

INDIAN INSTITUTE OF  
TECHNOLOGY, BOMBAY

**Design of a Bridge between  
RIFFA Channel Tester Module  
and the DDR3 SDRAM in  
ML605**

*Swatantara Chakraborty*  
*15307R011*

supervised by  
Prof. Madhav P. Desai

March 2, 2018

### **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 ML605 card 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

|          |                                                           |          |
|----------|-----------------------------------------------------------|----------|
| <b>1</b> | <b>Description of the Host Platform</b>                   | <b>3</b> |
| <b>2</b> | <b>Design of the AHIR-MIG Bridge Module</b>               | <b>5</b> |
| 2.1      | Parameter Specifications for the Current Design . . . . . | 5        |
| 2.2      | Description of the Request Packet . . . . .               | 7        |
| 2.3      | Description of the Response Packet . . . . .              | 8        |
| 2.4      | Proposed Design of the Bridge . . . . .                   | 9        |
| 2.4.1    | Design of the intermediate buffers . . . . .              | 9        |
| 2.4.2    | Design of the Request Module . . . . .                    | 9        |
| 2.4.3    | Design of the Response Module . . . . .                   | 12       |

# List of Figures

|      |                                                          |    |
|------|----------------------------------------------------------|----|
| 1.1  | The Current Status of the Host Environment . . . . .     | 3  |
| 1.2  | Target Status of the Host Environment . . . . .          | 4  |
| 2.1  | AHIR-MC Bridge . . . . .                                 | 6  |
| 2.2  | AHIR-MC Bridge . . . . .                                 | 6  |
| 2.3  | AHIR-MC Bridge . . . . .                                 | 7  |
| 2.4  | A Request Packet . . . . .                               | 7  |
| 2.5  | A Response Packet . . . . .                              | 8  |
| 2.6  | Bridge Overview . . . . .                                | 9  |
| 2.7  | Bridge Overview . . . . .                                | 9  |
| 2.8  | The Request Receive and Request Send Modules . . . . .   | 10 |
| 2.9  | The Request-Receive FSM . . . . .                        | 11 |
| 2.10 | The Request-Send FSM . . . . .                           | 12 |
| 2.11 | The Response Receive and Response Send Modules . . . . . | 12 |
| 2.12 | The Response receive Module . . . . .                    | 13 |
| 2.13 | The Response Send Module . . . . .                       | 13 |

# Chapter 1

## Description of the Host Platform

We start the discussion by describing the basics of the environment we are working on.

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

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.

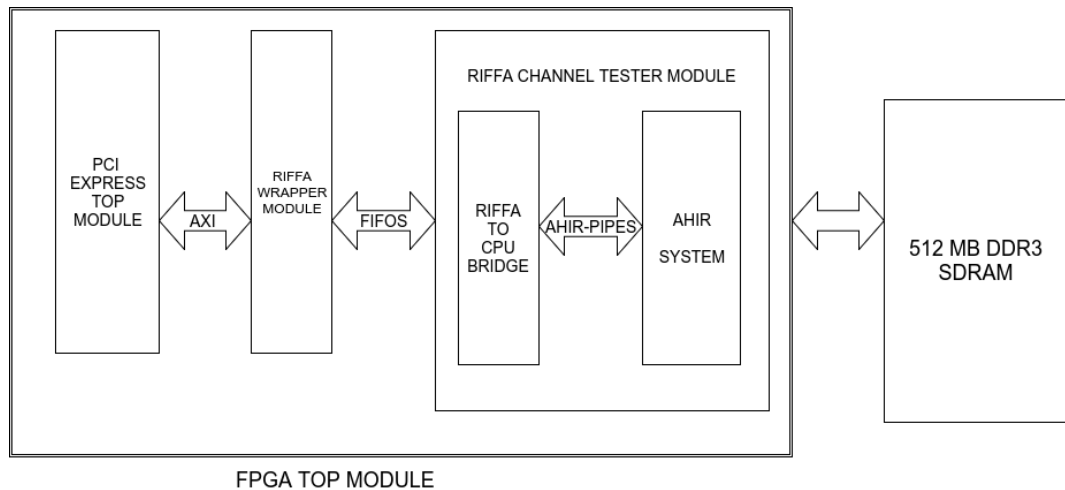


Figure 1.1: The Current Status of the Host Environment

Our target is to devise a direct communication between the AHIR System

and the DDR3 SDRAM.

