd_data_valid),
        .app_rdy          (c0_ddr3_app_rdy),
        .app_wdf_rdy      (c0_ddr3_app_wdf_rdy),
        .app_sr_req       (app_sr_req),
        .app_ref_req      (app_ref_req),
        .app_zq_req       (app_zq_req),
        .app_sr_active    (app_sr_active),
        .app_ref_ack      (app_ref_ack),
        .app_zq_ack       (app_zq_ack),
        .ui_clk           (ui_clk),
        .ui_clk_sync_rst  (ui_clk_sync_rst),
        .app_wdf_mask     (c0_ddr3_app_wdf_mask),
        // System Clock Ports
        .sys_clk_i        (sys_clk_i),
        // Reference Clock Ports
        .clk_ref_i        (clk_ref_i),
        .sys_rst          (sys_rst)
    );

```

```

//=====
//                               Performance traffic generator instantiation
//=====

```

```

mig7_perfsim_traffic_generator#
(
    .APP_DATA_WIDTH      (APP_DATA_WIDTH),
    .COL_WIDTH           (COL_WIDTH),
    .ROW_WIDTH           (ROW_WIDTH),
    .RANK_WIDTH          (RANK_WIDTH),
    .BANK_WIDTH          (BANK_WIDTH),
    .MEM_ADDR_ORDER      (MEM_ADDR_ORDER),
    .tCK                 (tCK ),
    .ADDR_WIDTH          (APP_ADDR_WIDTH)
)

```

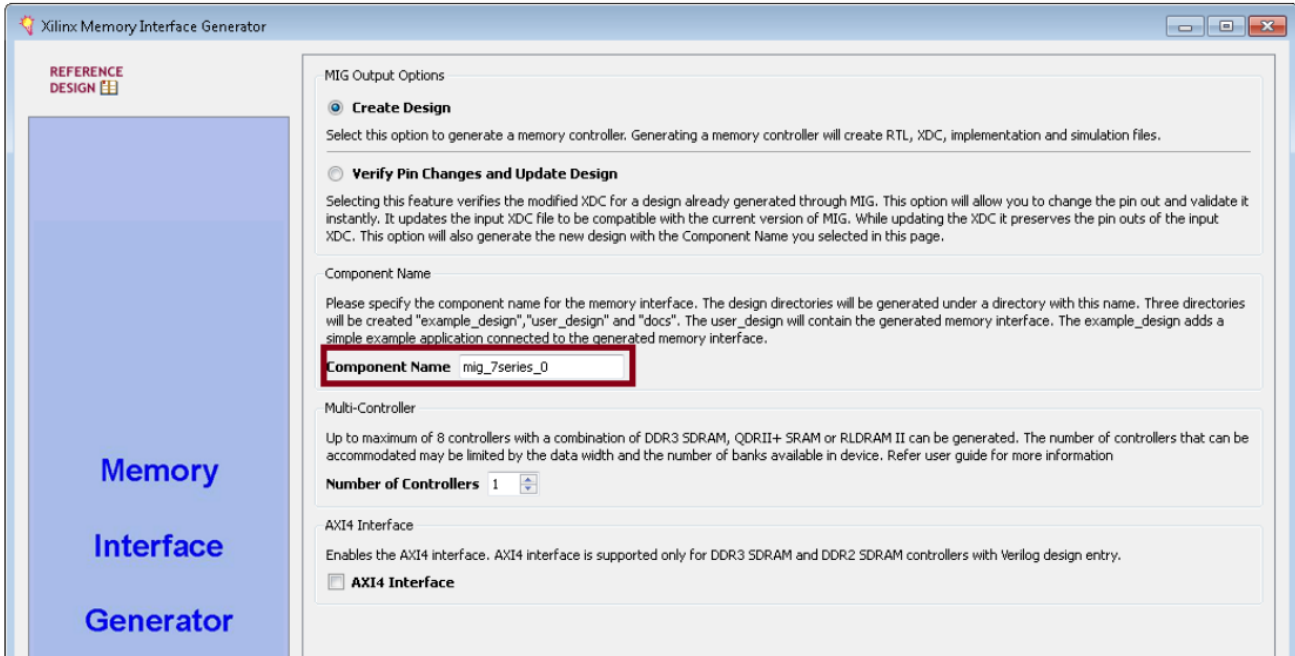


