Design of a Bridge between AHIR system and DDR3 SDRAM

Swatantara Chakraborty(15307R011)

Indian Institute of Technology,Bombay

Under Prof. Madhav P. Desai

July 4, 2018

Overview

Our aim is to design a bridge module that can directly load and fetch streams of data from the external DDR3 SDRAM in Xilinx cards through the PCIe bus in a RIFFA environment.

Long-term Goal

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 designed a proposed model for a pipelined bridge module .

Understanding the Memory Controller

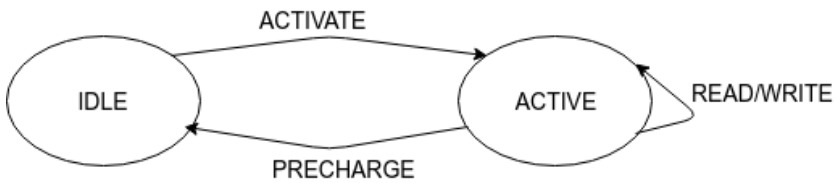
We start the discussion by describing the basics of the environment we are working on.The design is applicable to all Xilinx FPGAs, like VC-709, ML-605, KC-705 etc. We shall demonstrate the external memory in ML605 for example.

In the next 6 slides, we shall familiarise ourselves with the Xilinx Memory Controller Interface and its Operation.

What is DDR3 SDRAM

1. Double data rate type three SDRAM (DDR3 SDRAM) is a type of synchronous dynamic random-access memory (SDRAM) with a high bandwidth ("double data rate") interface.
2. It samples the data at both the rising and falling clock edges making the data sampling rate twice that of the clock.

The SDRAM interface has a separate data and command bus. When data is transferred to or from a bank other banks are activated and precharged (bank preparation).



External Memory in ML605 card

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)

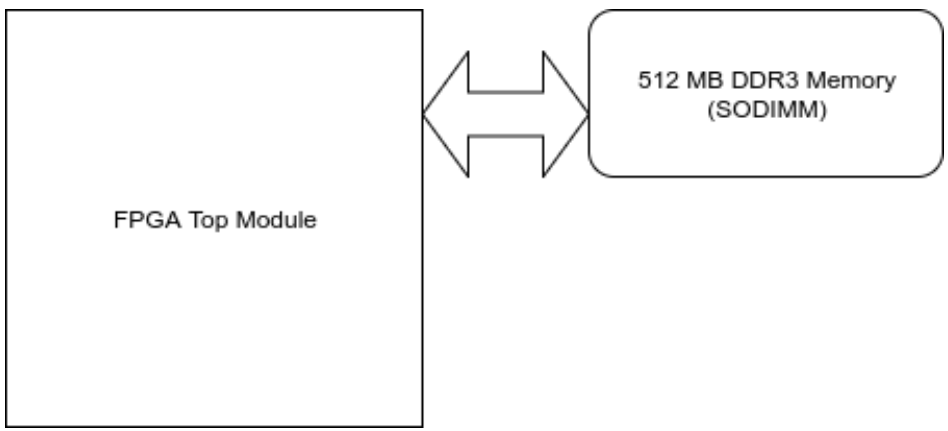
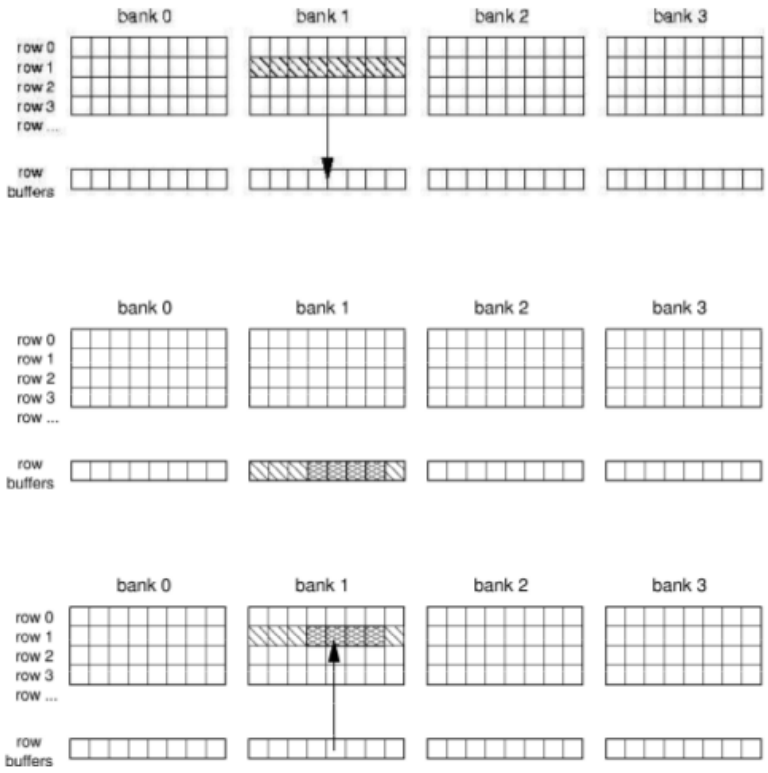


Figure : External Memory in ML605 card

Basic DDR3 SDRAM Architecture: Demo later

1. SDRAMs have a multi-bank architecture and is organized in banks, rows and columns.
2. The requested row is activated and copied to the row buffer of the corresponding bank.
3. Read and/or write bursts are issued to the active row.
4. The row is precharged and stored back into the memory array.



MIG Memory Controller Operation: Demo later

- The front-end:
  1. buffers requests and responses.
  2. provides an interface to the rest of the system.
  3. is independent of the memory type.
- The back-end:
  1. provides an interface towards the target memory.
  2. is dependent on the memory type.

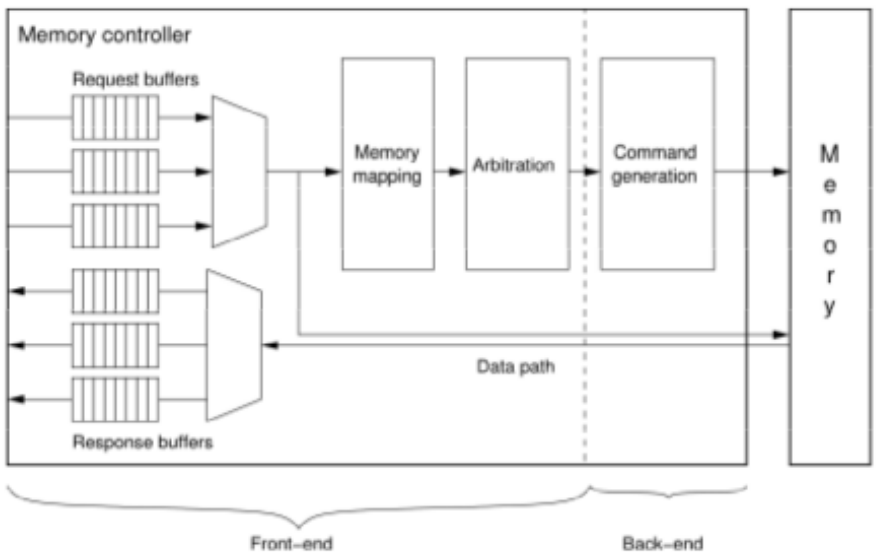


Figure : MIG Memory Controller Functional Blocks

MIG Memory Controller Operation: Functional Blocks Explained

1. The **Memory Map** decodes a memory address into (bank, row, column), decoding is done by slicing the address.
2. The **Arbiter** chooses the order in which requests access memory.
3. The **Command Generator** generates the commands for the target memory, customized for a particular memory generation.