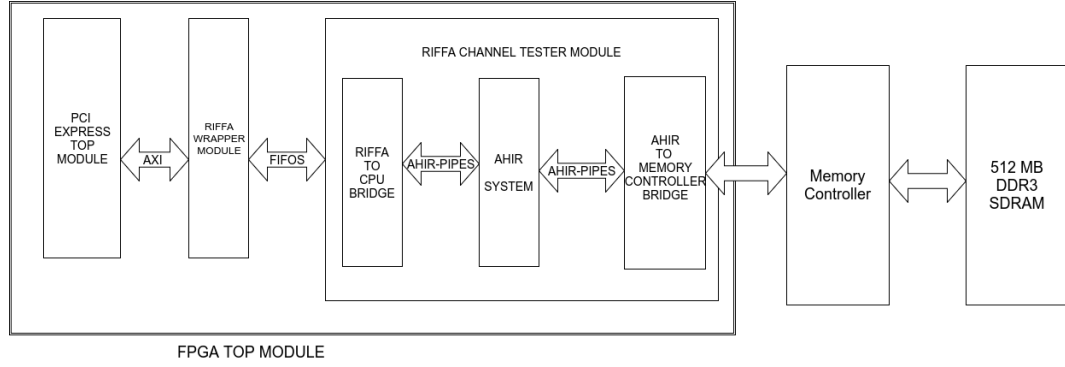


Figure 1.2: Target Status of the Host Environment

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.