```

u_traffic_gen
(
    .clk                (ui_clk ),
    .rst                (ui_clk_sync_rst ),
    .init_calib_complete (init_calib_complete),
    .cmp_error          (c0_data_compare_error),
    .app_wdf_rdy        (c0_ddr3_app_wdf_rdy ),
    .app_rd_data_valid  (c0_ddr3_app_rd_data_valid),
    .app_rd_data        (c0_ddr3_app_rd_data ),
    .app_rdy            (c0_ddr3_app_rdy),
    .app_cmd            (c0_ddr3_app_cmd  ),
    .app_addr           (c0_ddr3_app_addr ),
    .app_en             (c0_ddr3_app_en  ),
    .app_wdf_mask       (c0_ddr3_app_wdf_mask),
    .app_wdf_data       (c0_ddr3_app_wdf_data),
    .app_wdf_end        (c0_ddr3_app_wdf_end ),
    .app_wdf_wren       (c0_ddr3_app_wdf_wren)
);

```

5. Modify APP\_ADDR\_WIDTH, APP\_DATA\_WIDTH, RANK\_WIDTH, H, and BANK\_WIDTH according to your memory part selection. Values can be obtained from the \_mig.v file.
6. The yellow highlighted instantiation name mig\_7series\_0\_mig can vary based on your component name during IP creation. Verify if you have chosen a different name and change it accordingly.



**Figure 4: MIG Component Name**

7. Once the IP is generated open the \_mig.v file and cross check for any variations in LHS signal names and correct them.
8. app\_sr\_req, app\_ref\_req, and app\_zq\_req should be initialized to 0.
9. As example\_top.v is commented out and new files are added, you will probably see “?” beside the mig\_7series\_0\_mig.v file under simulation sources.



10. To map the correct file, right-click mig\_7series\_0\_mig.v, select “Add Sources”, browse to  
 <Project.directory>/mig\_7series\_0\_example.srscs/sources\_1/ip/mig\_7series\_0/mig\_7serie  
 and add the mig\_7series\_0\_mig\_sim.v file.
11. If you see “?” for the underlying files, add all RTL files in the clocking, controller,  
 ip\_top,phy, and UI folders.

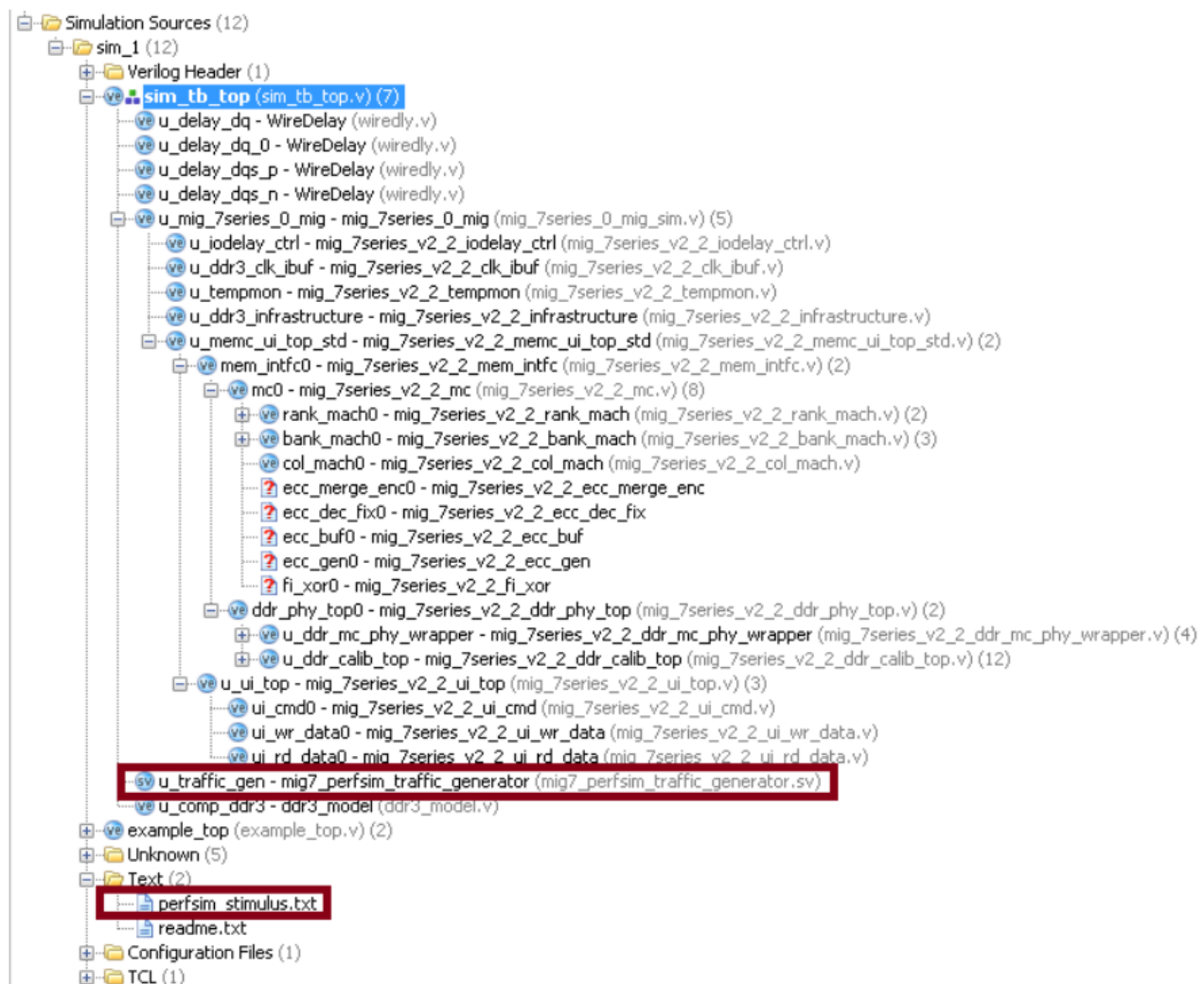


Figure 5: Simulation Hierarchy

12. Once the RTL changes are done and all of the required files are added to your Simulation Sources, the Hierarchy should be similar to Figure 5.
13. The files highlighted in red are newly added, and “?” is expected on ECC-related modules as the selected memory configuration has the ECC option disabled.

## Stimulus File Description

Each stimulus pattern is 48 bits, and the format is described in Figures 6-1 through 6-4.

Command Repeat[47:40]	Address [39:4]	Command[3:0]
-----------------------	----------------	--------------

**Figure 6-1: Stimulus pattern bit mapping**

Signal	Description
Command[3:0]	This corresponds to the WRITE/READ/NOP command that is sent to the user interface.
Address[35:0]	This corresponds to the address to the user interface.
Command Repeat[7:0]	This corresponds to the repetition count of the command. Up to 128 repetitions can be made for a command. In the burst length of eight mode, 128 transactions fill up the page in the memory.