MIG Memory Interface Solutions

The following diagram shows the user interface of the memory controller. It has been reproduced from ug406.pdf (Ref 1).

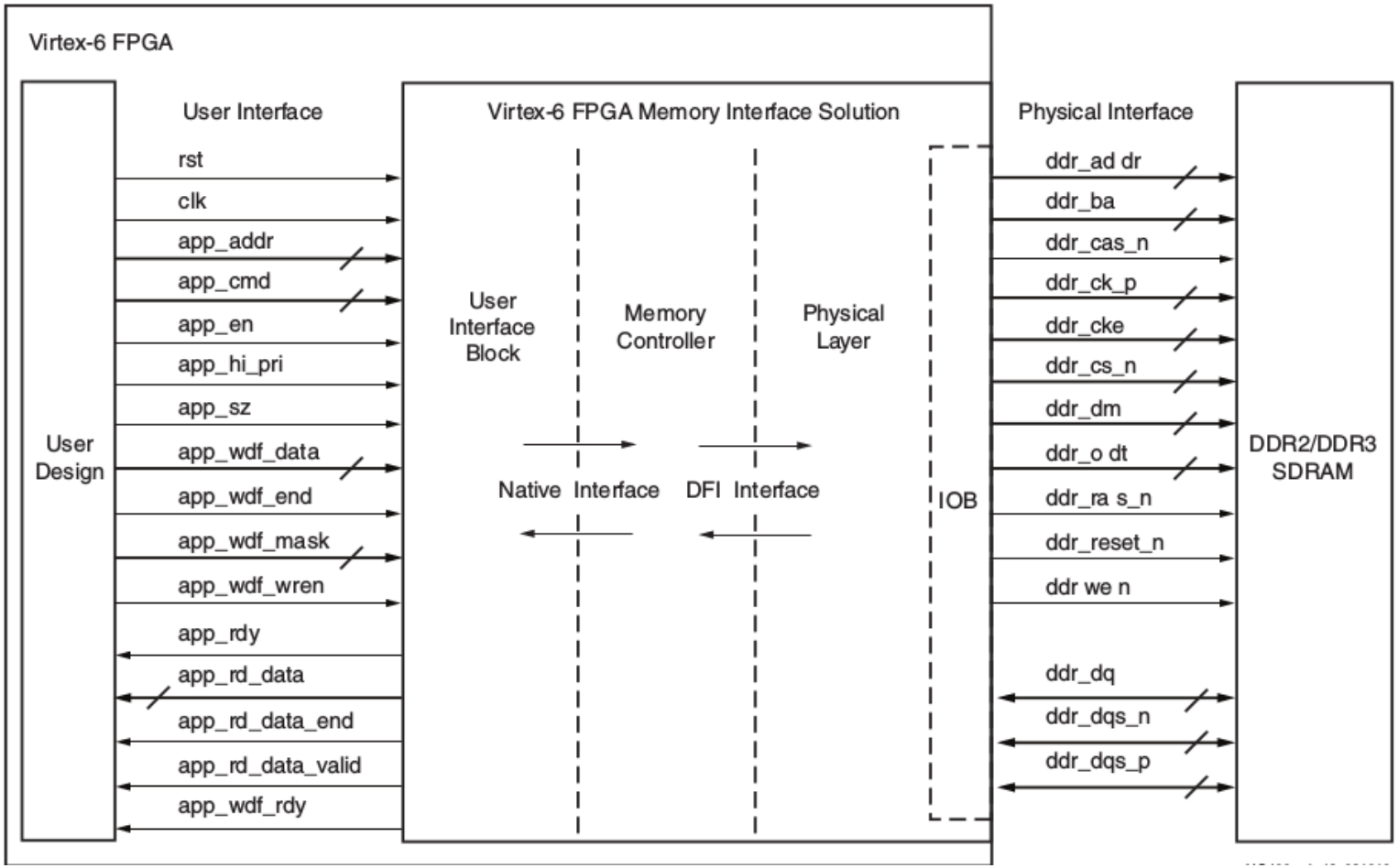


Figure : MIG Virtex 6 Memory Interface Solutions

Demonstrating a MIG Example

In the next 17 slides, we shall generate the Memory Controller core, using Xilinx Coregen, synthesize an example design, programme the ML605 and observe the output using Chipscope Analyzer.

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.(coregen &)
- The design rtl codes can be generated in VHDL or verilog. Here, we have chosen Verilog. Synthesis tool is selected as Xilinx ISE.

Generation of the Memory Controller By MIG

- 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.

Generation of the Memory Controller By MIG

- 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. The layout has been elaborated in the next slide.

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.

Bank Selection Contd.

- **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.

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

The Basic MIG Example-Design

The example-design of the memory controller interface that communicates with the DDR3 SDRAM.

