Design of a Bridge between AHIR system and DDR3 SDRAM

A Dissertation
submitted in partial fulfillment of the requirements
for the Degree of

Master of Technology

with
specialization in

Electronic Systems

by

Swatantara Chakraborty
(15307R011)

Supervisor

Prof. Madhav P. Desai

Department of Electrical Engineering
Indian Institute of Technology, Bombay
Powai, Mumbai - 400 076.

2017-18

Dissertation Approval

The dissertation entitled

Design of a Bridge between AHIR system and DDR3 SDRAM

by

Swatantara Chakraborty
(Roll No. : 15307R011)

is approved for the degree of

Master of Technology in Electrical Engineering

Prof. Madhav P. Desai	Prof.
Dept. of Electrical Engineering	Dept. of Electrical Engineering
(Supervisor)	(Examiner)

Date: June 30, 2018
Place: IIT, Bombay.

Declaration

I hereby declare that the dissertation entitled **"Design of a Design of a Bridge between AHIR system and DDR3 SDRAM"** represents my ideas in my own words. In all occurrences where ideas or words or diagrams are taken from books/papers/-electronic media, I have adequately cited and acknowledged the original sources. I declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will result in disciplinary action as per the norms of Institute and can also evoke legal action from the sources which have thus not been properly acknowledged or from whom proper permission has not been taken.

Swatantara Chakraborty
(Roll no. : 15307R011)

Date: June 30, 2018

Place: IIT, Bombay

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude to my guide Prof. Madhav P. Desai who with his profound experience and never ending enthusiasm provided constant support and guidance to my dissertation.

I acknowledge the *Department of Electrical Engineering, Indian Institute of Technology Bombay* for providing the various tools to work on.

Finally, I would like to thank my parents, friends and lab mates, who provided the needed base to work conveniently.

ABSTRACT

Our aim is to design a bridge module that can directly load and fetch streams of data from the external DDR3 SDRAM in an Xilinx cards through the PCIe bus in a RIFFA environment. This would enable us to prestore the necessary input data before executing the loaded design in the FPGA. Then, we can simply read the data while running the design instead of sending the packets through the PCIe bus through a RIFFA testbench during the execution. Hence, the overall throughput of any implemented design would improve to a great extent.

In our current project, we have proposed a design for a pipelined bridge module .

Contents

1 Description of the Host Platform **5**

1.1 Current Status 5

1.2 Target Status 7

2 Memory Interface Generator **9**

3 MIG Solutions: Overview **12**

3.1 Generation of the Memory Controller By MIG 12

3.2 Customized Pin Layout for Optimum Design : Bank Selection 13

3.3 The Final Customized Pin-Layout 16

3.4 Generation of the MIG IP 18

3.5 Explanation of the MIG Example Design: Ref Example Design Block Diagram 20

3.6 Example Design Flow : Traffic Generation 20

3.7 Verification of the example-top constraints 20

3.8 Building the Example Design 23

3.9 Generation of the example-top bitfile 23

3.10 Programming the FPGA 25

3.10.1 How to Debug 25

3.10.2 Output from Chipscope Analyzer 26

4 Design of the AHIR-MIG Bridge Module **28**

4.1 Parameter Specifications for the Current Design 28

4.2 Description of the Request Packet 32
4.3 Description of the Response Packet 34

5 Proposed Design of the Bridge 36

5.0.1 Design of the intermediate buffers 36
5.0.2 Design of the Request Module 36
5.0.3 Design of the Response Module 37

6 Test Results 42

6.0.4 Request Receive 42
6.0.5 Request Send 42
6.0.6 Response receive 42
6.0.7 Response Send 42

7 References 47

List of Figures

1.1 The Current Status of the Host Environment 6
1.2 Target Status of the Host Environment 8

3.1 Data Group Selection 15
3.2 Data Group Selection 17
3.3 Block Diagram for the Example Design 19
3.4 Successful Constraint Verification 22
3.5 Synthesis, Build, Map and Par of the Example Design 24
3.6 Waveforms Obtained From Chipscope Analyzer 27
3.7 Waveforms Obtained From Chipscope Analyzer 27

4.1 AHIR-MC Bridge 30
4.2 AHIR-MC Bridge 30
4.3 AHIR-MC Bridge 31
4.4 MIG cmd Path 31
4.5 MIG data Path 31
4.6 A Request Packet 33
4.7 A Response Packet 35

5.1 Bridge Overview 38
5.2 FIFO Buffers 38
5.3 Ahir Data Register 38
5.4 The Request Receive Module 39
5.5 The Cmd Buffer Count Module 39

5.6 The Pending Buffer Count Module 40
5.7 The Request-Send Module 40
5.8 The Response receive Module 40
5.9 The Response Send Module 41

6.1 Request Receive Read Cmd 43
6.2 Request Receive Write Cmd 43
6.3 Request Send Two Reads and Write Cmd 44
6.4 Response Receive 45
6.5 Response Read and Write 46

CHAPTER 1

Description of the Host Platform

We start the discussion by describing the basics of the environment we are working on. The design is applicable to all Xilinx FPGAs, like VC-709, ML-605, KC-705 etc. The following diagram demonstrates the external memory in ML605 for example.

The ML605 card provides a Virtex-6 FPGA (XC6VLX240T - 1FFG1156). A single 512 MB DDR3(Double Data Rate Type 3) Synchronous Dynamic Random Access Memory is provided for user applications. The overall block diagram is shown below: (includes only the DDR3 and the FPGA top). Fig 1.1 and 1.2 illustrates the current status of the host platform

1.1 Current Status

The AHIR-SYSTEM.VHDL is the top module of the user design generated through the AHIR tool-chain . This module along with the RIFFA hdl are synthesized together to generated the necessary bit file to be dumped into the FPGA via the PCI-e bus. Currently, there is no way to communicate with the DDR3 SDRAM directly from the AHIR system.

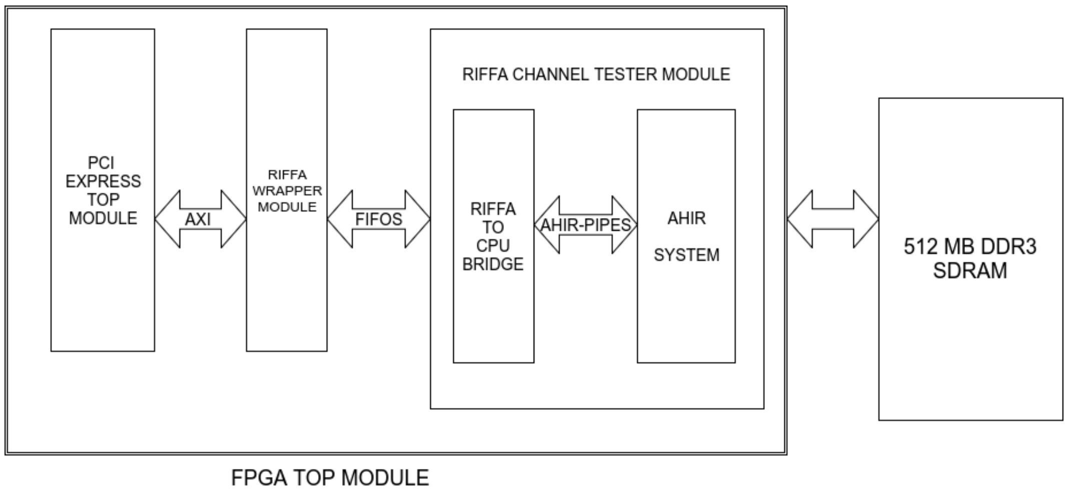


Figure 1.1: The Current Status of the Host Environment

1.2 Target Status

Our target is to devise a direct communication between the AHIR System and the DDR3 SDRAM.

The Memory Controller is generated using the Xilinx MIG(Memory Interface Generator) which is a part of the Xilinx Coregen. This provides a controller module that enables us to interact with the DDR3 SDRAM.

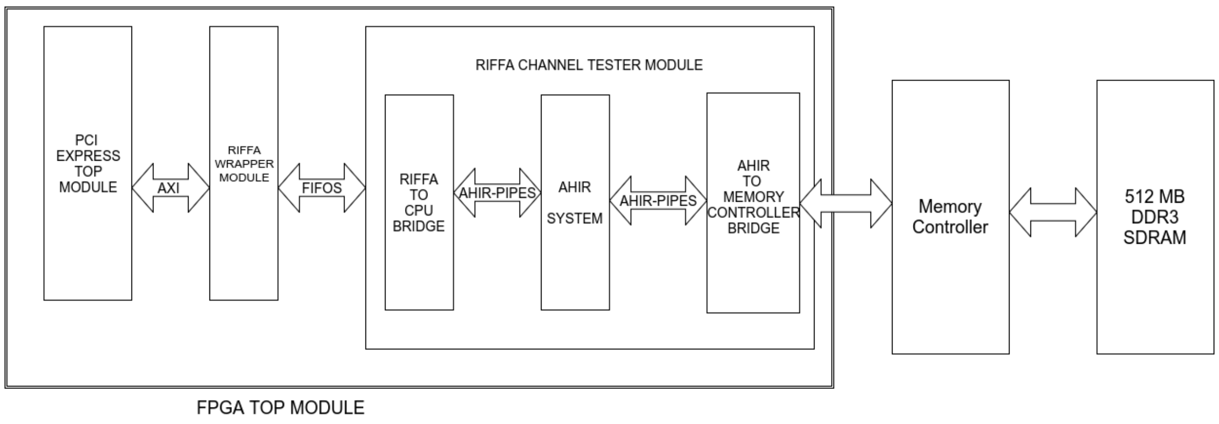


Figure 1.2: Target Status of the Host Environment

CHAPTER 2

Memory Interface Generator

This chapter focusses on giving out a basic overview of the MIG user interface.

The following are the modules :

- **User-Design** : The user design block shown in Figure is any FPGA design that requires to be connected to an external DDR2 or DDR3 SDRAM. The user design connects to the memory controller via the user interface. An example user design is provided with the core.

- **User Interface Block and User Interface** : The user interface (UI) block presents the UI to the user design block. It provides a simple alternative to the native interface by presenting a flat address space and buffering read and write data.

- **Memory Controller and Native Interface** : The front end of the memory controller (MC) presents the native interface to the UI block. The native interface allows the user design to submit memory read and write requests and provides the mechanism to move data from the user design to the external memory device, and vice versa. The back-end of the MC is not our concern for the time being.