**Figure 6-2: Stimulus pattern description**

Command	Code	Description
WRITE	0	This corresponds to the Write operation that needs to be performed.
READ	1	This corresponds to the Read operation that needs to be performed.
NOP	7	This corresponds to the idle situation for the bus.

**Figure 6-3: Stimulus Command description**

### Address Encoding (Address [35:0])

The address is encoded in the stimulus as per Figure 7-1 to Figure 7-6. All of the address fields need to be entered in the hexadecimal format.

All of the address fields are a width that is divisible by four to enter in the hexadecimal format. The test bench only sends the required bits of an address field to the Memory Controller. For example, in an eight bank configuration, only bank Bits [2:0] are sent to the Memory Controller, and the remaining bits are ignored. The extra bits for an address field are provided for you to enter the address in a hexadecimal format. You must confirm that the value entered corresponds to the width of a given configuration.

Rank[3:0]	Bank[3:0]	Row[15:0]	Column[11:0]
-----------	-----------	-----------	--------------

**Figure 6-4: Stimulus Address Encoding**

- Column Address (Column[11:0]) – Column Address in the stimulus is provided to a maximum of 12 bits, but you need to address this based on the column width parameter set in your design.

- Row Address (Row[15:0]) – Row address in the stimulus is provided to a maximum of 16 bits, but you need to address
- This is based on the row width parameter set in your design.
- Bank Address (Bank[3:0]) – The Bank address in the stimulus is provided to a maximum of four bits, but you need to address this based on the bank width parameter set in your design.
- Rank Address (Rank[3:0]) – Rank address in the stimulus is provided to a maximum of four bits, but you need to address this based on the rank width parameter set in your design.
- The address is assembled based on the top-level MEM\_ADDR\_ORDER parameter and sent to the user interface.

### **Command Repeat (Command Repeat [7:0])**

- The command repetition count is the number of times the respective command is repeated at the User Interface. The address for each repetition is incremented by 8. The maximum repetition count is 128.
- The test bench does not check for the column boundary, and it wraps around if the maximum column limit is reached during the increments.
- The 128 Commands fill up the page. For any column address other than 0, the repetition count of 128 ends up crossing.
- The column boundary wraps around to the start of the column address.

### **Bus Utilization**

The bus utilization is calculated at the User Interface, taking the total number of Reads and writes into consideration, and the following equation is used:

$$bw\_cumulative = \frac{((rd\_command\_cnt + wr\_command\_cnt) \times (BURST\_LEN / 2) \times 100)}{((end\_of\_stimulus - calib\_done) / tCK);}$$

- BL8 takes four memory clock cycles
- End\_of\_stimulus is the time when all the commands are done.
- calib\_done is the time when the calibration is done.

## Example Patterns

These examples are based on the MEM\_ADDR\_ORDER set to BANK\_ROW\_COLUMN.

### Single Read Pattern

00\_0\_2\_000F\_00A\_1 – This pattern is a single read from 10th column, 15th row, and second bank.

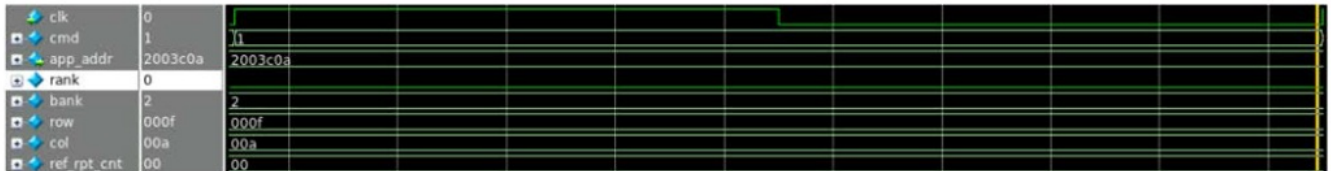


Figure 7-1: Single Read Pattern

### Single Write Pattern

00\_0\_1\_0040\_010\_0 – This pattern is a single write to the 32nd column, 128th row, and first bank.

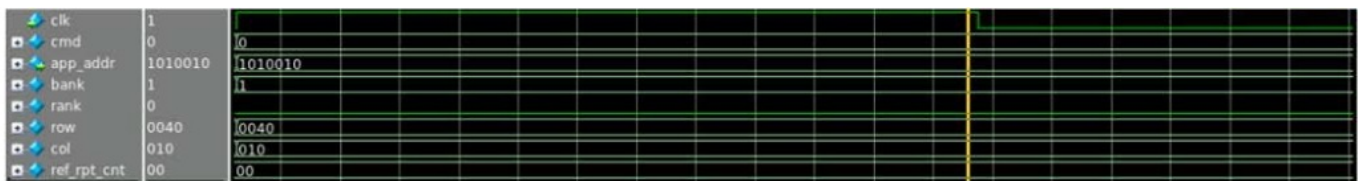


Figure 7-2: Single Write Pattern

### Single Write and Read to Same Address

- 00\_0\_2\_000F\_00A\_0 – This pattern is a single write to 10th column, 15th row, and second bank.
- 00\_0\_2\_000F\_00A\_1 – This pattern is a single read from 10th column, 15th row, and second bank

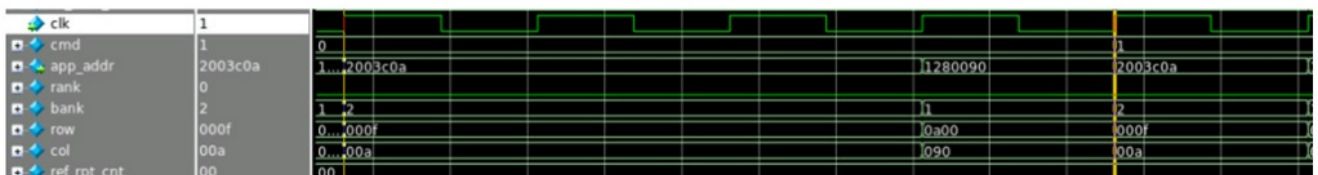


Figure 7-3: Single Write and Read to Same Address

### Multiple Writes and Reads with Same Address

- 0A\_0\_0\_0010\_000\_0 – This corresponds to 10 writes with addresses starting from 0

to 80, which can be seen in the column.

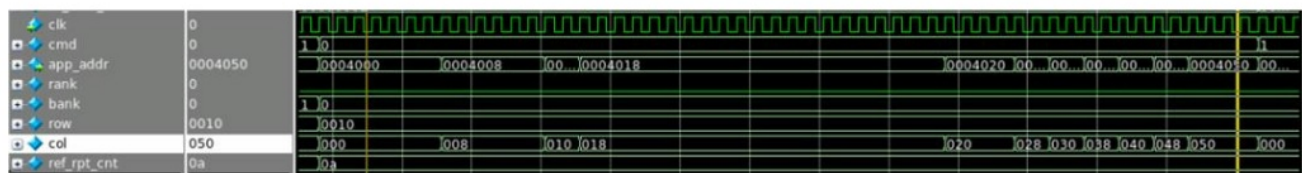


Figure 7-4: Multiple Writes with Same Address

- 0A\_0\_0\_0010\_000\_1 – This corresponds to 10 reads with address starting from 0 to 8,0, which can be seen in the column.

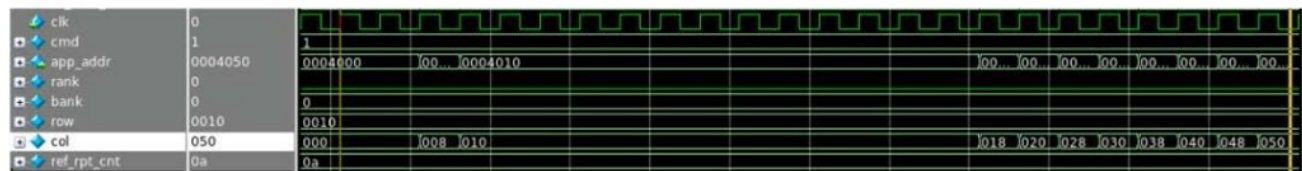


Figure 7-5: Multiple Reads with Same Address

## Page Wrap During Writes

0A\_0\_2\_000F\_3F8\_0 – This corresponds to 10 writes with column address wrapped to the start of the page after one write.

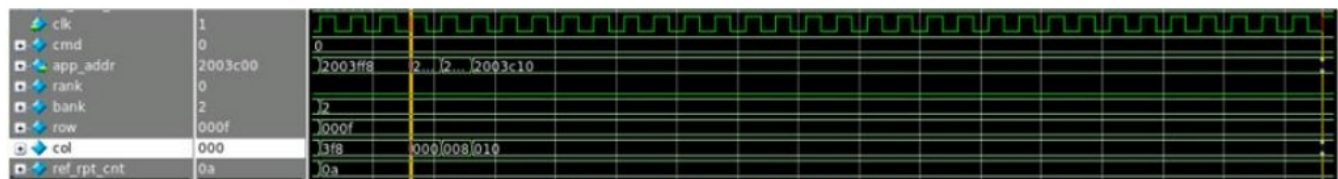


Figure 7-6: Page Wrap during Writes

## Simulating the Performance Traffic Generator

At this point, you are done with the MIG example design simulation. This implies that your simulation setup is ready, you have done performance simulation RTL modifications, the new simulation hierarchy is correct, and you have understood the stimulus patterns. Run the simulation once again with 16 writes and reads in perfsim\_stimulus.txt.

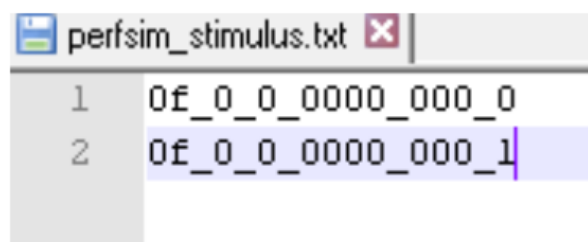
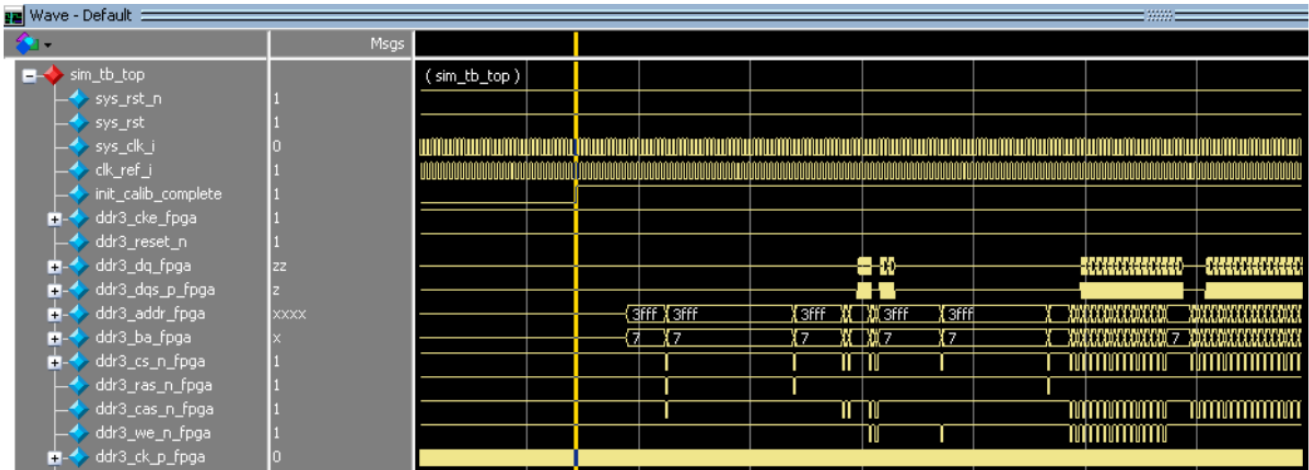


Figure 8-1: Perfsim\_stimulus for 16 writes and read

- Do run all, wait until the `init_calib_complete` signal is asserted, and you will be able to see the proposed number of writes and reads. The simulation will then stop.



**Figure 8-2: Performance Simulation DDR3 phy signals for 16 writes and reads**

- When you are prompted to quit the simulation, select No and go to the transcript window, where you will be able to see the performance statistics.