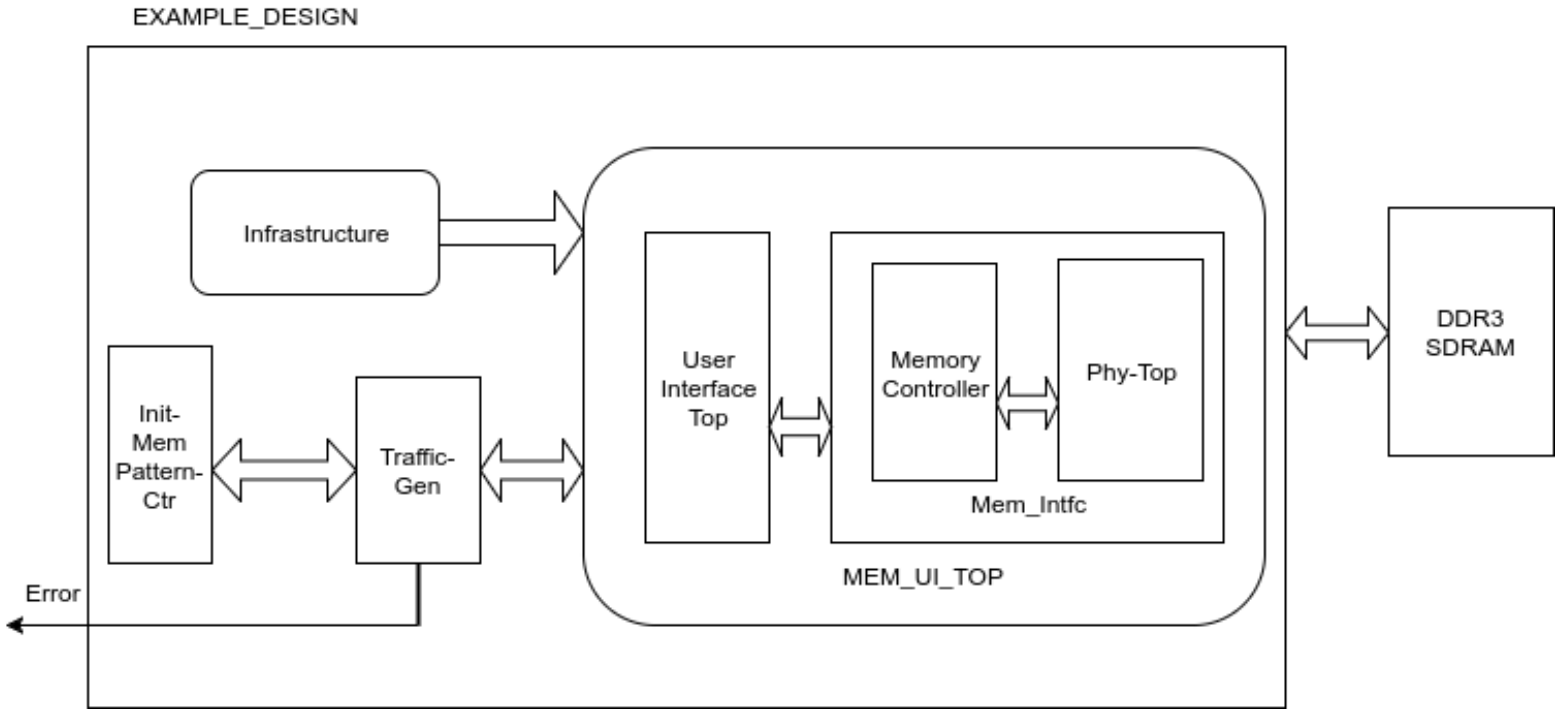


Figure : Block Diagram for the Example Design

Explanation of the MIG Example

- **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.

Example Design Flow : Traffic Generation

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.

Init Memory Pattern Control

The 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.

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 need 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.

Running the Compiled Design : Chipscope Analyzer

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 resultant waveforms have been obtained as illustrated in the next few slides.

Running the Compiled Design : Chipscope Analyzer

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.

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.

Running the Compiled Design : Chipscope Analyzer

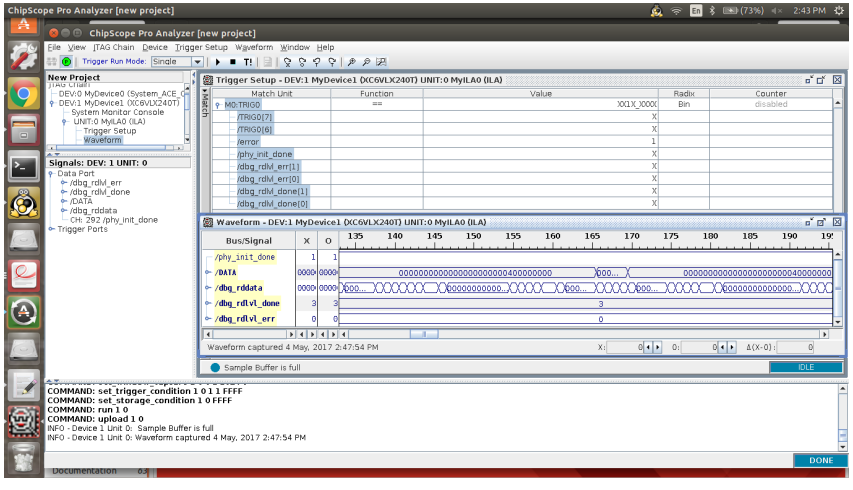
- **dbg_rdlvl_done** : Each bit is driven to a static 1 as each stage of read leveling is completed. The dbg_rdlvl_done[0] signal corresponds to stage 1. If both the read stages are successfully completed, both dbg_rdlvl_done[0] and dbg_rdlvl_done[1] should be one, that is dbg_rdlvl_done as a bit vector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

Running the Compiled Design : Chipscope Analyzer

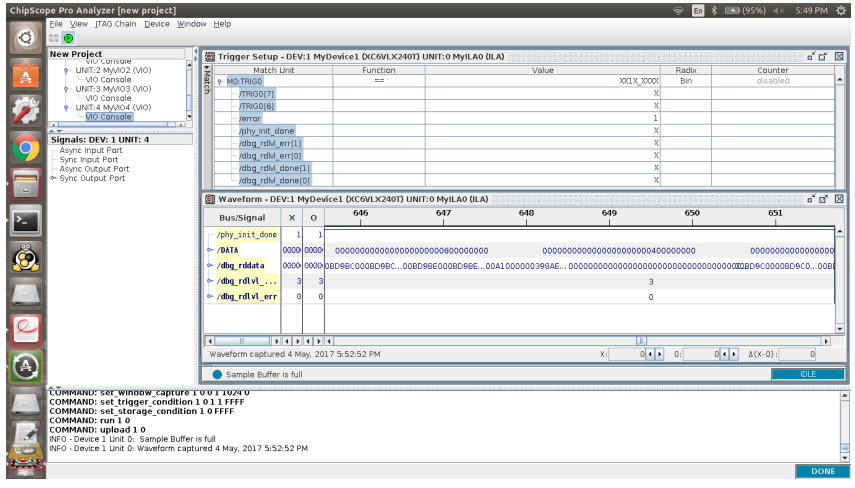
The debug signals to be monitored using Chipscope Analyzer should be included in the example_top.cdc file . Note that the DATA pins of the ILA are not utilised.

Output from Chipscope Analyzer

The dbg_rdlvl_done is 3 and the dbg_rdlvl_err is 0 indicating successful completion.