The **MIG User Interface** is illustrated below:

The *input connections* to the UI are as follows :

1. **app-addr[ADDR-WIDTH â 1:0]** : This input indicates the address for the current request.
2. **app-cmd[2:0]** : This input selects the command for the current request.
3. **app-en** : This is the active-High strobe for the app-addr[], app-cmd[2:0], app-sz, and app-hi-pri inputs.
4. **app-hi-pri** : This active-High input elevates the priority of the current request.
5. **app-sz** : This reserved input should be connected to logic 1.
6. **app-wdf-data [APP-DATA-WIDTH â 1:0]** : This provides the data for write commands.
7. **app-wdf-end** : This active-High input indicates that the current clock cycle is the last cycle of input data on app-wdf-data[].
8. **app-wdf-mask[APP-MASK-WIDTH â 1:0]** : This provides the mask for app-wdf-data[].*(Optional: Not Used)*
9. **app-wdf-wren** : This is the active-High strobe for app-wdf-data[].
10. **clk** : This UI clock must be half of the DRAM clock.
11. **rst** : This is the active-High UI reset.

The *output connections* to the UI are as follows :

- **app-rd-data[APP-DATA-WIDTH â 1:0]** : This provides the output data from read commands.
- **app-full** : This output indicates that the UI command FIFO is full. If the signal is asserted when app-en is enabled, the current app-cmd and app-addr must be retried until app-full is deasserted.
- **app-rd-data-end** : This active-High output indicates that the current clock cycle is the last cycle of output data on app-wdf-data[].

- **app-rd-data-valid** : This active-High output indicates that app-rd-data[] is valid.
- **app-rdy** : This active-High output acknowledges the requests strobed by app-en.
- **app-wdf-rdy** : This output indicates that the write data FIFO is ready to receive data. Write data is accepted when app-wdf-rdy = 1âb1 and app-wdf-wren = 1âb1.
- **app-wdf-full** : This output indicates that the write data FIFO is full. If this signal is asserted when app-wdf-wren is enabled, the current write input data must be retried.

The memory controller can be connected using either the UI or the native interface. The UI resembles a simple FIFO interface and always returns the data in order. The native interface offers higher performance in some situations, but is more challenging to use. So, we shall use the User Interface for a simple logical address interface. It provides a simple alternative to the native interface by presenting a flat address space and buffering read and write data.

Write Path

The write data is registered in the write FIFO when app-wdf-wren is asserted and app-wdf-full is Low. If app-wdf-full is asserted, the user logic needs to hold app-wdf-wren and app-wdf-end High along with the valid app-wdf-data value until app-wdf-full is deasserted. The app-wdf-mask signal can be used to mask out the bytes to write to external memory.

CHAPTER 3

MIG Solutions: Overview

The Memory Controller to the DDR3 SDRAM has been generated using Xilinx MIG tool and the output has been analyzed using Xilinx Chipscope Analyzer.

3.1 Generation of the Memory Controller By MIG

The Memory Interface Generator tool, a part of the Xilinx Coregenerator, creates memory controllers for Xilinx FPGAs. It generates complete customized VHDL or Verilog RTL source codes, pin-out and design constraints for the FPGA selected.

- The coregenerator is invoked from the terminal and the FPGA details for ML605 are selected.
- The design rtl codes can be generated in VHDL or verilog. Here, we have chosen Verilog. Synthesis tool is selected as Xilinx ISE.
- The MIG module (IP) has to be customized in the following steps:

  1. The Controller type is selected as DDR3 SDRAM. The allowed clock cycle range is (2500-3300) ps. For a speed grade of -1, the frequency is 400 MHz as selected in our design.

2. The Memory type is SODIMMs for which the permissible data width is 64 bits. The ordering of commands is made strict so that the controller executes the commands in the exact order as they are received, and does not reorder them to improve efficiency.

- The Debug signals for the memory controller is activated in order to monitor the debug signals on Chipscope ILA.
- The pin-layout has to be customized in order to meet the design constraints. So we have to deselect the standard pin-layout and pick the optimum banks for a new design.

3.2 Customized Pin Layout for Optimum Design : Bank Selection

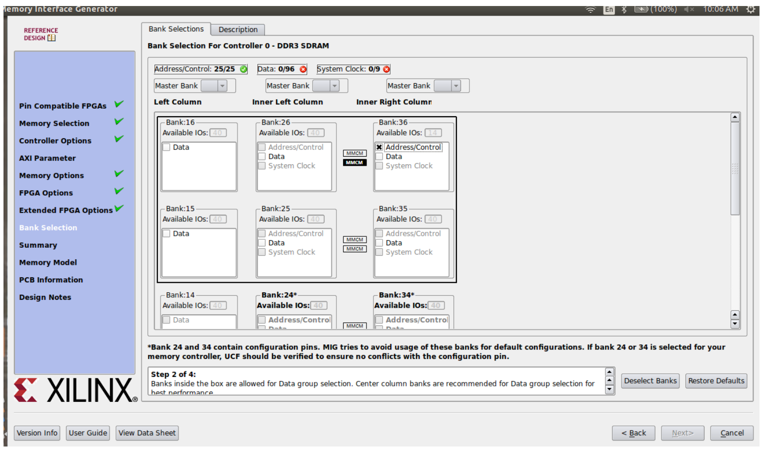
This feature allows the selection of banks for the memory interface. Banks can be selected for different classes of memory signals, such as:

- Address and control signals
- Data signals
- System clock

We start with the address/control signals.

- **Address/Control Group** : Select the address/control group from one of the white banks. Only inner columns are allowed
- **Data Group** : After the address bank has been assigned, MIG will only allows the data group to be chosen within the banks inside the black box as shown in the next slide.
- **System Clock** : It is selected from any one of the enabled banks.

- **Master Bank Selection** : Two extra pins are required to set up a DCI reference that provides better signal integrity.We need to select the master bank from one of the list of banks shown in the pull-down menu. Digitally Controlled Impedance(DCI) allows the use of on-chip internal resistors in the FPGA for termination.



Bank Selection For Controller 0 - DDR3 SDRAM

Address/Control: **25/25** ✓ Data: **0/96** ✗ System Clock: **0/9** ✗

Master Bank | Master Bank | Master Bank

Left Column | **Inner Left Column** | **Inner Right Column**

Bank:16 Available IOs: 40 ☐ Data	Bank:26 Available IOs: 40 ☐ Address/Control ☐ Data ☐ System Clock	Bank:36 Available IOs: 14 ☒ Address/Control ☐ Data ☐ System Clock
Bank:15 Available IOs: 40 ☐ Data	Bank:25 Available IOs: 40 ☐ Address/Control ☐ Data ☐ System Clock	Bank:35 Available IOs: 40 ☐ Address/Control ☐ Data ☐ System Clock
Bank:14 Available IOs: 40 ☐ Data	**Bank:24*** **Available IOs:** 40 ☐ **Address/Control**	**Bank:34*** **Available IOs:** 40 ☐ **Address/Control**

***Bank 24 and 34 contain configuration pins. MIG tries to avoid usage of these banks for default configurations. If bank 24 or 34 is selected for your memory controller, UCF should be verified to ensure no conflicts with the configuration pin.**

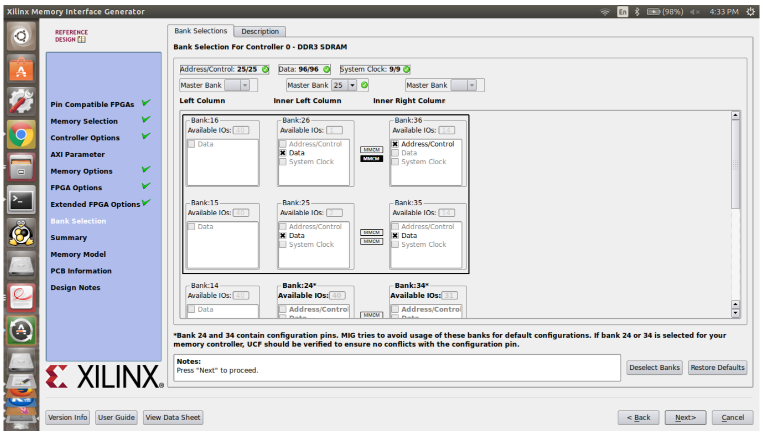
Step 2 of 4:
Banks inside the box are allowed for Data group selection. Center column banks are recommended for Data group selection for best performance

Figure 3.1: Data Group Selection

3.3 The Final Customized Pin-Layout

- We have selected Bank 36 for Address/Control Group,
- Bank 26, Bank 25 , Bank 35 for Data Group
- Bank 34 for System Clock
- The inner left column Master Bank(25) has been selected for DCI.

The customization of the MIG IP is now complete.



Bank Selection For Controller 0 - DDR3 SDRAM

Address/Control: 25/25 Data: 96/96 System Clock: 9/9

Master Bank 25

Bank:16 Available IOs: 40 — Data

Bank:26 Available IOs: 1 — Address/Control, [✓ Data], System Clock

Bank:36 Available IOs: 14 — [✓ Address/Control], Data, System Clock

Bank:15 Available IOs: 40 — Data

Bank:25 Available IOs: 2 — Address/Control, [✓ Data], System Clock

Bank:35 Available IOs: 14 — Address/Control, [✓ Data], System Clock

Bank:14 Available IOs: 40 — Data

Bank:24* Available IOs: 40 — Address/Control

Bank:34* Available IOs: 31 — Address/Control

*Bank 24 and 34 contain configuration pins. MIG tries to avoid usage of these banks for default configurations. If bank 24 or 34 is selected for your memory controller, UCF should be verified to ensure no conflicts with the configuration pin.

Notes:
Press "Next" to proceed.

Figure 3.2: Data Group Selection

3.4 Generation of the MIG IP

The MIG IP is generated by accepting the Xilinx license. The MIG output consists of the the desired memory controller interface organized in the form of verilog design files and executables.

The generated core consists of an example-design of the memory controller interface that communicates with the DDR3 SDRAM.