```

#####
# Performance Simulation Completed Successfully!!!
# Data integrity check is not performed in performance simulation
#####
#
# *****
#                               PERFORMANCE STATISTICS
# *****
#      Number of Writes          : 16
#      Number of Reads           : 16
#      Calibration is done at    : 56444000.0 ps
#      Transactions ended at     : 57094000.0 ps
#      Percentage of Bus utilization : 29.000000
#
#
#
# ** Note: $finish      : ../../../../mig_7series_0_example.srscs/sim_1/imports/Perf_files/mig7_perfsim_traffic_generator.sv(565)
#      Time: 57094 ns  Iteration: 0  Instance: /sim tb top/u traffic gen

```

**Figure 8-3: Performance Statistics for 16 writes and reads**

- If you select “quit simulation,” performance statistics will be written to a file named `mig_band_width_output.txt` located in the `sim_1/behav` folder.
- Example directory path:-  
`/mig_7series_0_example_perf_sim\mig_7series_0_example.sim/sim_1/behav`



Name	Date modified	Type	Size
msim	1/6/2015 12:39 PM	File folder	
work	1/6/2015 12:39 PM	File folder	
compile.bat	1/6/2015 12:38 PM	Windows Batch File	1 KB
compile.log	1/6/2015 12:39 PM	Text Document	25 KB
elaborate.bat	1/6/2015 12:39 PM	Windows Batch File	1 KB
elaborate.log	1/6/2015 12:40 PM	Text Document	14 KB
gbl.v	11/28/2014 9:31 AM	V File	2 KB
ies_run.sh	11/25/2014 11:50 ...	SH File	6 KB
mig_band_width_output.txt	1/6/2015 12:44 PM	Text Document	1 KB
mig7_perfsim_traffic_generator.sv	1/6/2015 10:04 AM	SV File	29 KB
modelsim.ini	1/6/2015 12:39 PM	Configuration sett...	75 KB
perfsim_stimulus.txt	1/6/2015 12:38 PM	Text Document	1 KB
readme.txt	11/25/2014 11:50 ...	Text Document	9 KB
sim.do	11/25/2014 11:50 ...	DO File	7 KB

**Figure 8-4: Performance Simulation output file**

You might wonder why the percentage of bus utilization is only 29. Rerun the simulation with the same IP settings, but just changing the stimulus file to 256 writes and 256 reads

- ff\_0\_0\_0000\_000\_0
- ff\_0\_0\_0000\_000\_1

You will now see the percentage as 85, which implies that DDR3 offers better bus utilization for a long sequence of writes and read bursts.