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



Issues Faced

The mig version 33 is directly compatible with the ML605 Virtex 6 memory interface solutions. But as we have used mig38, certain changes have to be made in the example design rtl. The modified design is provided alongside this document. Some of the modifications are:

- CLKFBOUT_MULT_F = 6
- OUTPUT_DRV to HIGH
- nDQS_COLx
- DQS_LOC_COLx
- RST_ACT_LOW = 0 (was 1)

Quick Review of the Steps

- Open the terminal. Make a directory, say, July4_mig. Invoke coregen, coregen&.
- File –¿ New Project –¿ Choose July4_mig –¿ Save.
- Choose Family: Virtex6, Device: xc6vlx240t, Package: ff1156, Speed Grade: -1
- Go to **Generation** option. Make Design Entry Verilog. Click on Ok. Window closes.
- Right Click on Mig Virtex6 & Spartan 6 from the drop-down menu on the left. Select Customize and Generate..
- Click on Next for the next 4 windows. In the 5th window, select Memory type: SODIMMs, Ordering: Strict.
- Click on Next. Make Debug: ON. Go to Next.
- Select New Design for pin layout. Go to Next.

Quick Review of the Steps

- Click on Deselect Banks. Now, select Bank36 : Address/Control, Bank26: Data, Bank25: Data, Bank35: Data,Bank34: System Clock. From the middle column on the top, select Master Bank 25.
- Click on Next for two successive windows. Click on Decline. Click on Next for the next 2 windows.
- Click on Generate.
- Go back to the command prompt. Go to ml605_modified_main/mig39/example_design/par and run the ise_flow.sh, i.e., ./ise_flow.sh
- Invoke Chipscope Analyzer, analyzer & at the command prompt.

Quick Review of the Steps

- Click on Open Cable/Search JTAG Chain button (top left).
- Click on OK on the MYDevice prompt.
- Click on Device from the top bar, Go to Device1 and click on Configure.
- In the new window, click on Select New File. Select July4_mig/ml605_modified_main/mig39/example_design/par/exampl
- After it has finished, click on the T button on the top to trigger the chipscope pro.
- Click on the Apply setting and arm trigger button next but one to the T button.
- Check for the debug waveforms.

Quick Review of the Steps

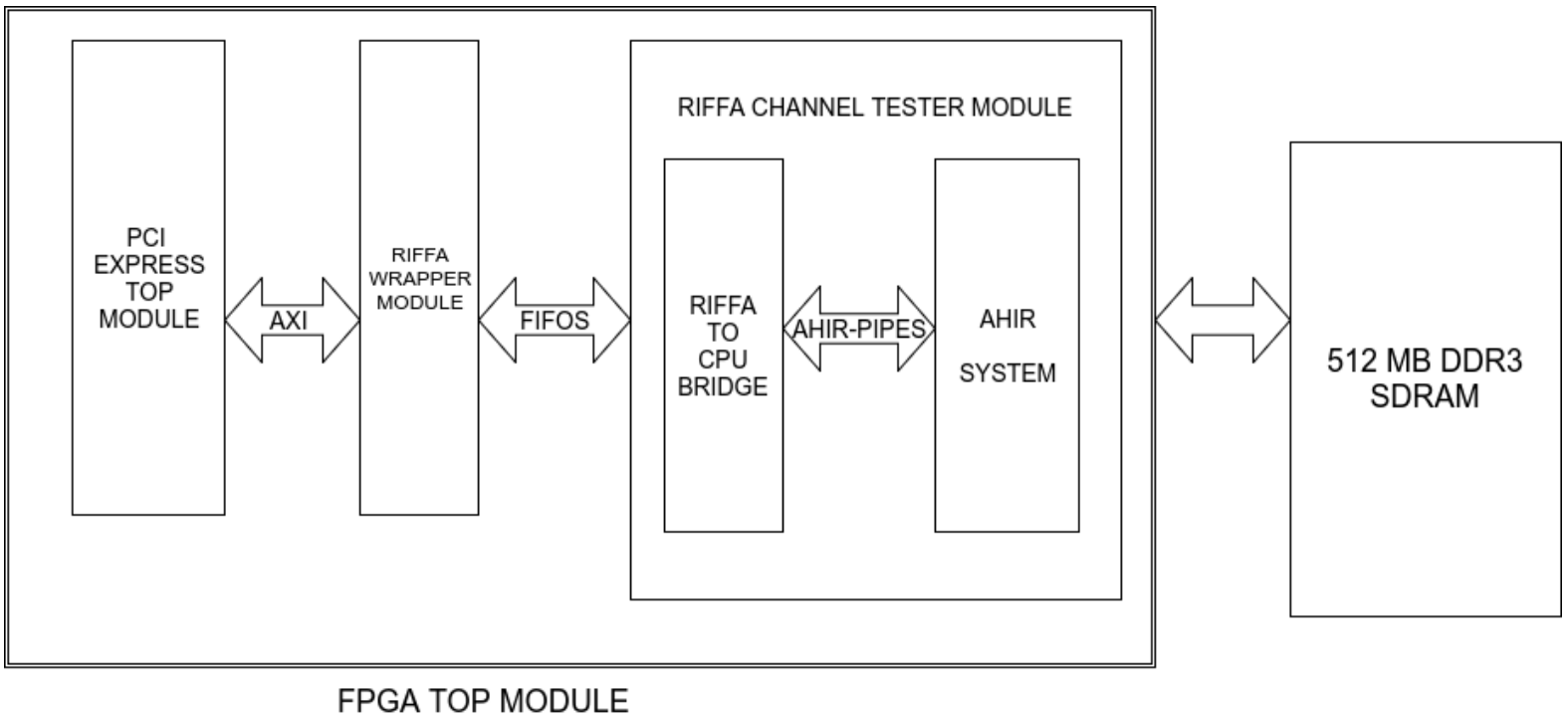
- To adjust the PRBS (pseudo random bit sequence), go to the VIO console on Unit 4 (top left) and click.
- In the list of debug signals, make any one 1 to induce an error. Click on the U button on top to update the change.
- Now again , click on T and arm trigger like before.
- Check dbg_rddata for the PRBS. You may have to zoom into the waveforms.

Bridge Design

In the next slides, we shall discuss the pipelined bridge design.

Current Status

The FPGA-top module receives/sends data stream from/to the CPU through the PCI-e . The RIFFA-Wrapper interacts with the PCI-e and sends the data to the channel tester through FIFOs. The channel tester has a riffa-to-cpu-bridge which communicates with the AHIR-system. The AHIR-system consists of the user-generated application. The following block diagram elaborates the same.