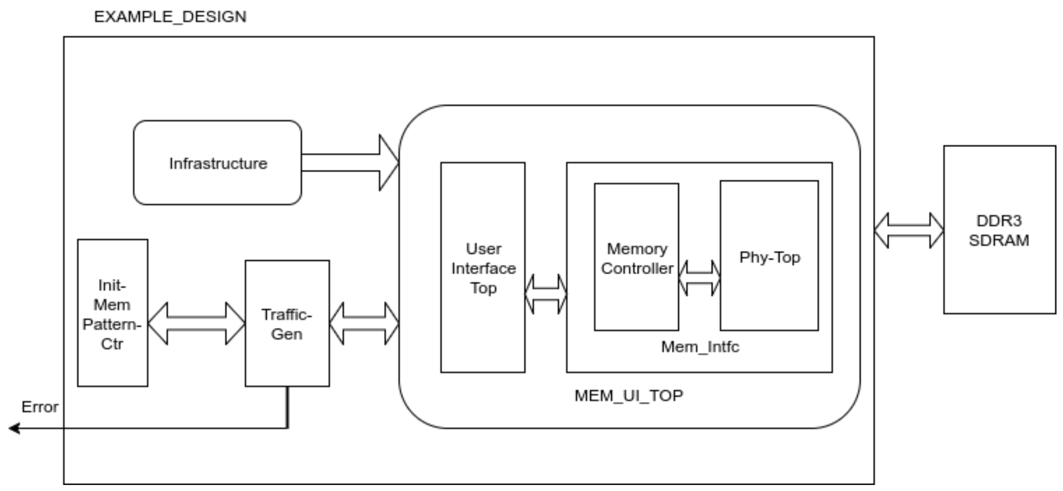


Figure 3.3: Block Diagram for the Example Design

3.5 Explanation of the MIG Example Design: Ref Example Design Block Diagram

- **Infrastructure**: This module helps in clock generation and distribution, and reset synchronization.
- **Mem_intfc** : This is the top-level memory interface block , instantiates the memory controller block and the phy block.
- **Mem_ui_top** : This is the top-level memory interface controller wrapper with the user interface.
- **Phy_Top** : Top-level memory physical layer interface.
- **MC** : Top level memory sequencer structural block.

3.6 Example Design Flow : Traffic Generation

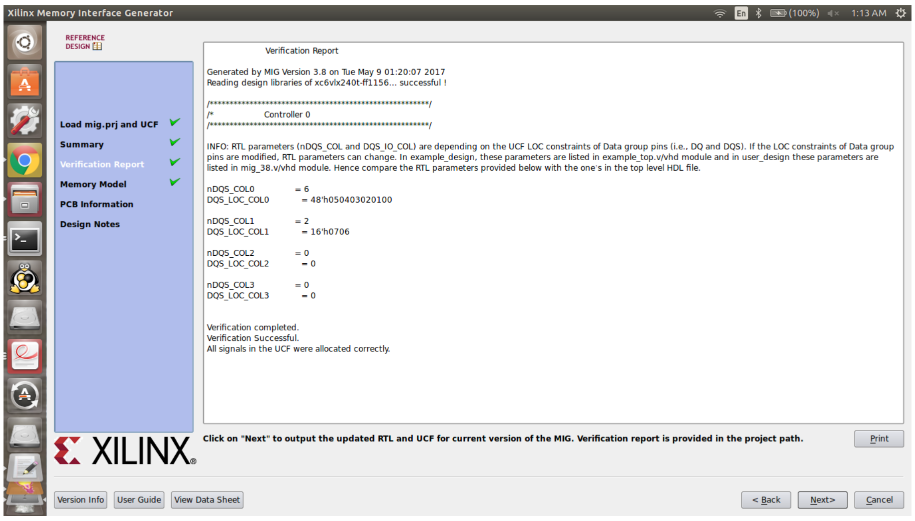
The **traffic generator module** contained can be parameterized to create various stimulus patterns for the memory design. It can produce repetitive test patterns for verifying design integrity as well as pseudo-random data streams that model real-world traffic.

he **Init Memory Pattern Control** block directs the traffic generator to step sequentially through all the addresses in the address space, writing the appropriate data value to each location in the memory device as determined by the selected data pattern. The pseudo-random data , address and command patterns are generated through an internal 64 bit LFSR.

3.7 Verification of the example-top constraints

It is important to verify whether the selected pin-out for the controller core satisfies the user constraint file specifications. To do so, we invoke the MIG GUI again and select the

MIG project file(.prj) and the example_top.ucf file for the verification. The verification is successful for the design we have generated.



```
                    Verification Report

Generated by MIG Version 3.8 on Tue May 9 01:20:07 2017
Reading design libraries of xc6vlx240t-ff1156... successful !

/*******************************************************/
/*                  Controller 0
/*******************************************************/

INFO: RTL parameters (nDQS_COL and DQS_IO_COL) are depending on the UCF LOC constraints of Data group pins (i.e., DQ and DQS). If the LOC constraints of Data group
pins are modified, RTL parameters can change. In example_design, these parameters are listed in example_top.v/vhd module and in user_design these parameters are
listed in mig_38.v/vhd module. Hence compare the RTL parameters provided below with the one's in the top level HDL file.

nDQS_COL0           = 6
DQS_LOC_COL0          = 48'h050403020100

nDQS_COL1           = 2
DQS_LOC_COL1          = 16'h0706

nDQS_COL2           = 0
DQS_LOC_COL2          = 0

nDQS_COL3           = 0
DQS_LOC_COL3          = 0


Verification completed.
Verification Successful.
All signals in the UCF were allocated correctly.
```

Click on "Next" to output the updated RTL and UCF for current version of the MIG. Verification report is provided in the project path.

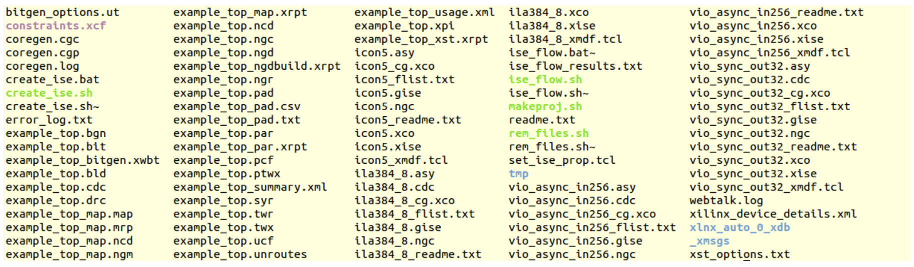
Figure 3.4: Successful Constraint Verification

3.8 Building the Example Design

Now that the design satisfies the constraints, our next step is to run the design through synthesis, build, map, and par. To do so, we write a script file ise_flow.sh along with the necessary specifications. As we need to monitor the debug signals using Xilinx Chipscope Analyzer later, we need to include the ila384_8_cg.xco module before we synthesize the design (Integrated Logic Analyzer). The Synthesis tool used is XST and the implementation tool is ISE.

3.9 Generation of the example-top bitfile

The following screenshot shows all the generated files to example_top.bit



```
bitgen_options.ut        example_top_map.xrpt       example_top_usage.xml  ila384_8.xco              vio_async_in256_readme.txt
constraints.xcf          example_top.ncd            example_top.xpi        ila384_8.xise             vio_async_in256.xco
coregen.cgc              example_top.ngc            example_top_xst.xrpt   ila384_8_xmdf.tcl         vio_async_in256.xise
coregen.cgp              example_top.ngd            icon5.asy              ise_flow.bat~             vio_async_in256_xmdf.tcl
coregen.log              example_top_ngdbuild.xrpt  icon5_cg.xco           ise_flow_results.txt      vio_sync_out32.asy
create_ise.bat           example_top.ngr            icon5_flist.txt        ise_flow.sh               vio_sync_out32.cdc
create_ise.sh            example_top.pad            icon5.gise             ise_flow.sh~              vio_sync_out32_cg.xco
create_ise.sh~           example_top_pad.csv        icon5.ngc              makeproj.sh               vio_sync_out32_flist.txt
error_log.txt            example_top_pad.txt        icon5_readme.txt       readme.txt                vio_sync_out32.gise
example_top.bgn          example_top.par            icon5.xco              rem_files.sh              vio_sync_out32.ngc
example_top.bit          example_top_par.xrpt       icon5.xise             rem_files.sh~             vio_sync_out32_readme.txt
example_top_bitgen.xwbt  example_top.pcf            icon5_xmdf.tcl         set_ise_prop.tcl          vio_sync_out32.xco
example_top.bld          example_top.ptwx           ila384_8.asy           tmp                       vio_sync_out32.xise
example_top.cdc          example_top_summary.xml    ila384_8.cdc           vio_async_in256.asy       vio_sync_out32_xmdf.tcl
example_top.drc          example_top.syr            ila384_8_cg.xco        vio_async_in256.cdc       webtalk.log
example_top_map.map      example_top.twr            ila384_8_flist.txt     vio_async_in256_cg.xco    xilinx_device_details.xml
example_top_map.mrp      example_top.twx            ila384_8.gise          vio_async_in256_flist.txt xlnx_auto_0_xdb
example_top_map.ncd      example_top.ucf            ila384_8.ngc           vio_async_in256.gise      _xmsgs
example_top_map.ngm      example_top.unroutes       ila384_8_readme.txt    vio_async_in256.ngc       xst_options.txt
```

Figure 3.5: Synthesis, Build, Map and Par of the Example Design

3.10 Programming the FPGA

The generated example_top.bit file has been dumped into the FPGA using Xilinx impact tool.(through JTAG cable) Xilinx Chipscope Analyzer ILA(Integrated Logic Analyzer) has been invoked to monitor the debug signals. To do so, the example_top.bit file and the example_top.cdc file generated when the ise_flow.sh script was run, are included in a new Chipscope project. The ChipScope Pro tool allows the user to set trigger conditions to capture application and MIG signals in hardware. Captured signals can then be analyzed through the ChipScope Pro Logic Analyzer tool.

3.10.1 How to Debug

In order to debug we have to load the provided example_design onto the board in question. This is a known working solution with a traffic generator design that checks for data errors. This design should complete successfully with the assertion of phy_init_done and no assertions of error. Assertion of phy_init_done signifies successful completion of calibration while no assertions of error signifies that the data is written to and read from the memory compare with no data errors.

- **dbg_rdlvl_done** : Each bit is driven to a static 1 as each stage of read leveling is completed. The dbg_rdlvl_done[0] signal corresponds to stage 1. If both the read stages are successfully completed, both dbg_rdlvl_done[0] and dbg_rdlvl_done[1] should be one, that is dbg_rdlvl_done as a bit vector should be shown as 3 in hex format.

- **dbg_rdlvl_err** : This output indicates if an error occurred during each stage of read leveling. If there is no error, both dbg_rdlvl_err[0] and dbg_rdlvl_err[1] should be 0.