## Chapter 2

# 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)

### 2.1 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.

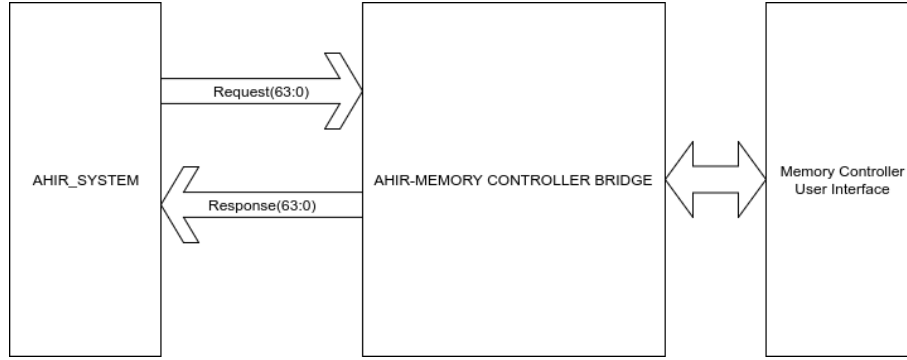


Figure 2.1: AHIR-MC Bridge

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

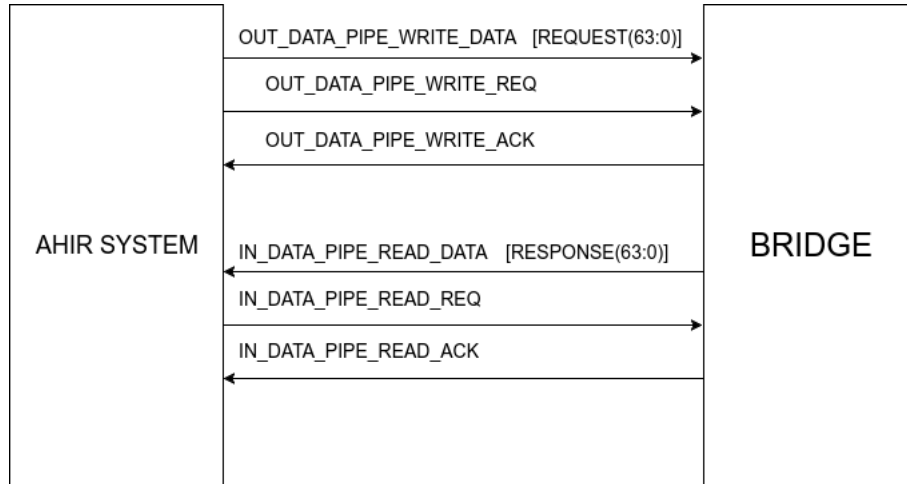


Figure 2.2: AHIR-MC Bridge

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



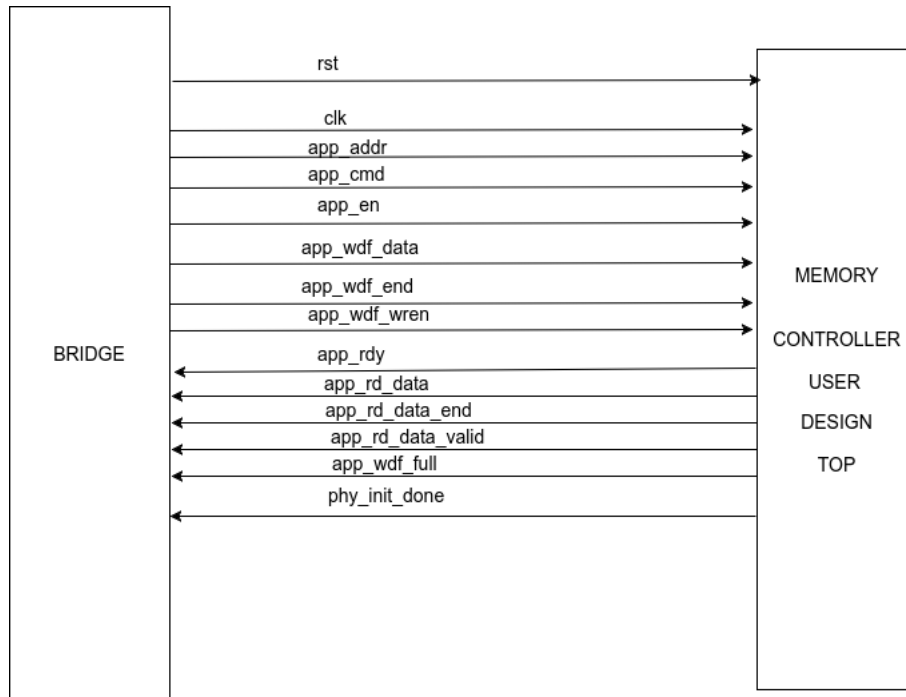


Figure 2.3: AHIR-MC Bridge

## 2.2 Description of the Request Packet

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

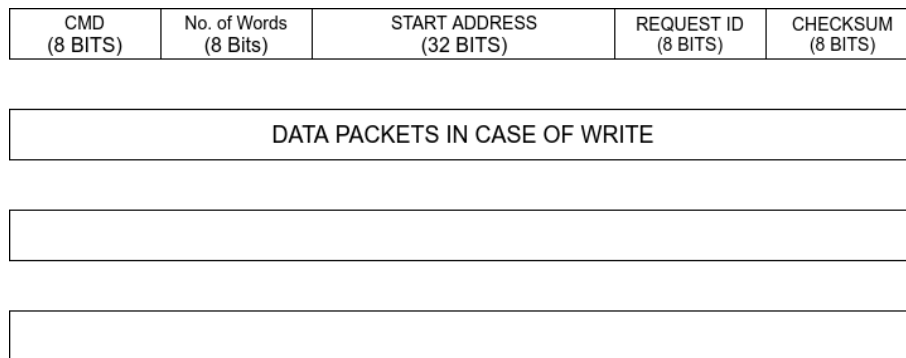


Figure 2.4: A Request Packet

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.*

## 2.3 Description of the Response Packet

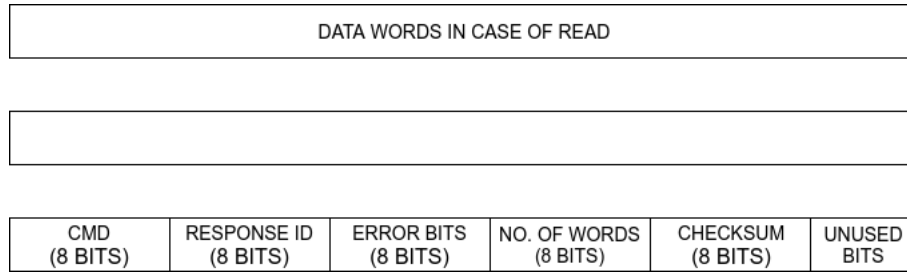


Figure 2.5: A Response Packet

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.

## 2.4 Proposed Design of the Bridge

A four-phase 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.

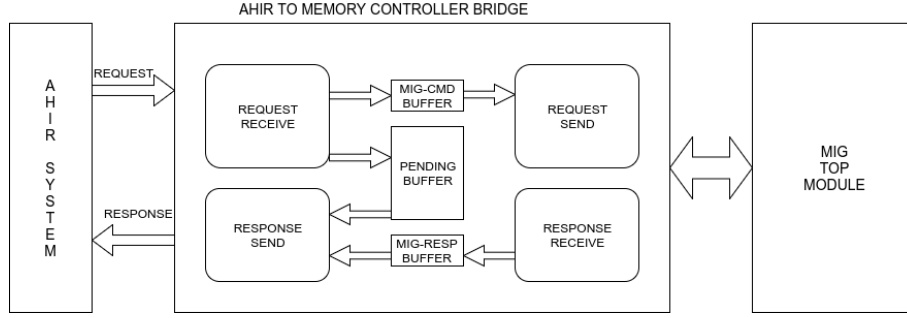


Figure 2.6: Bridge Overview

### 2.4.1 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 8 staged. All these three buffers are PIPO structures and the output is multiplexed as shown below :