Target Status

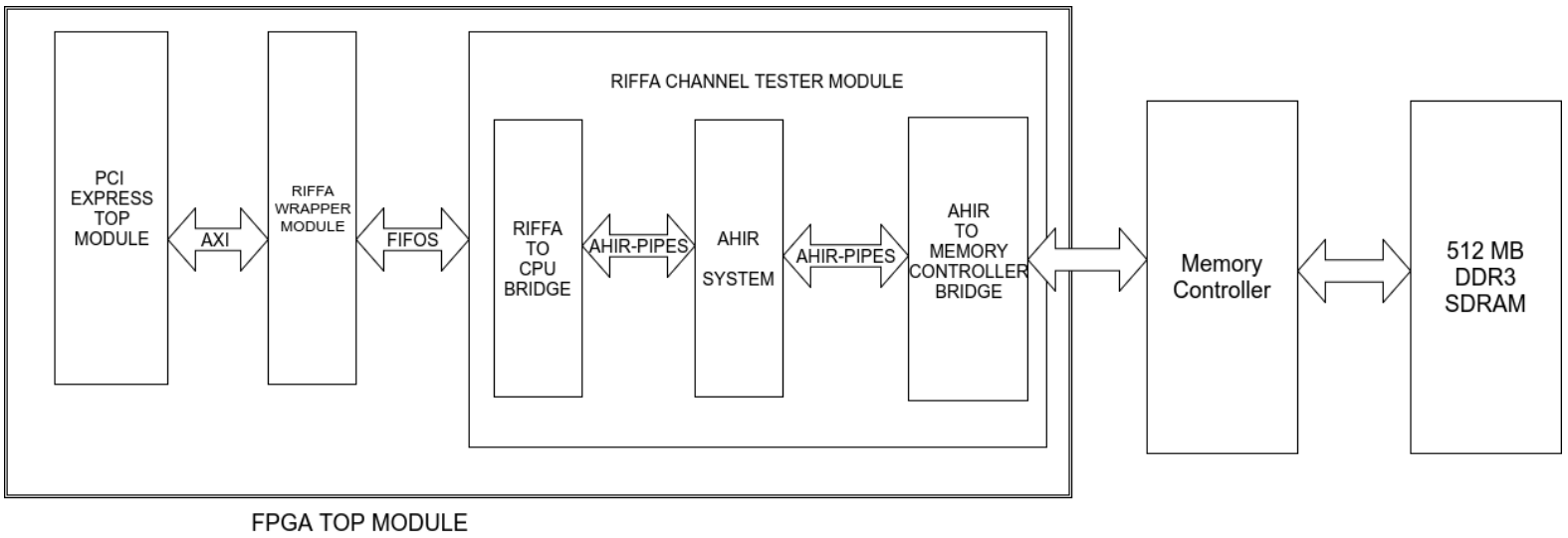


Figure : Target Status of the Host Environment

AHIR-MIG Interface

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.

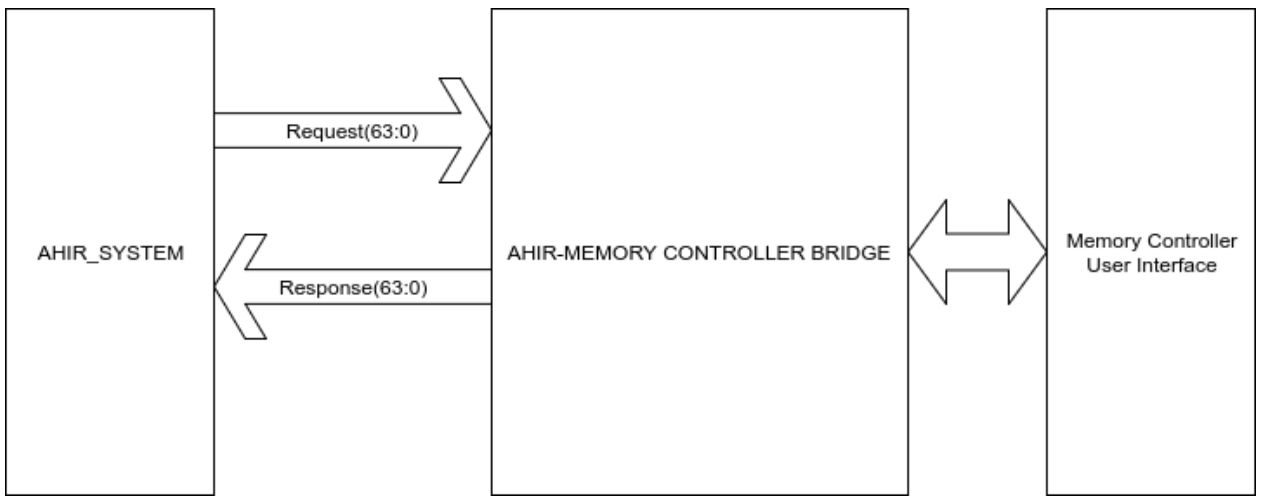
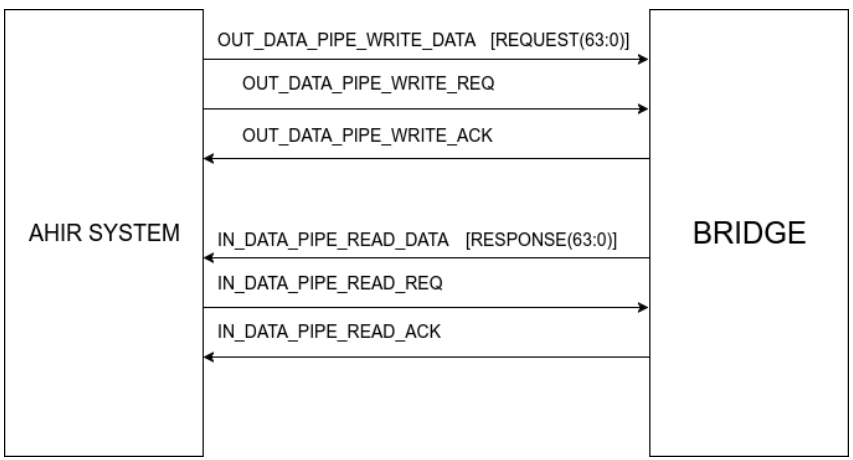
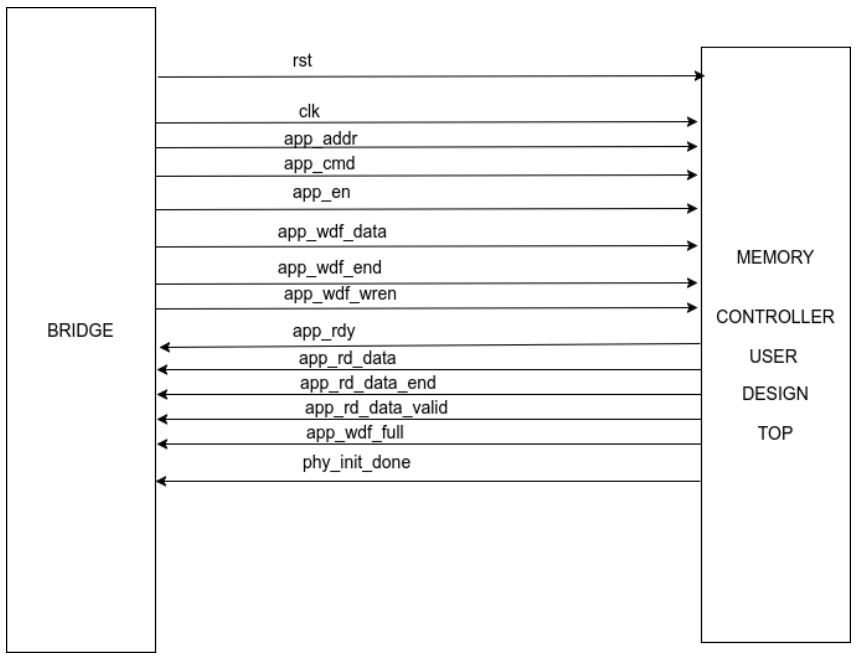