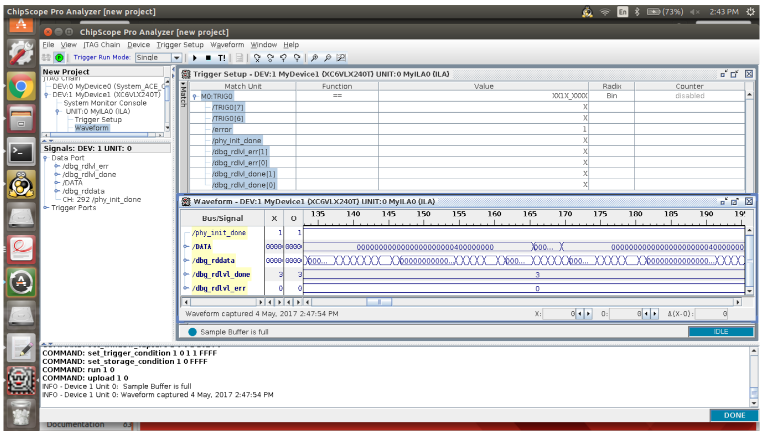
- **dbg_rddata** : This is the capture read data synchronized to the clk clock domain. It should show the psedo-random data generated by the traffic generator.

The debug signals to be monitored using Chipscope Analyzer should be included in the example_top.cdc file . Note that the DATA pins of the ILA are not utilised.

3.10.2 Output from Chipscope Analyzer

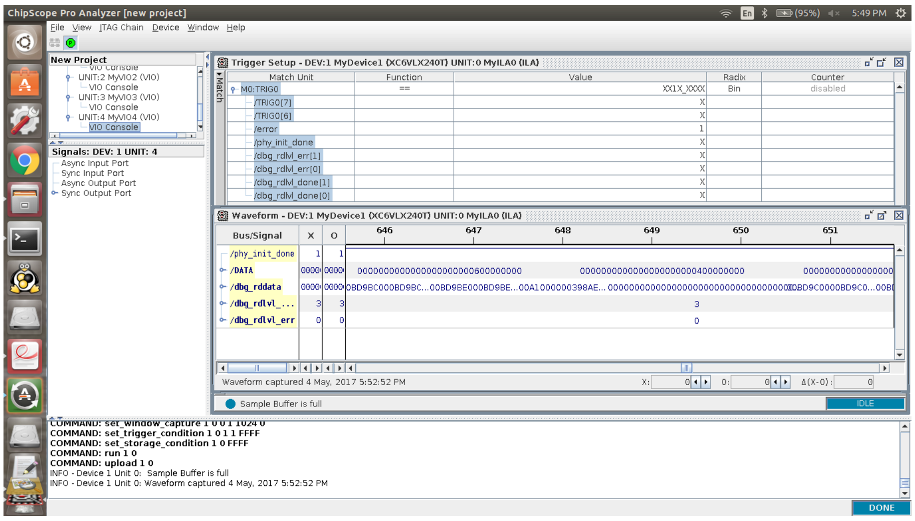
The dbg_rdlvl_err is 0 , dbg_rdlvl_done is 3 and phy_init_done is asserted indicating successful calibration.

The dbg_rddata shows the pseudo-random data generated , as expected.


ChipScope Pro Analyzer [new project]
Trigger Setup - DEV:1 MyDevice1 (XC6VLX240T) UNIT:0 MyILA0 (ILA)
Waveform - DEV:1 MyDevice1 (XC6VLX240T) UNIT:0 MyILA0 (ILA)
/phy_init_done
/DATA
/dbg_rddata
/dbg_rdlvl_done
/dbg_rdlvl_err
Sample Buffer is full
IDLE
DONE

```
COMMAND: set_trigger_condition 1 0 1 1 FFFF
COMMAND: set_storage_condition 1 0 FFFF
COMMAND: run 1 0
COMMAND: upload 1 0
INFO - Device 1 Unit 0:  Sample Buffer is full
INFO - Device 1 Unit 0: Waveform captured 4 May, 2017 2:47:54 PM
```

Figure 3.6: Waveforms Obtained From Chipscope Analyzer


ChipScope Pro Analyzer [new project]
Trigger Setup - DEV:1 MyDevice1 (XC6VLX240T) UNIT:0 MyILA0 (ILA)
Waveform - DEV:1 MyDevice1 (XC6VLX240T) UNIT:0 MyILA0 (ILA)
/phy_init_done
/DATA
/dbg_rddata
/dbg_rdlvl_...
/dbg_rdlvl_err
Sample Buffer is full
IDLE
DONE

```
COMMAND: set_window_capture 1 0 0 1 1024 0
COMMAND: set_trigger_condition 1 0 1 1 FFFF
COMMAND: set_storage_condition 1 0 FFFF
COMMAND: run 1 0
COMMAND: upload 1 0
INFO - Device 1 Unit 0:  Sample Buffer is full
INFO - Device 1 Unit 0: Waveform captured 4 May, 2017 5:52:52 PM
```

Figure 3.7: Waveforms Obtained From Chipscope Analyzer

CHAPTER 4

Design of the AHIR-MIG Bridge Module

Our job is to interface the ahir-system module with the user interface of the memory controller as explained earlier. For this, we need to study the input-output interfaces of the ahir system block as well. So, we first list out the AHIR pipes between the riffa-to-cpu bridge inside the channel tester and the ahir-system module. The **bridge-request interface** consists of:

1. out-data-pipe-read-req (request-req)
2. out-data-pipe-read-ack (request-ack)
3. out-data-pipe-read-data (request-data)

The **bridge-response interface** consists of:

1. in-data-pipe-write-ack (response-ack)
2. in-data-pipe-write-req (response-req)
3. in-data-pipe-write-data (response-data)

4.1 Parameter Specifications for the Current Design

- For a SODIMM DDR3 SDRAM, the data width is fixed at 64 bits.

- The address width is taken as 32 bits. However, 64 and 128 bit addresses may also be selected.
- The maximum burst length is fixed at 8.
- The clock rate can be between 75 MHz and 266 MHz for the user design because the DDR3 SDRAM clock should be double that of the design clock and the DDR3 permissible clock range is between 150 MHZ and 533 MHz.
- The target FPGA is Virtex 6 in an ML605 card.

The Bridge module will have one 64 bit input pipe called the Request pipe and one 64 bit output pipe called the Response pipe on the AHIR side of the interface.

The Bridge interface on the AHIR System side is illustrated in the diagram below:

The Bridge Interface on the Memory Controller Top side is showed in the diagram below:

The timing diagrams for the MIG interface are given below:

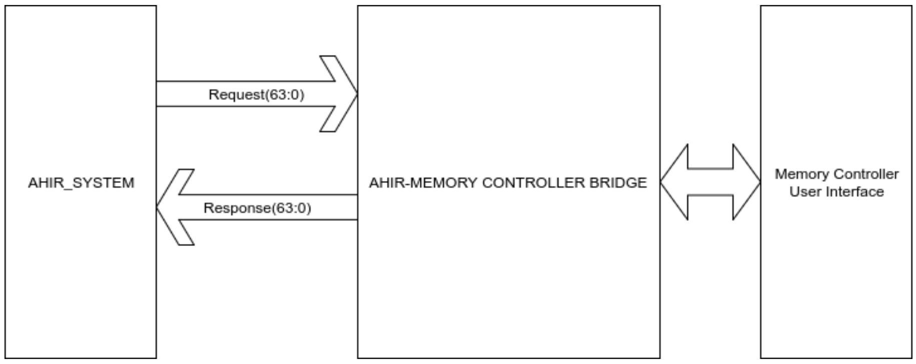

AHIR_SYSTEM
Request(63:0)
Response(63:0)
AHIR-MEMORY CONTROLLER BRIDGE
Memory Controller User Interface

Figure 4.1: AHIR-MC Bridge

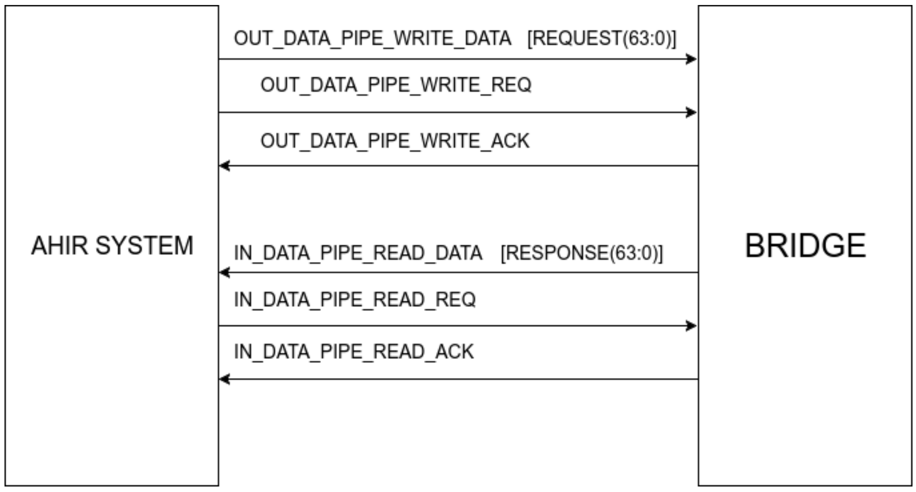


Figure 4.2: AHIR-MC Bridge

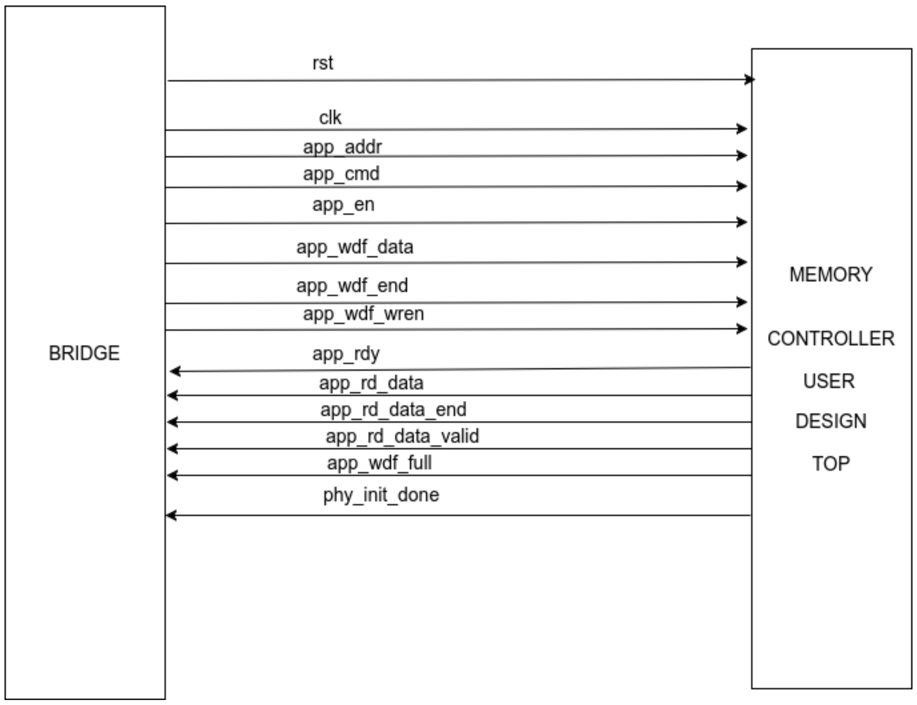


Figure 4.3: AHIR-MC Bridge

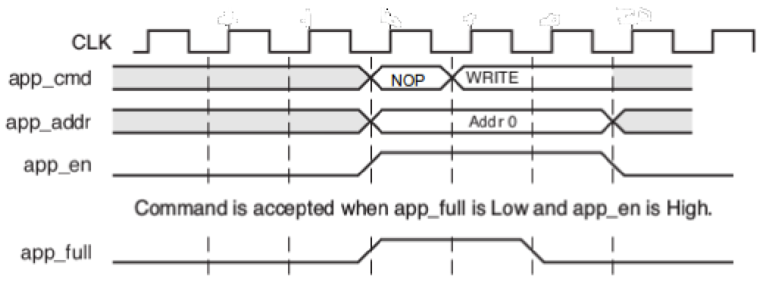

CLK
app_cmd
NOP
WRITE
app_addr
Addr 0
app_en
Command is accepted when app_full is Low and app_en is High.
app_full