```
# *****
#                               PERFORMANCE STATISTICS
# *****
#      Number of Writes           : 256
#      Number of Reads            : 256
#      Calibration is done at      : 56444000.0 ps
#      Transactions ended at      : 60028000.0 ps
#      Percentage of Bus utilization : 85.000000
#
```

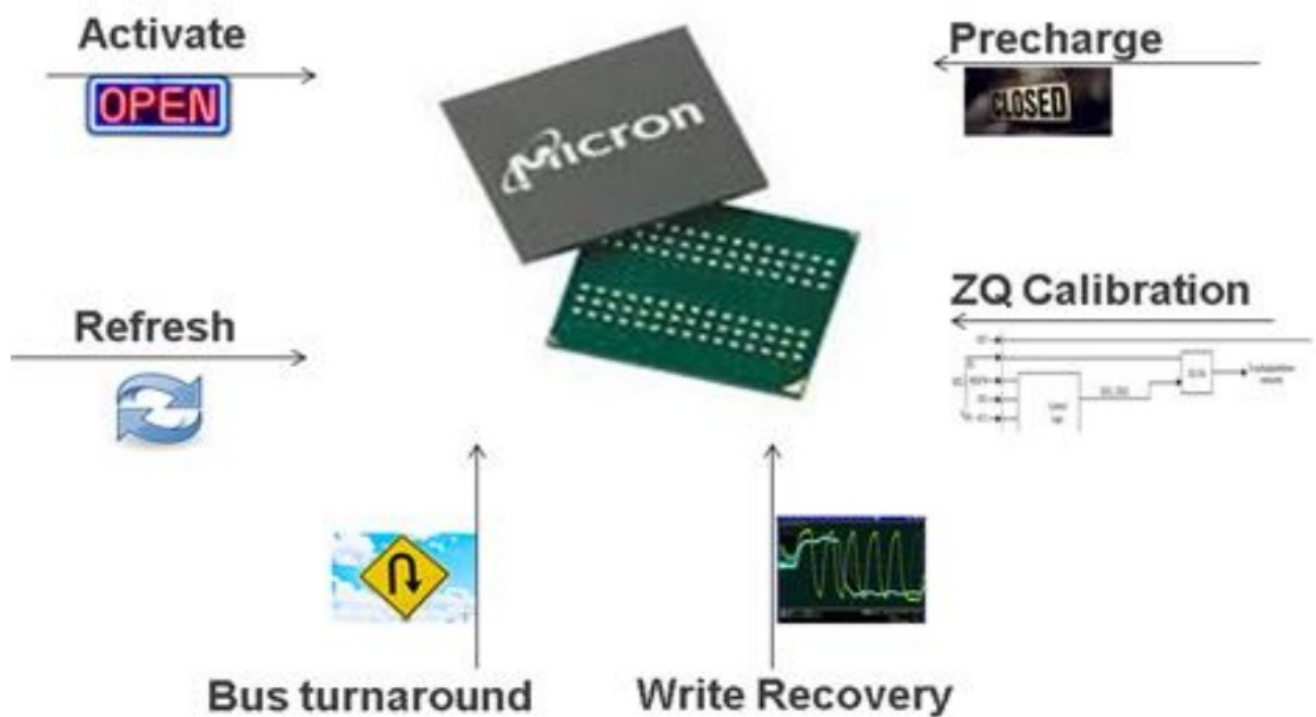
**Figure 8-5: Performance Statistics for 256 writes and reads**

## General ways to improve Performance

The factors that influence efficiency can be divided into two sections:

### 1. Memory Specific





**Figure 9: Efficiency factor –Memory Specific**

## 2. Controller Specific

Figure 9 gives you an overview of the terms that are memory-specific.

Unlike SRAMs and Block Memories, DDR2 or DDR3 performance is not just the maximum data rate.

It depends on many timing factors, including:

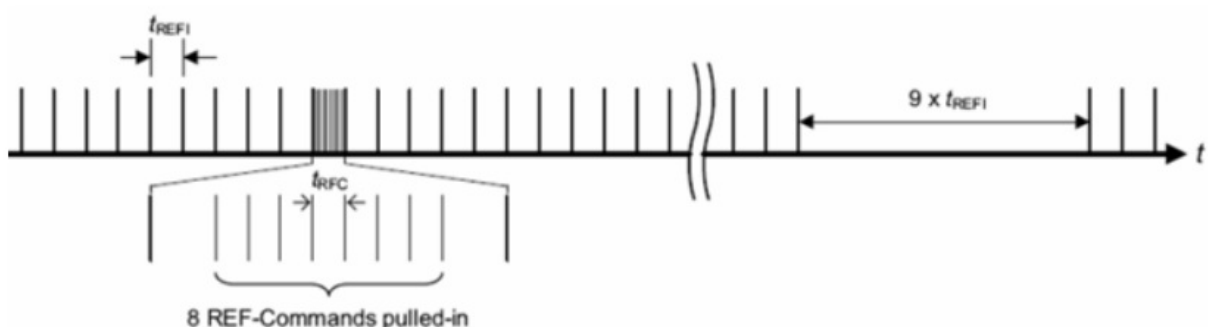
- tRCD: Row Command Delay (or ras to cas delay).
- tCAS(CL): Column address strobe latency.
- tRP: Row precharge delay.
- tRAS: Row Active Time (activate to prechange).
- tRC: Row cycle time.  $tRC = tRAS + tRP$
- tRAC: Random access delay.  $tRAC = tRCD + tCAS$
- tCWLCASas write latency.
- tZQ: ZQ calibration time.
- tRFC: Row Refresh Cycle Time
- tWTR: Write to Read delay. Last write transaction to Read command time.
- tWR: Write Recovery time. Last write transaction to Precharge time

- The timing of all the listed parameters depends on the type of memory used and the memory part's speed grade.
- More details on the definitions and timing specifications can be found in the DDR2 and DDR3 JEDEC standards or any memory device datasheet.

Efficiency mainly depends on how memory is accessed. Different address patterns give different efficiency results.

## Memory timing overheads

1. Activation time and Precharge time when changing to new banks/rows or changing rows within the same bank.- So, you reduce row change, which can remove  $t_{RCD}$  and  $t_{RP}$ .
2. Send continuous write or read commands -Maintaining  $t_{CCD}$  timing.
3. Minimize write-to-read and read-to-write command changeover – Write recovery time to change to read accesses, and bus turnaround time to change from read to write.
4. Set a proper refresh interval.
  - a. DDR3 SDRAM requires Refresh cycles at an average periodic interval of  $t_{REFI}$ .
  - b. A maximum of 8 additional Refresh commands can be issued in advance (“pulled in”). This does not reduce the number of refreshes, but the maximum interval between two surrounding Refresh commands is limited to  $9 \times t_{REFI}$



*Figure 10: Refresh pulls in*

5. Utilize all of the banks – A suitable addressing mechanism is preferable.
  - a. Row-Bank-Column: For a transaction occurring over a sequential address space, the core automatically opens up the same row in the next bank of the DRAM device to continue the transaction when the end of an existing row is reached. It is well suited to applications that require bursting of large data packets

to sequential address locations.