Figure : AHIR-MC Bridge

AHIR-Bridge Interface and Bridge-Memory Controller Interface





Request and Response Packet Formats

cmd (8 bits)	no of words (3 bits)	Request Id (8 bits)	Start Address (32 bits)	checksum (8 bits)	Unused Bits
Data words in case of write					
Data Words in case of write					

cmd (8 bits)	no of words (3 bits)	Response Id (8 bits)	Error Bits (2 bits)	checksum (8 bits)	Unused Bits
Data words in case of read					
Data Words in case of read					

Bridge Overview

A four-stage pipelined 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.

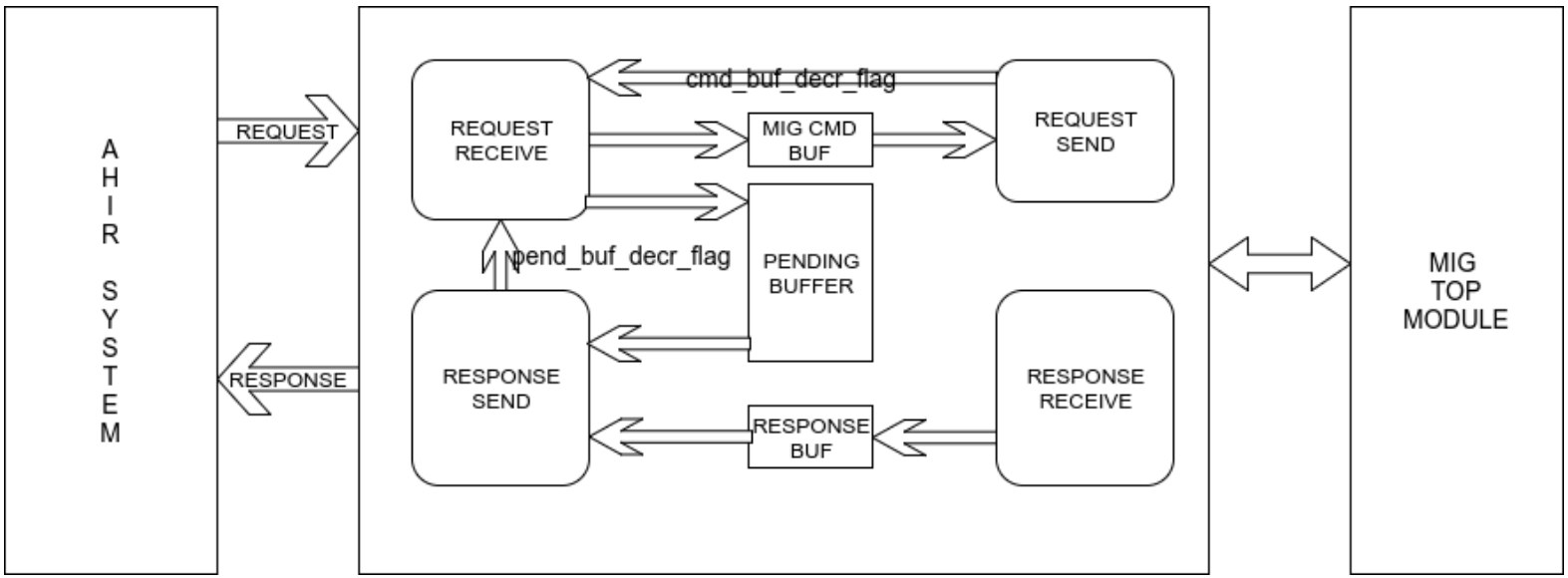
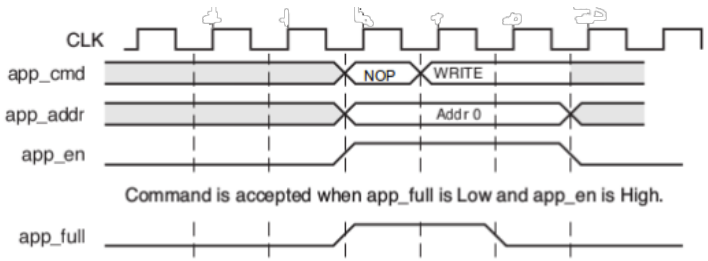


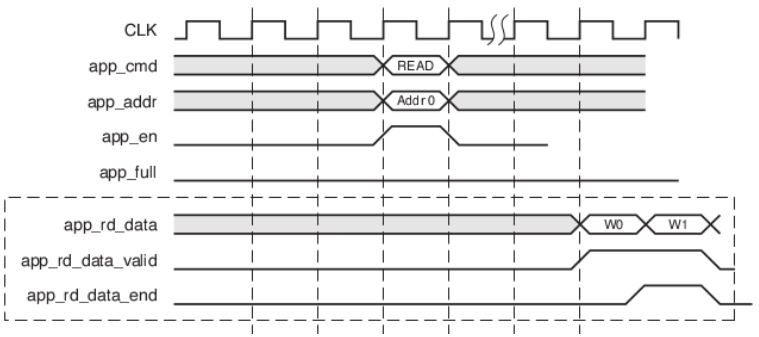
Figure : Bridge Overview

Memory Interface Generator: Timing Diagrams

The command path timing diagram is given below:



This is the timing diagram for MIG data path, a read command is being executed here.



Design of the Request Module

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.