Figure 4.4: MIG cmd Path

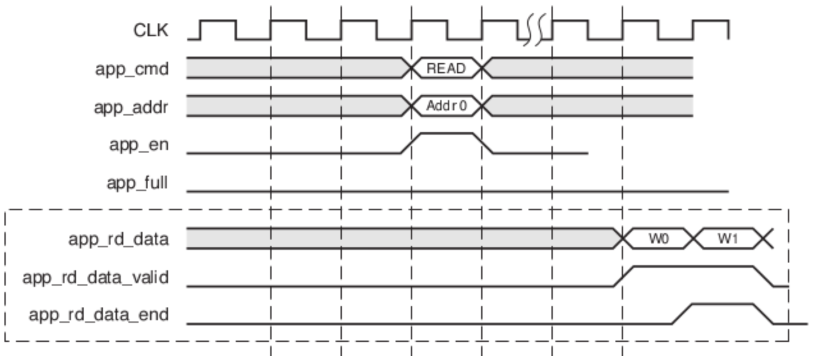


Figure 4.5: MIG data Path

4.2 Description of the Request Packet

The Request packet is a formatted header. Each packet is of 64 bits.

The cmd-type indicates whether it is a read/write request. 00 indicates a read command while 01 indicates a write command. Provision is kept for a 32 bit address, the start address is specified.

The number of packets indicates the number of 64 bit data packets that are to be read, starting from the start address of the burst. 3 bits are enough to express the number of packets , for a given burst length of 8. But 5 extra bits are kept, leaving scope for further development.

The Request Id indicates the identity or index of the request. *Note that the checksum bits are kept for future development, they are not included in our current design.*

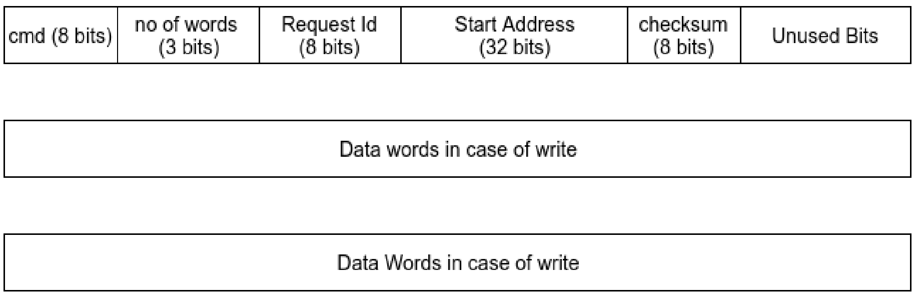


Figure 4.6: A Request Packet

4.3 Description of the Response Packet

The error fields indicate whether there is an error in the response or not. Currently, it is a single bit indicating just the presence or absence of error. The phy_init_done signal from the User Top module of the Memory Controller is 1 when the memory read/write operation has been completed successfully. This bit signal is mapped to the error bit in the response packet. So, a value of 0 would indicate error while that of 1 would indicate successful completion.

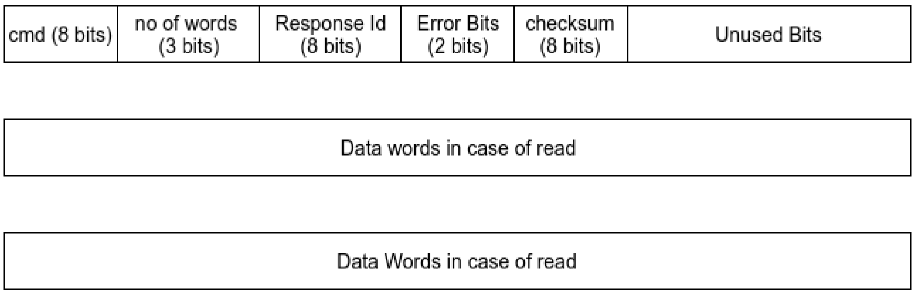


Figure 4.7: A Response Packet

CHAPTER 5

Proposed Design of the Bridge

A four-stage ipelined bridge has been conceptualised, the phases being, namely, the request-receive (A), request-send (B), response-receive(C) and response-send (D). The following diagram gives an overview of the system.

5.0.1 Design of the intermediate buffers

There are three buffers , namely, mig-command-buffer, pending buffer and the mig-response-buffer. The mig-cmd-buffer stores the request-data from ahir-system waiting to be sent to the mig top. The pending buffer stores the pending requests. The mig-response buffer similarly stores the response-words. Each buffer is currently 9 staged, i.e., the depth of the buffer is such that it can store one full request of maximum length. The maximum length request is a write request of 8 words.(9 including the command word); All these three buffers are FIFO structures. The cmd buffer, pending buffer and the response buffer parameters are henceforth denoted by indices 1,2 and 3 respectively.

5.0.2 Design of the Request Module

The request-module interface consists of two submodules, namely, the request-receive and the request-send. These two are described as follows: In order to reduce the critical path delay , an intermediate 64 bit register called the ahir data register is introduced between the ahir system and the request receive module. This will store one word of request.

Although this will increase the best case latency by one cycle, the overall optimum perfomance would be improved as we have decoupled the dependency on the buf_counts of the cmd and pending buffers in this approach.

In order to avoid deadlocks, a count is kept for each of the cmd and pending buffers. The depth of the cmd buffer is kept at 9 (to store one full request of maximum length), and that of the pending buffer is kept at 4.

The request-send module operates as follows:

5.0.3 Design of the Response Module

The response-module interface is shown below:

The **response receive** module operates as follows : The **response send** module operates as follows : In main state, if the administrative word is formed, pend_buf_decr_flag is set. Here, the count_words is a flag local to the response send module which counts the no of words buffered under the current response(that being sent).

As the data words for read (app_rd_data) can be handled by the response buffer 2 cycles after it is received from the mig interface , there is a two stage fifo inside between the interface called the **Mig Data Register**