- b. Bank-Row-Column: When crossing a row boundary, the current row will be closed, and another row will be opened within the same bank. MSB is a bank address that can be used to switch from different banks. It is suitable for shorter, more random transactions to one block of memory for some time, and then a jump to another block (bank)

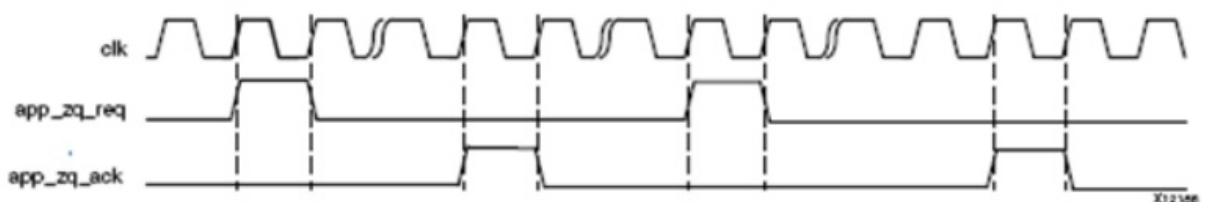
## 6. Burst Length

- a. BL 8 is supported for DDR3 on the 7 series. BC4 has a very low efficiency, which is less than 50%. This is because the execution time of BC4 is the same as BL8. The data is just masked inside the component.
- b. In cases where you do not wish to write a full burst, either data mask or write-after-read can be considered.

## 7. Set a proper ZQ interval (DDR3 Only)

The controller sends both ZQ Short (ZQCS) and ZQ Long (ZQCL) Calibration commands.

- a. Adhere to the DDR3 JEDEC Standard
- b. ZQ Calibration is discussed in section 5.5 of the JEDEC Spec JESD79-3 DDR3 SDRAM Standard
- c. ZQ Calibration calibrates On-Die Termination (ODT) at regular intervals to account for variations across VT
- d. Logic is contained in bank\_common.v/vhd
- e. Parameter Tzqcs determines the rate at which a ZQ Calibration command is sent to the memory
- f. It is possible to disable the counter and manually send using app\_zq\_req, it is similar to manually sending a Refresh. Refer to (Xilinx Answer 47924) for details.

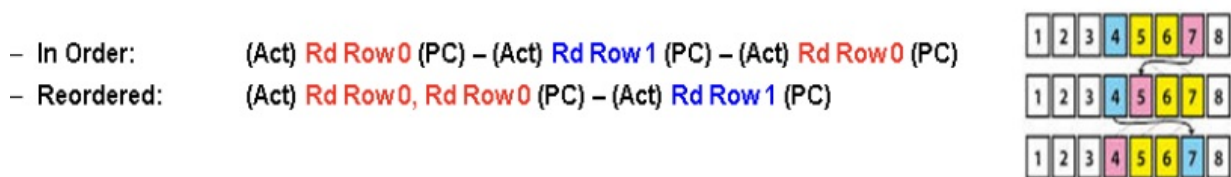


**Figure 11: Manual ZQ calibration**

## Controller Overheads

1. Periodic Reads – Refer to (Xilinx Answer 43344) for details.

- a. Do not change the period of the read.
  - b. Skip periodic reads during writes and issue the number of missed reads before a true read
2. Reordering – Refer to (Xilinx Answer 34392) for details. For User and AXI Interface designs, it is preferable to have this enabled.
- a. Reorder is the logic that looks ahead several commands and changes the user command order to make nonmemory commands not occupy valid bandwidth. The performance also related to actual traffic pattern.
  - b. Based on the address pattern, reorder helps to skip precharge and activate commands and makes tRCD and tRP do not occupy the data bandwidth.



**Figure 11: Reordering example**

3. Try to increase the number of Bank Machines.
- a. Most of the controller's logic resides in the bank machines, and they correspond to DRAM banks
  - b. A given bank machine manages a single DRAM bank at any given time.
  - c. Bank machine assignment is dynamic, so it is not necessary to have a bank machine for each physical bank.
  - d. Bank machines can be configured, but it is a tradeoff between area and performance.
  - e. The allowable number of bank machines ranges from 2-8.
  - f. By default, 4 Bank Machines are configured through RTL parameters.
  - g. To change Bank Machines, consider the parameter `nBANK_MACHS = 8` contained in `memc_ui_top`

### **Example for 8 Bank Machines – nBANK\_MACHS = 8**

You are now aware of the factors that influence performance. Consider an upstream application that gives you 512 data bytes per packet, and you need to save them to different memory locations. As 512 data bytes is equal to 64 DDR3 data bursts, re-run the example design with a stimulus file containing 512 writes, 512 reads, and row switching for every 64 writes or reads:

```

3f_0_0_0000_000_0
3f_0_0_0001_000_0
3f_0_0_0002_000_0
3f_0_0_0003_000_0
3f_0_0_0004_000_0
3f_0_0_0005_000_0
3f_0_0_0006_000_0
3f_0_0_0007_000_0
3f_0_0_0000_000_1
3f_0_0_0001_000_1
3f_0_0_0002_000_1
3f_0_0_0003_000_1
3f_0_0_0004_000_1
3f_0_0_0005_000_1
3f_0_0_0006_000_1
3f_0_0_0007_000_1

```

At the end of the simulation, you will see that bus utilization is at 77 percent.

```

*****
PERFORMANCE STATISTICS
*****
Number of Writes           : 512
Number of Reads            : 512
Calibration is done at     : 56444000.0 ps
Transactions ended at      : 64402000.0 ps
Percentage of Bus utilization : 77.000000

```

Figure 11: Performance Statistics for 512 writes and 512 reads – Row switching for 64 writes or reads.

You can now apply the knowledge learned in the earlier section to improve the efficiency. To utilize all of the banks instead of changing the row, modify the address pattern to change the bank as shown below. This is equivalent to setting ROW\_BANK\_Column in the memory address mapping setting in the MIG GUI.

```

3f_0_0_0000_000_0
3f_0_1_0000_000_0
3f_0_2_0000_000_0
3f_0_3_0000_000_0
3f_0_4_0000_000_0
3f_0_5_0000_000_0
3f_0_6_0000_000_0
3f_0_7_0000_000_0
3f_0_0_0000_000_1
3f_0_1_0000_000_1
3f_0_2_0000_000_1
3f_0_3_0000_000_1
3f_0_4_0000_000_1
3f_0_5_0000_000_1
3f_0_6_0000_000_1
3f_0_7_0000_000_1

```

At the end of the simulation, you will see that the earlier 77 Percent Bus Utilization is now 87!

```

*****
                        PERFORMANCE STATISTICS
*****
Number of Writes          : 512
Number of Reads           : 512
Calibration is done at    : 56444000.0 ps
Transactions ended at     : 63478000.0 ps
Percentage of Bus utilization : 87.000000

```

*Figure 12: Performance Statistics for 512 writes and 512 reads - Bank switching for 64 writes or reads.*

If you still require higher efficiency, you can go for large packet sizes of 1024 or 2048 bytes, or consider a manual refresh.

**Note:** Xilinx does not encourage bypassing controller refresh, as we are unsure if you will be able to meet the JEDEC auto Refresh timing, which affects data reliability. From the controller's you can change NBANNBANK\_MACH to see the performance improvement. However, this may affect your design timing, Please refer to (Xilinx Answer 36505) for details on nBANK\_MACH.

```

103 parameter DRAM_WIDTH      = 8,
104 // # of DQ per DQS
105 parameter ECC             = "OFF",
106 parameter DATA_WIDTH     = 8,
107 parameter ECC_TEST        = "OFF",
108 parameter PAYLOAD_WIDTH   = (ECC_TEST == "OFF") ? DATA_WIDTH : DQ_WIDTH,
109 parameter MEM_ADDR_ORDER  = "BANK_ROW_COLUMN",
110 //Possible Parameters
111 //1.BANK_ROW_COLUMN : Address mapping is
112 // in form of Bank Row Column.
113 //2.ROW_BANK_COLUMN : Address mapping is
114 // in the form of Row Bank Column.
115 //3.TG_TEST : Scrambles Address bits
116 // for distributed Addressing.
117
118 //parameter nBANK_MACHS    = 4,
119 parameter nBANK_MACHS     = 8,
120 parameter nBANKS          = 1,
121 // # of Ranks.

```

Figure 13: nBANK\_MACH parameter

Open core\_name\_mig\_sim.v file and change the parameters nBANK\_MACHS from 4 to 8 and re-run the simulation.

To have the parameter value take effect in hardware, you need to update the core\_name\_mig.v file. I used the same pattern where we got 87% bus utilization (Figure 2). With nBANK\_MACHS set to 8, the efficiency is now 90%.

```

# =====
# Performance Simulation Completed Successfully!!!
# Data integrity check is not performed in performance simulation
# =====
#
# *****
#
# PERFORMANCE STATISTICS
# *****
#
# Number of Writes          : 512
# Number of Reads          : 512
# Calibration is done at    : 56444000.0 ps
# Transactions ended at    : 63238000.0 ps
# Percentage of Bus utilization : 90.000000
#
# ..

```

Figure 13: Efficiency with nBANK\_MACH = 8

Also, make a note that  $\frac{1}{2}$  and  $\frac{1}{4}$  controllers negatively affect efficiency due to their latencies. For example, since we can only send commands every 4 CK cycles, there is sometimes extra padding when adhering to minimum DRAM timing specs, which can decrease efficiency from the theoretical. Try out different controllers to find the one that suits your efficiency requirement. References

1. Zynq-7000 AP SoC and 7 Series FPGAs MIS v2.3 [UG586]




2. Xilinx MIG Solution Centre <http://www.xilinx.com/support/answers/34243.html>

Revision History


13/03/2015 – Initial release..


Documents / Resources

	<p><a href="#">XILINX 63234 END FPGA Distributor [pdf]</a> User Guide</p> <p>63234 END FPGA Distributor, 63234, END FPGA Distributor, FPGA Distri butor</p>
---	---

References

- [AMD Customer Community](#)
- [User Manual](#)

 XILINX

 XILINX

63234, 63234 END FPGA Distributor, END FPGA Distributor, FPGA Distributor,

—Previous Post

[XILINX ZCU111 Zynq Ultra Scale Boards and Kits Instructions](#)

Leave a comment

Your email address will not be published. Required fields are marked \*

Comment \*

Name

Email

Website

☐ Save my name, email, and website in this browser for the next time I comment.

Post Comment

Manuals+, Privacy Policy | @manuals.plus | YouTube

This website is an independent publication and is neither affiliated with nor endorsed by any of the trademark owners. The "Bluetooth®" word mark and logos are registered trademarks owned by Bluetooth SIG, Inc. The "Wi-Fi®" word mark and logos are registered trademarks owned by the Wi-Fi Alliance. Any use of these marks on this website does not imply any affiliation with or endorsement.