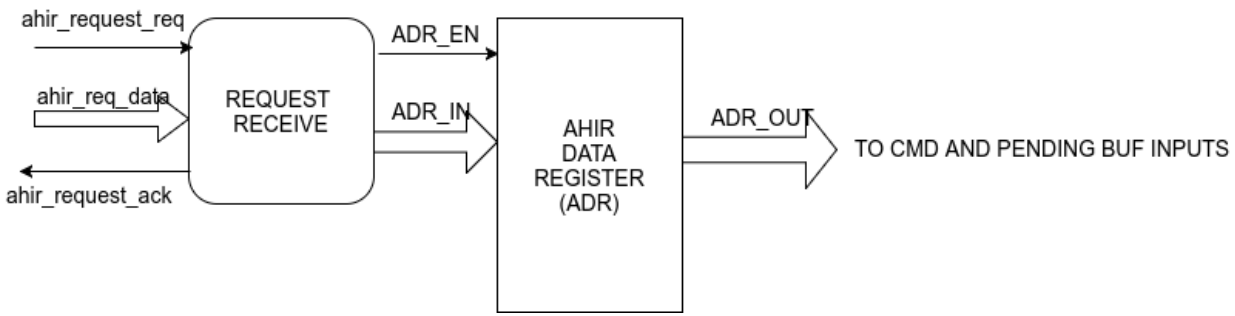


Figure : Ahir Data Register

Request Receive Module

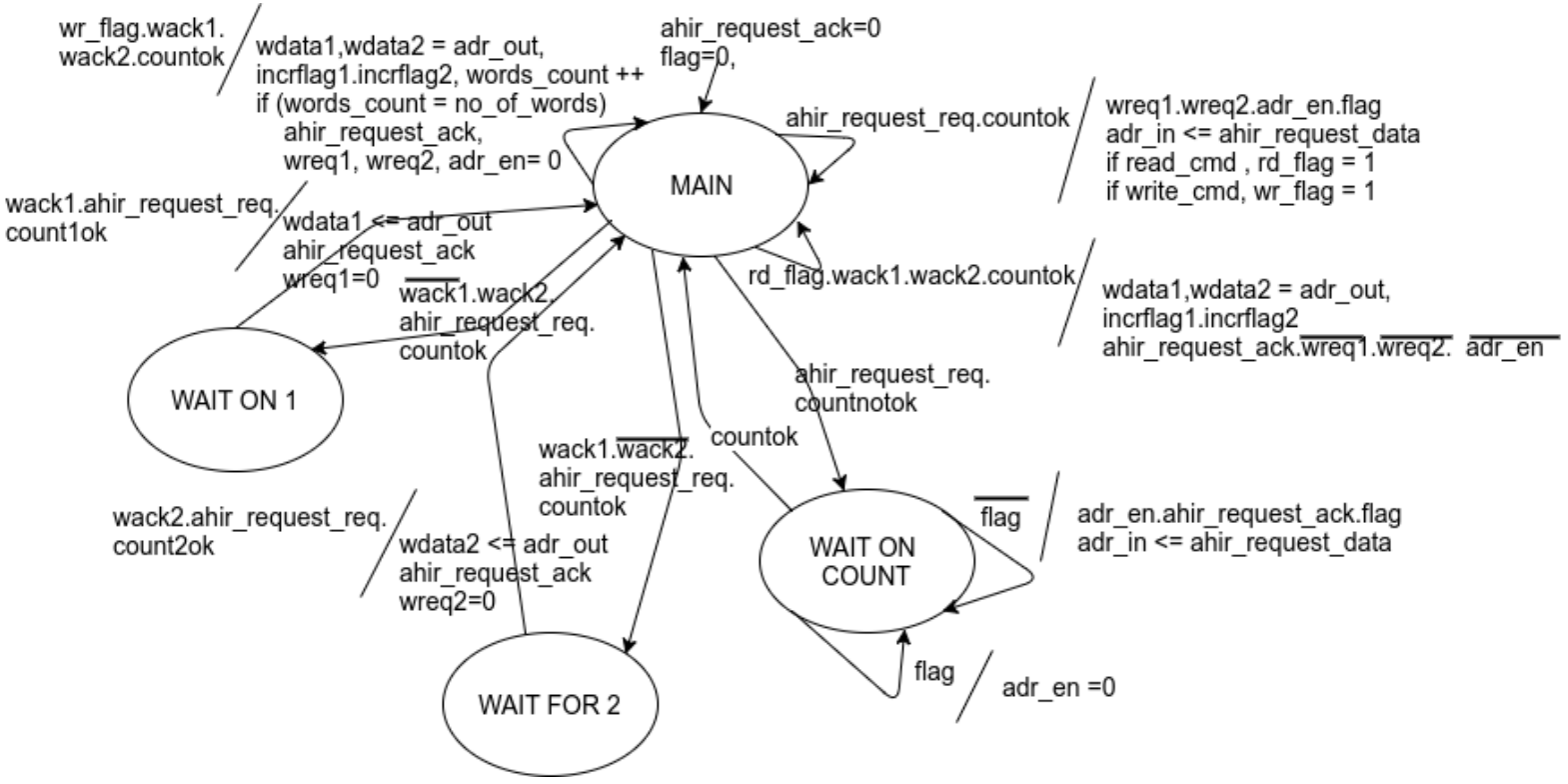


Figure : The Request Receive Module

Deadlock Avoidance

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.