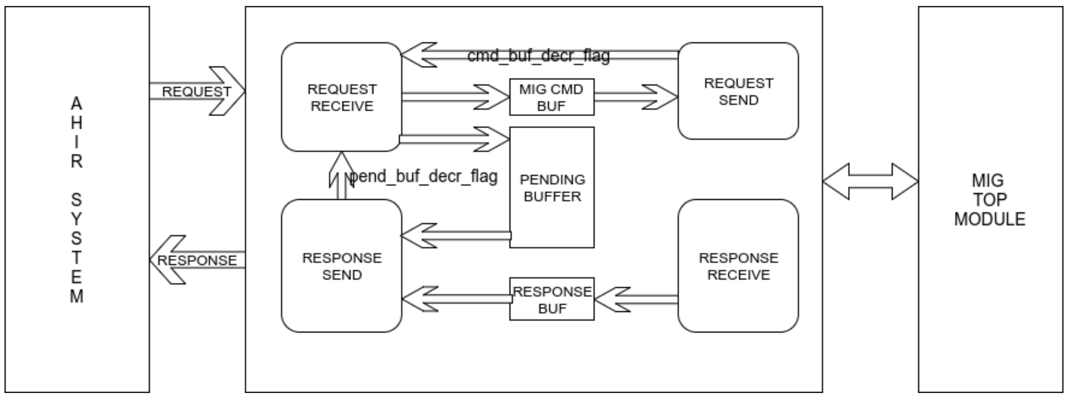


Figure 5.1: Bridge Overview

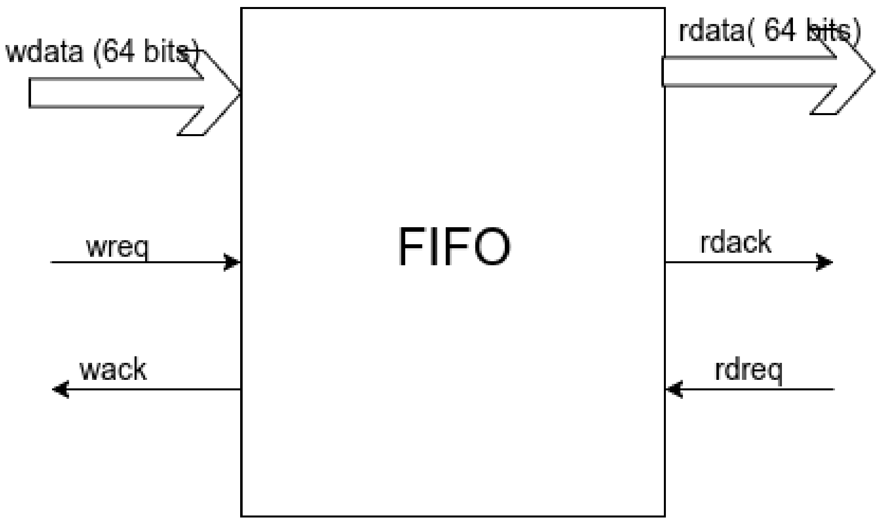


Figure 5.2: FIFO Buffers

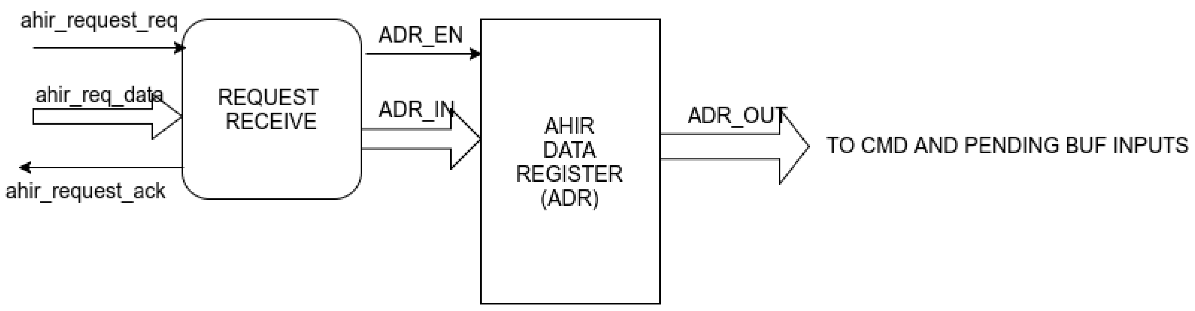


Figure 5.3: Ahir Data Register

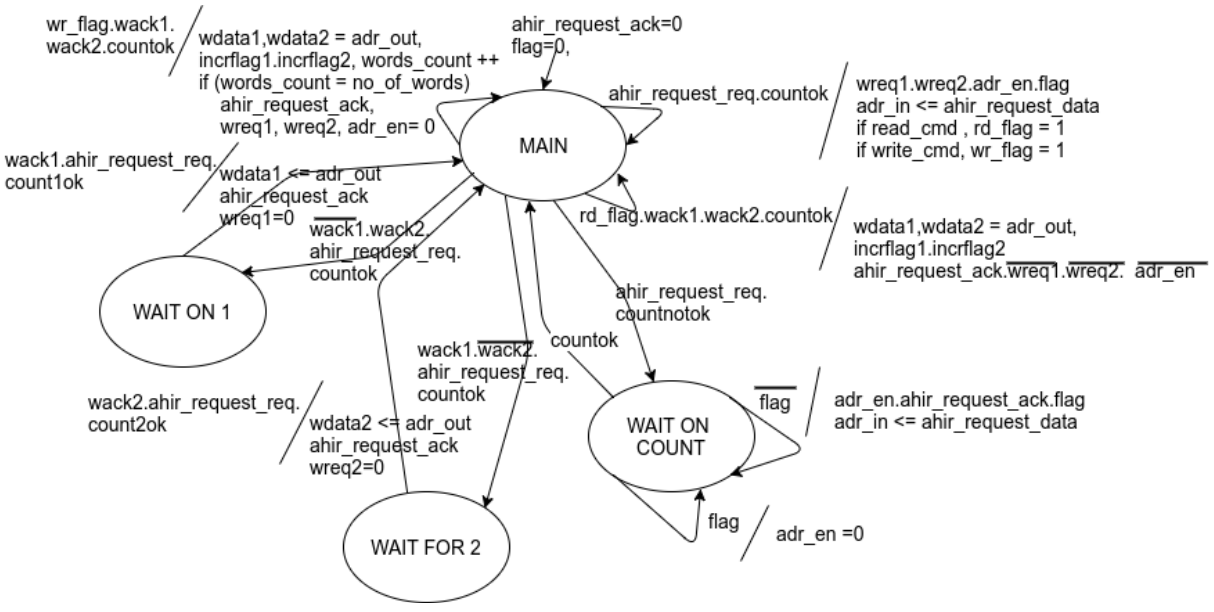


Figure 5.4: The Request Receive Module

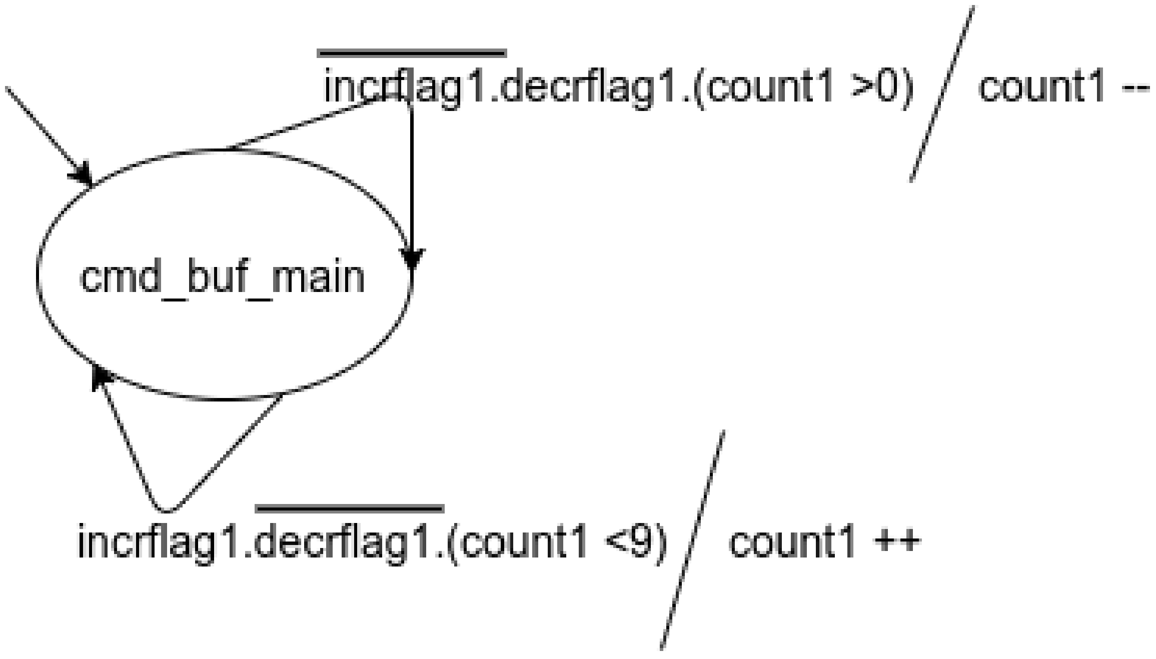

incrflag1.decrflag1.(count1 >0)
count1 --
cmd_buf_main
incrflag1.decrflag1.(count1 <9)
count1 ++

Figure 5.5: The Cmd Buffer Count Module

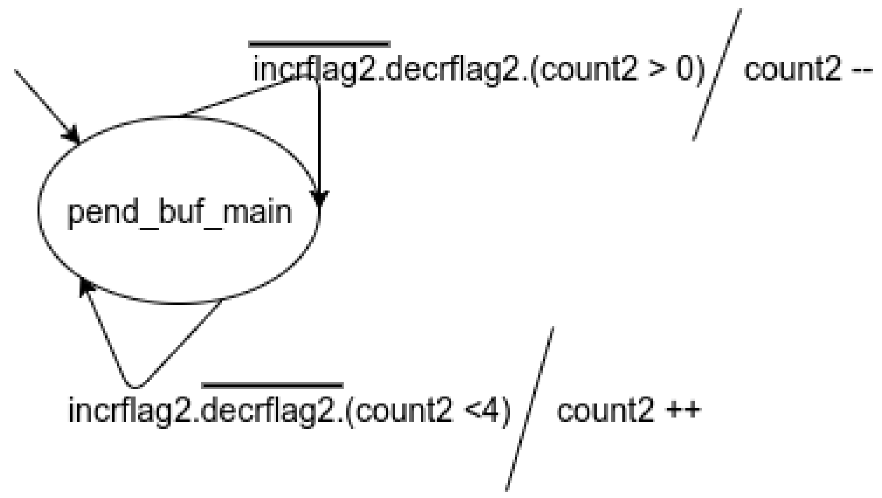

$\overline{\text{incrflag2}}$.decrflag2.(count2 > 0) / count2 --
pend_buf_main
incrflag2.$\overline{\text{decrflag2}}$.(count2 <4) / count2 ++

Figure 5.6: The Pending Buffer Count Module

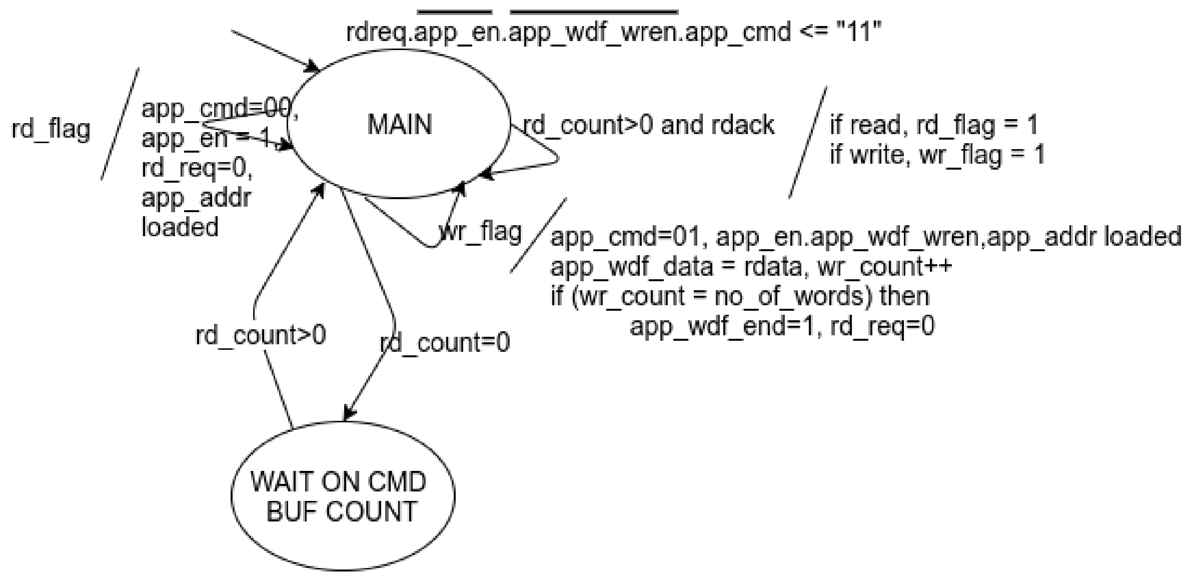

$\overline{\text{rdreq}}$.app_en.$\overline{\text{app\_wdf\_wren}}$.app_cmd <= "11"
rd_flag / app_cmd=00, app_en = 1, rd_req=0, app_addr loaded
MAIN
rd_count>0 and rdack / if read, rd_flag = 1 if write, wr_flag = 1
wr_flag / app_cmd=01, app_en.app_wdf_wren,app_addr loaded app_wdf_data = rdata, wr_count++ if (wr_count = no_of_words) then app_wdf_end=1, rd_req=0
rd_count>0
rd_count=0
WAIT ON CMD BUF COUNT

Figure 5.7: The Request-Send Module

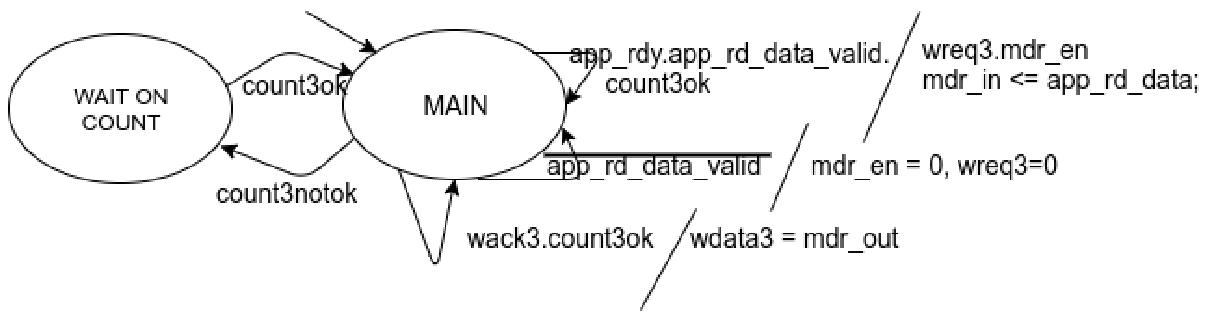

WAIT ON COUNT
count3ok
count3notok
MAIN
app_rdy.app_rd_data_valid. count3ok / wreq3.mdr_en mdr_in <= app_rd_data;
$\overline{\text{app\_rd\_data\_valid}}$ / mdr_en = 0, wreq3=0
wack3.count3ok / wdata3 = mdr_out

Figure 5.8: The Response receive Module

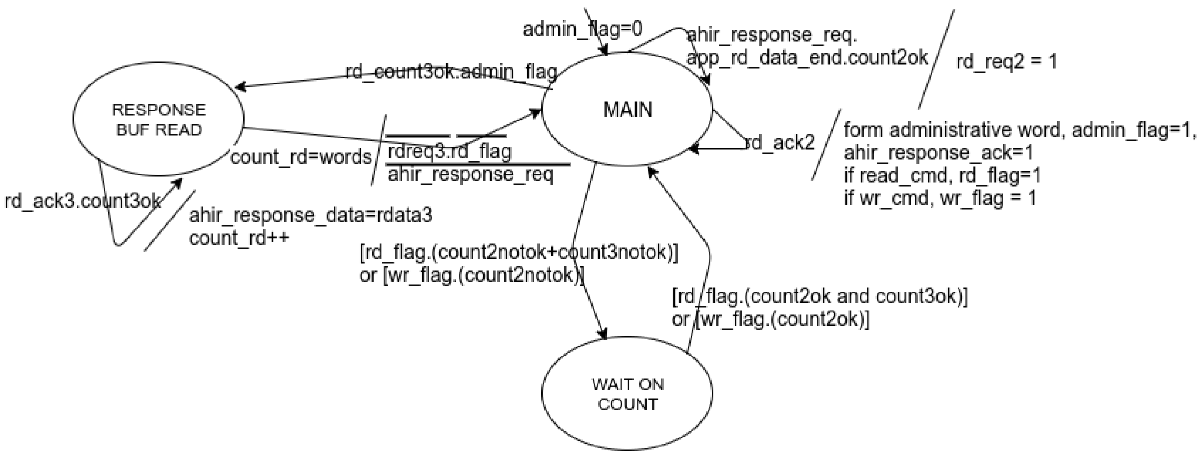


Figure 5.9: The Response Send Module

CHAPTER 6

Test Results

6.0.4 Request Receive

The request receive module has been tested first through two consecutive read commands, one where the count is ok and the next where the count is not ok and it has to wait on cmd buffer first and then on pending buffer. The results are obtained as follows :

Similarly, a write test has been executed .

The results are obtained as follows :

6.0.5 Request Send

Two consecutive read tests followed by a write test have been run.

6.0.6 Response receive

This has been tested by four back to back data words from mig .

6.0.7 Response Send

A read test followed by a write test have been made.