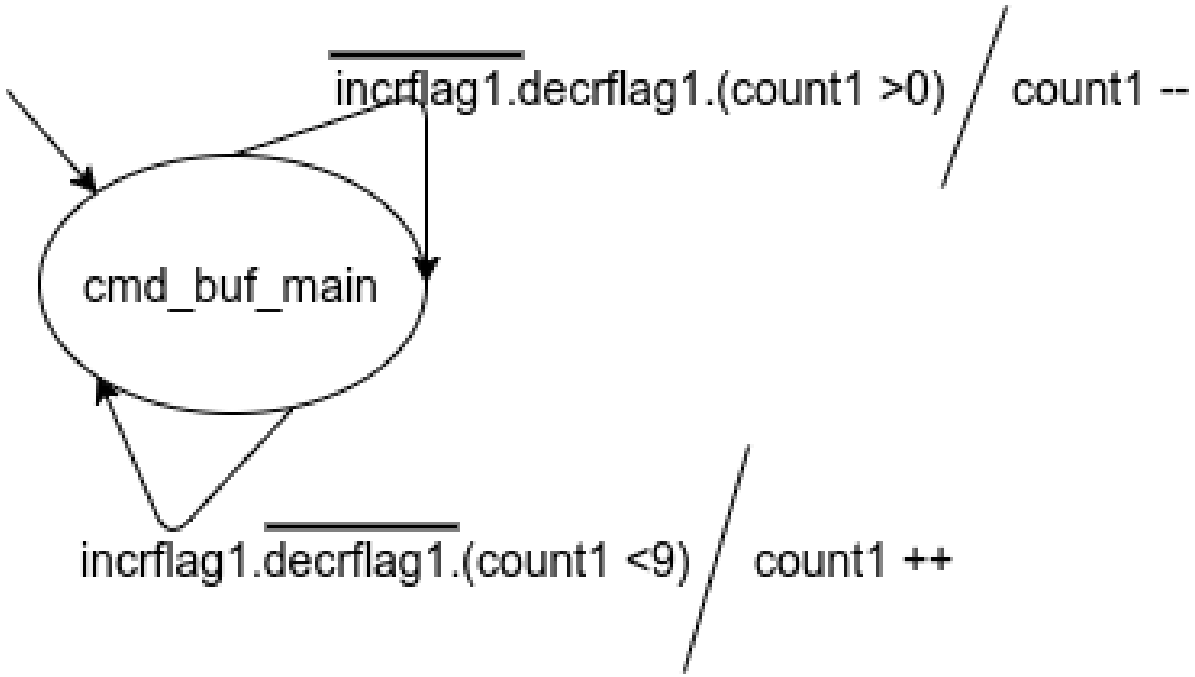


Figure : The Cmd Buffer Count Module

Request Send Module

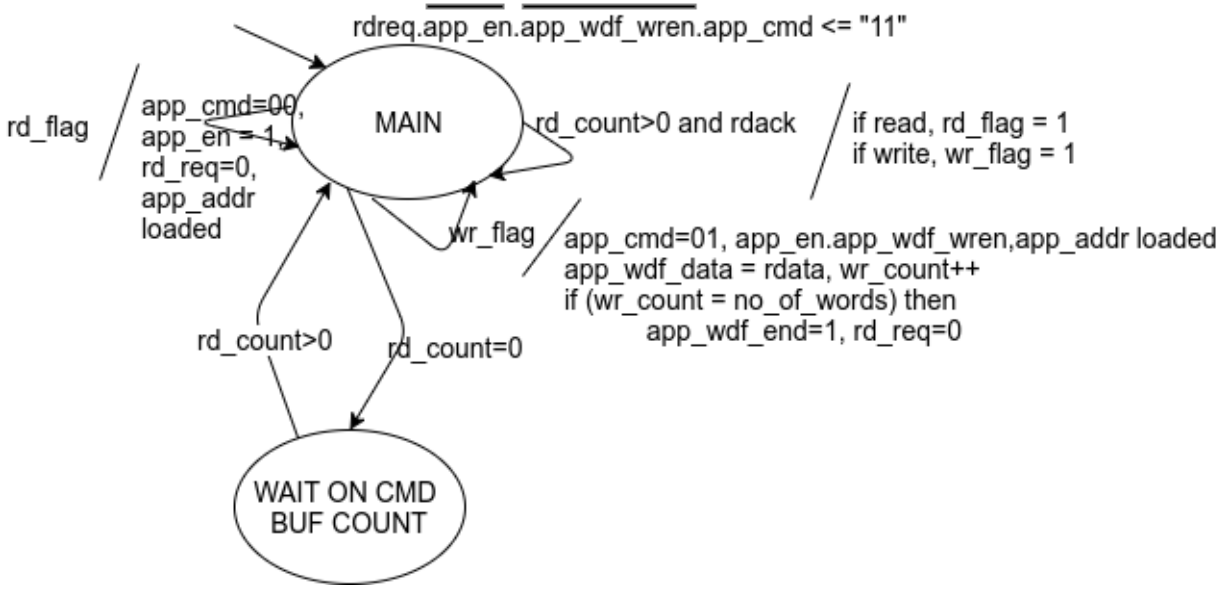


Figure : The Request-Send Module

Response Receive

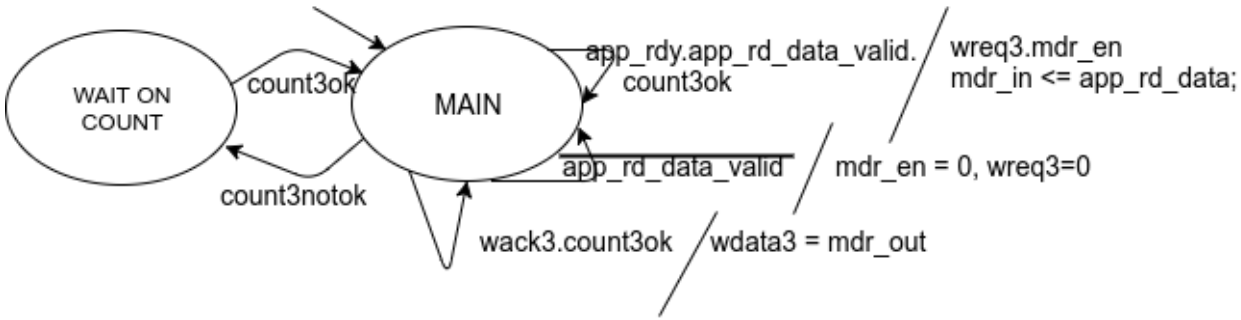


Figure : The Response receive Module

Response Send

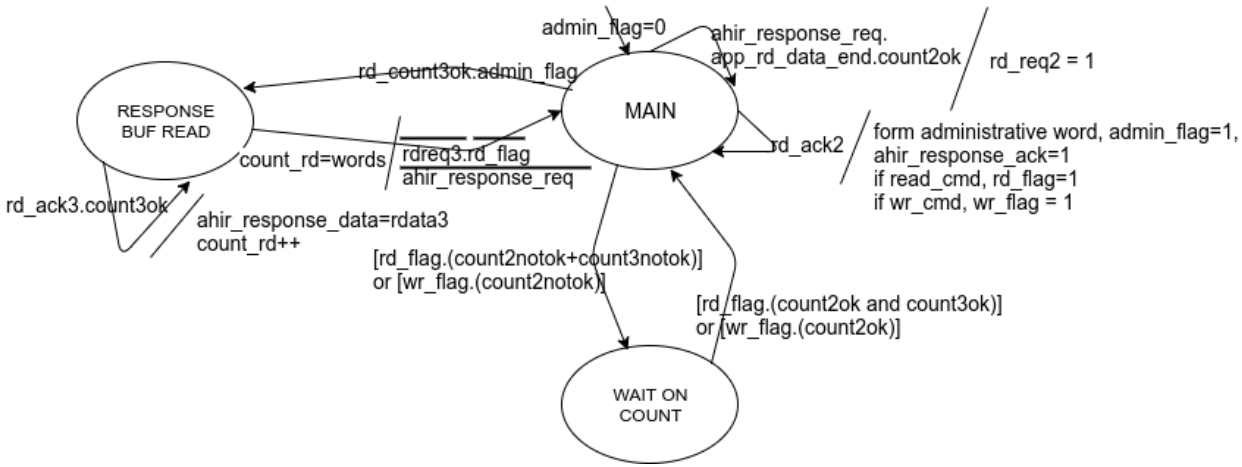


Figure : The Response Send Module

References

1. Virtex-6 FPGA Memory Interface Solutions User Guide, UG406 June 22, 2011
2. ML605 Board and FMC XM104 Connectivity Card IBERT Design: SMA and SATA Interfaces, Xilinx, March 2012(XTP091)
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