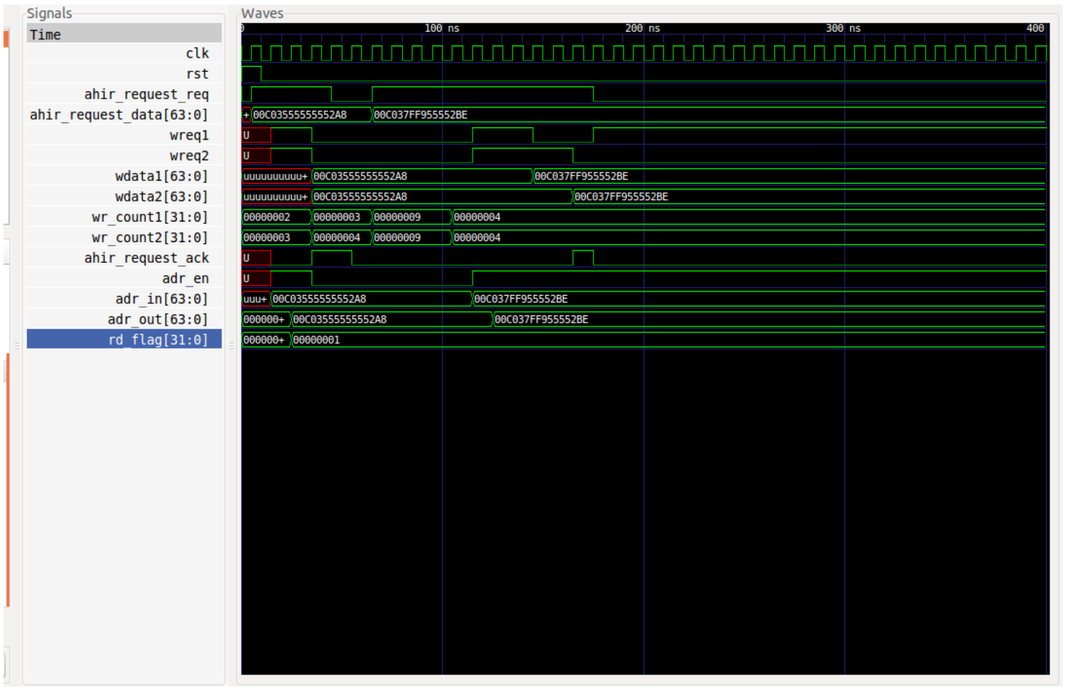


Figure 6.1: Request Receive Read Cmd

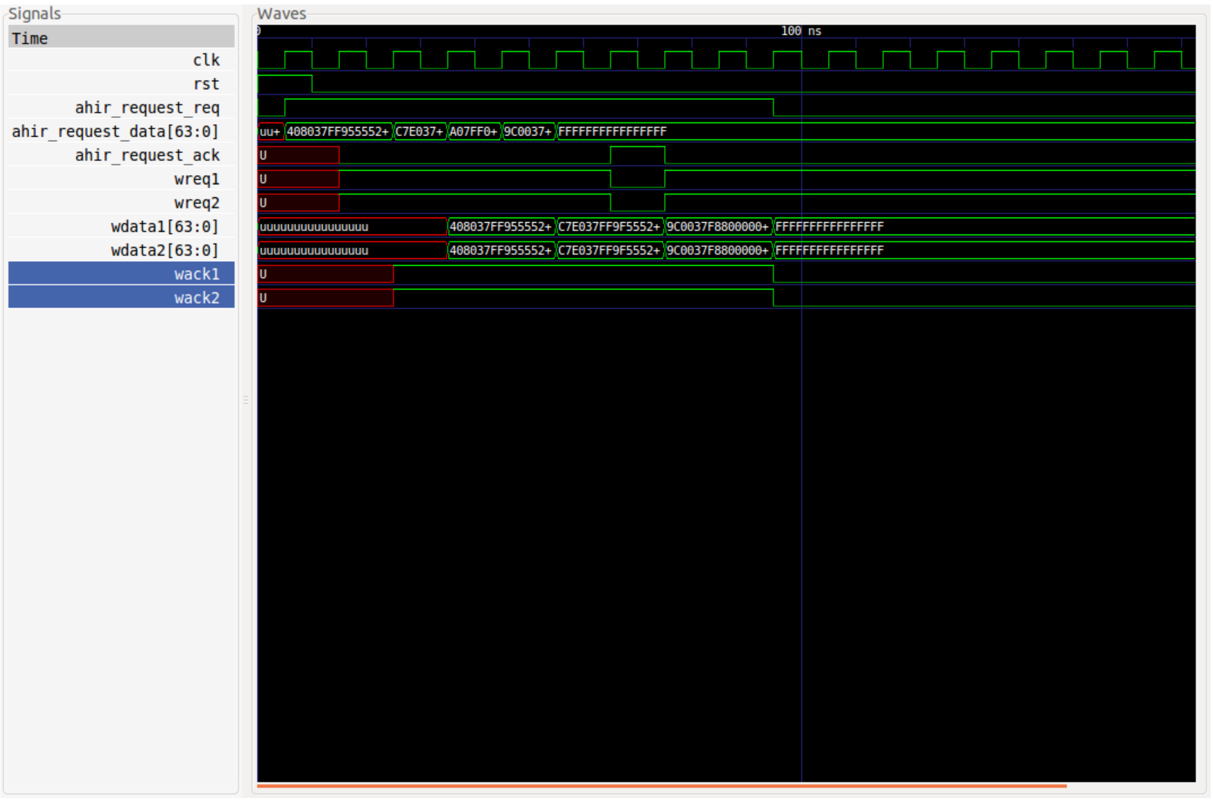


Figure 6.2: Request Receive Write Cmd

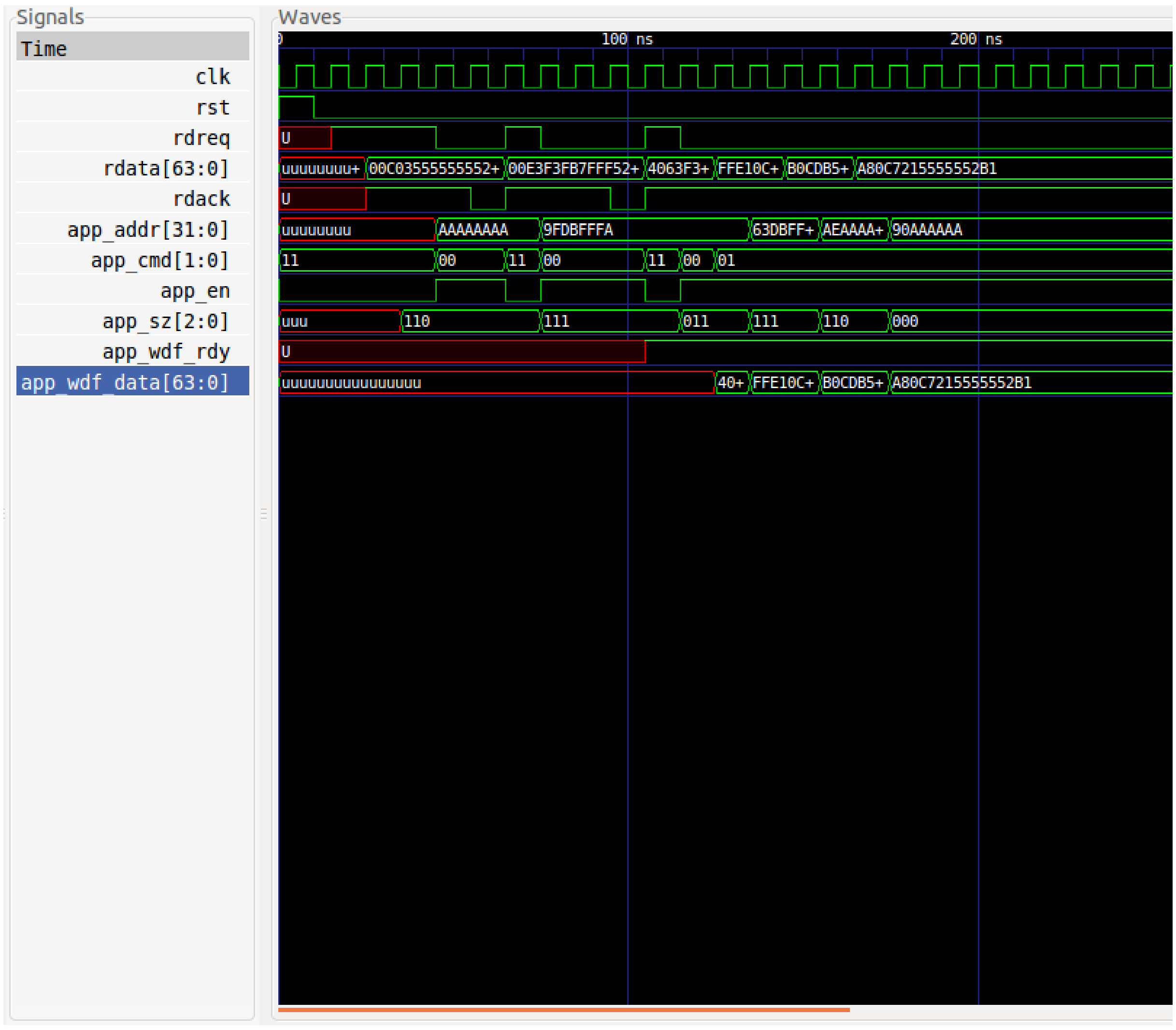


Figure 6.3: Request Send Two Reads and Write Cmd

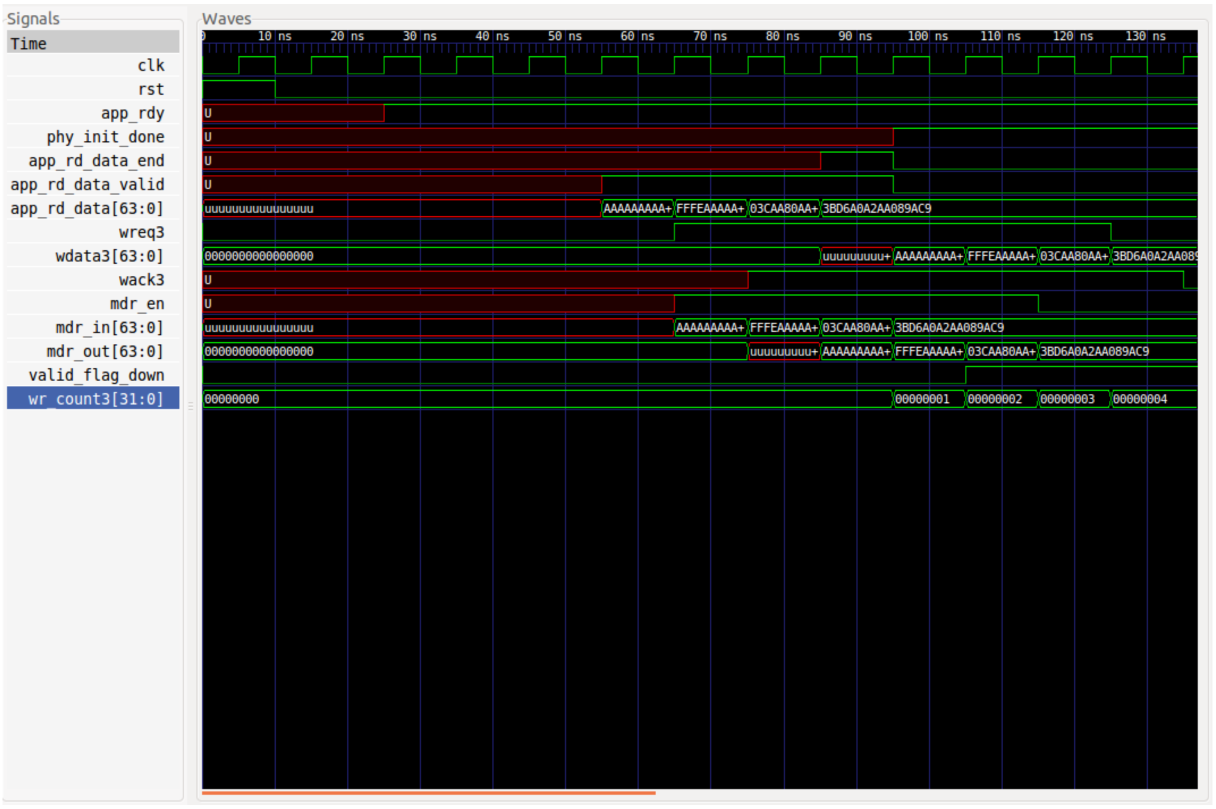


Figure 6.4: Response Receive

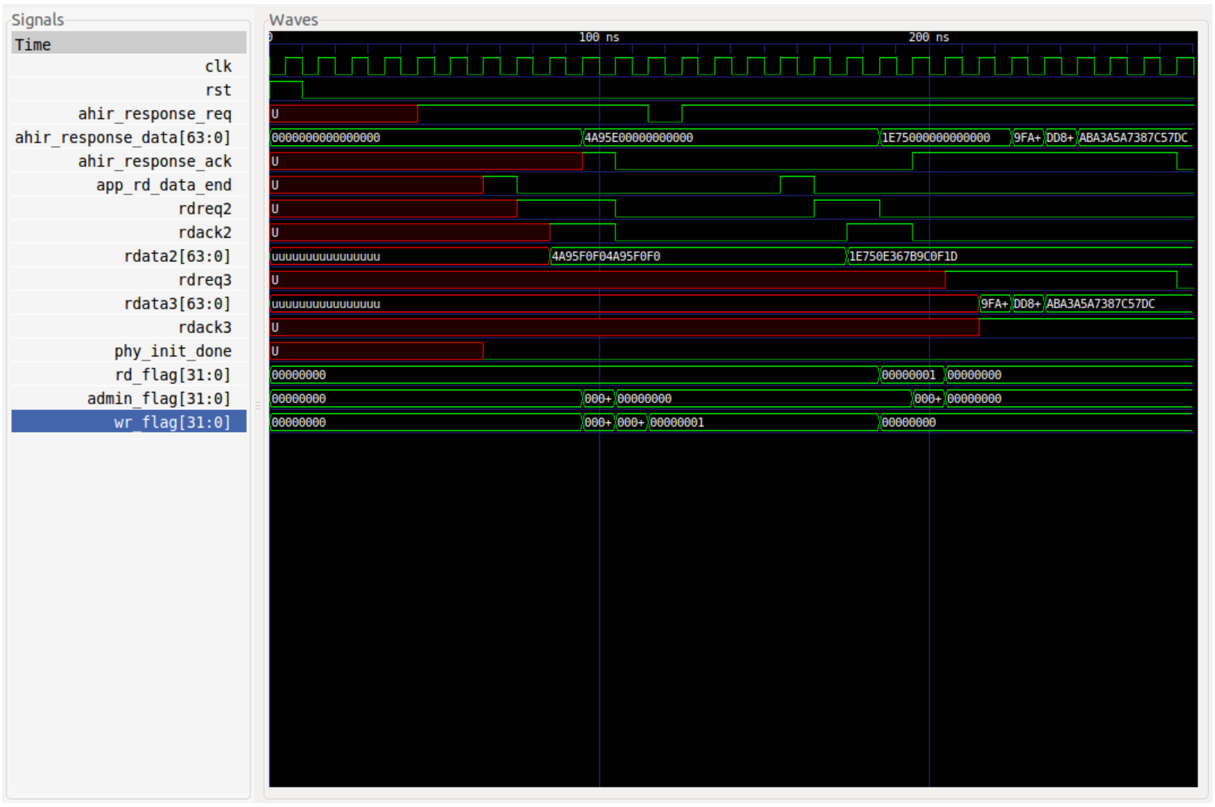


Figure 6.5: Response Read and Write

CHAPTER 7

References

1. ML605 Board and FMC XM104 Connectivity Card IBERT Design: SMA and SATA Interfaces, Xilinx, March 2012(XTP091)
2. Virtex-6 FPGA Memory Interface Solutions, UG406 January 18, 2012, User Guide
3. Virtex-6 FPGA Memory Interface Solutions, DS186 January 18, 2012
4. https://forums.xilinx.com/t5/Memory-Interfaces/MIG-DDR3-example-design-init-calib-complete-never-goes-high/td-p/721154
5. https://www.xilinx.com/support/answers/34319.html
6. https://www.xilinx.com/products/intellectual-property/mig.htm
7. https://www.xilinx.com/products/intellectual-property/mig.htm
8. www.instructables.com/id/Configuring-the-MIG-7-Series-to-Use-the-DDR-Memory/
9. https://www.fpgarelated.com/groups/comp.arch.fpga/kw/MIG.php