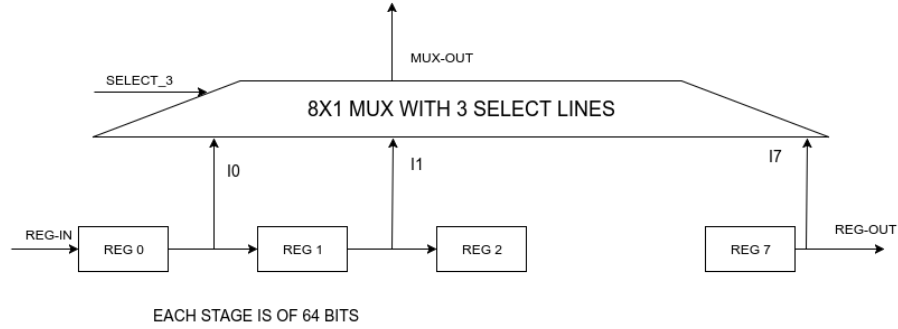


Figure 2.7: Bridge Overview

### 2.4.2 Design of the Request Module

The request-module interface is shown below:

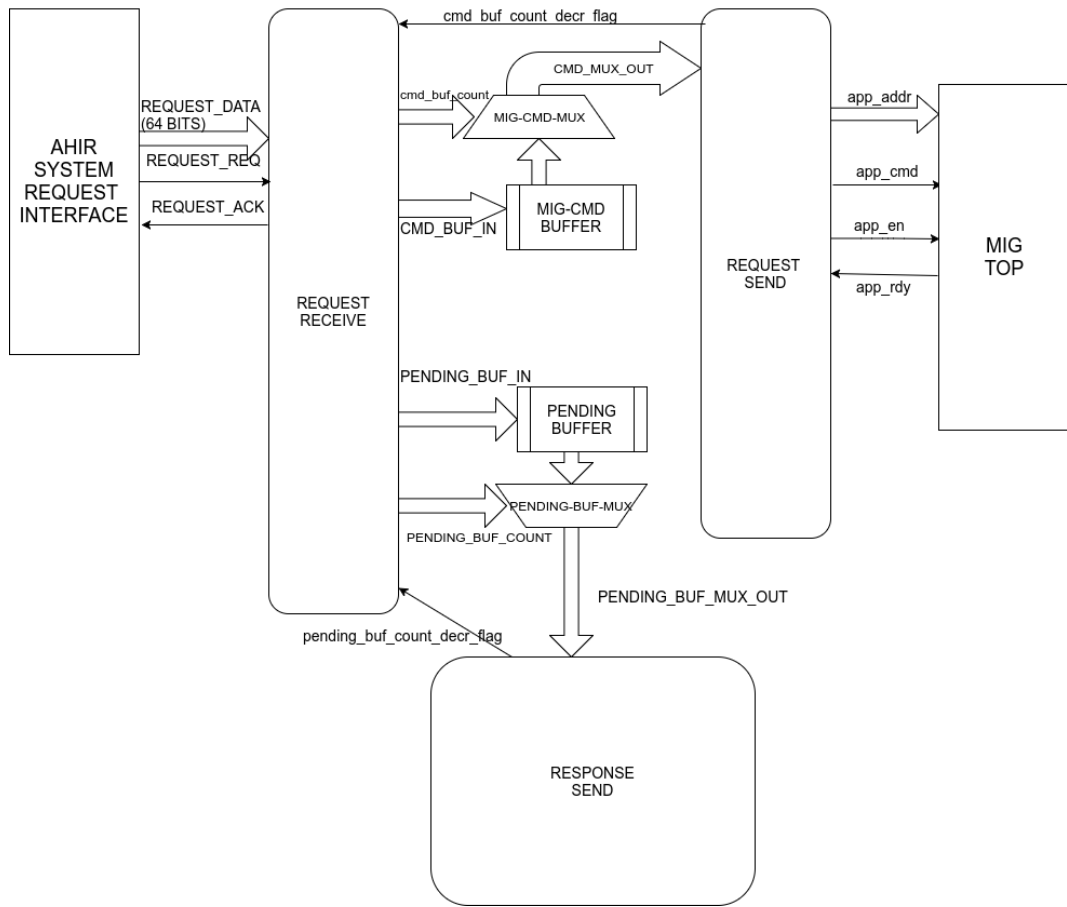


Figure 2.8: The Request Receive and Request Send Modules

The above block diagram operation is explained below: The request-receive module operates as follows:

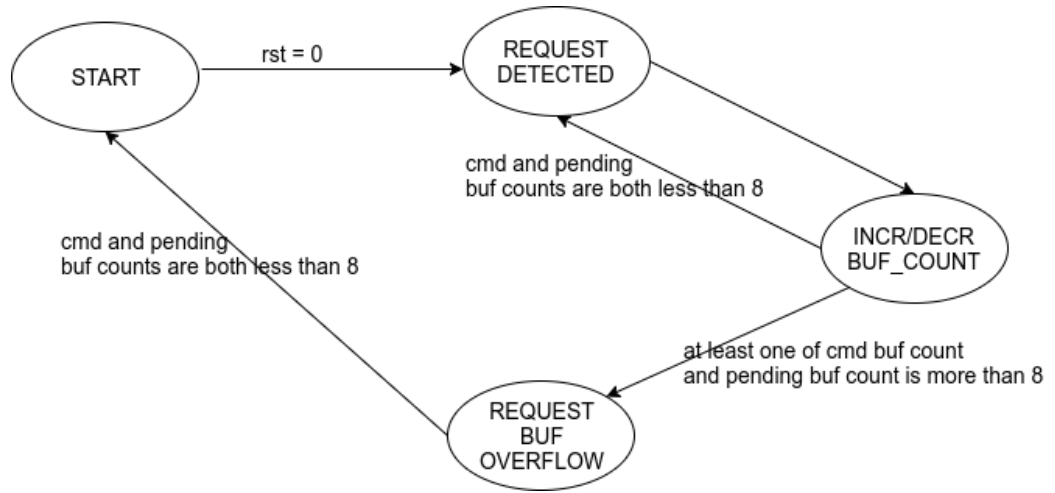


Figure 2.9: The Request-Receive FSM

- The **Request Detected** state is described here. If `request_req = 1` from the ahir side, the `request_data` will be put into both the `cmd` and `pending` buffers and the `request_ack` will be asserted. The `buf_counts` keep count of the number of request words put into the respective buffers. When connected to the select inputs of the respective 8x1 muxes, the `buf_counts` will put the earliest `request_word` stored in the buffer at the mux output. On the other hand, the `buf_count_decr_flags` are asserted by the request-send module each time a new request has been successfully sent to the MIG side.
- It will then go to the **Incr/Decr Buf Count** state. If (`cmd_buf_count_decr_flag = 0`) then increment the `cmd_buf_count ++` and if (`pending_buf_count_decr_flag = 0`), `pending_buf_count ++`. Similarly, if (`cmd_buf_count_decr_flag = 1`) decrement the same.
- As the storage buffers have been kept at a depth of 8, if either of the buffers (`cmd` or `pending`) overflow, it will wait for the pending requests to be serviced, (wait for the count to be less than 8, counts being from 0 to 7 that is).

The request-send module is described as shown :

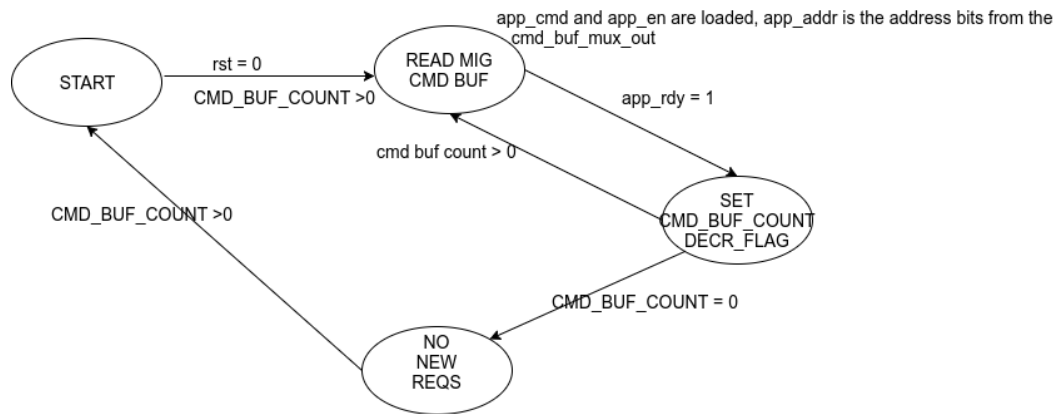


Figure 2.10: The Request-Send FSM

When app\_rdy is asserted, it means that the current request (pointed to by the cmd\_buf\_count) is being serviced. So the cmd\_buf\_count\_decr\_flag will be set in the next state.

### 2.4.3 Design of the Response Module

The response-module interface is shown below:

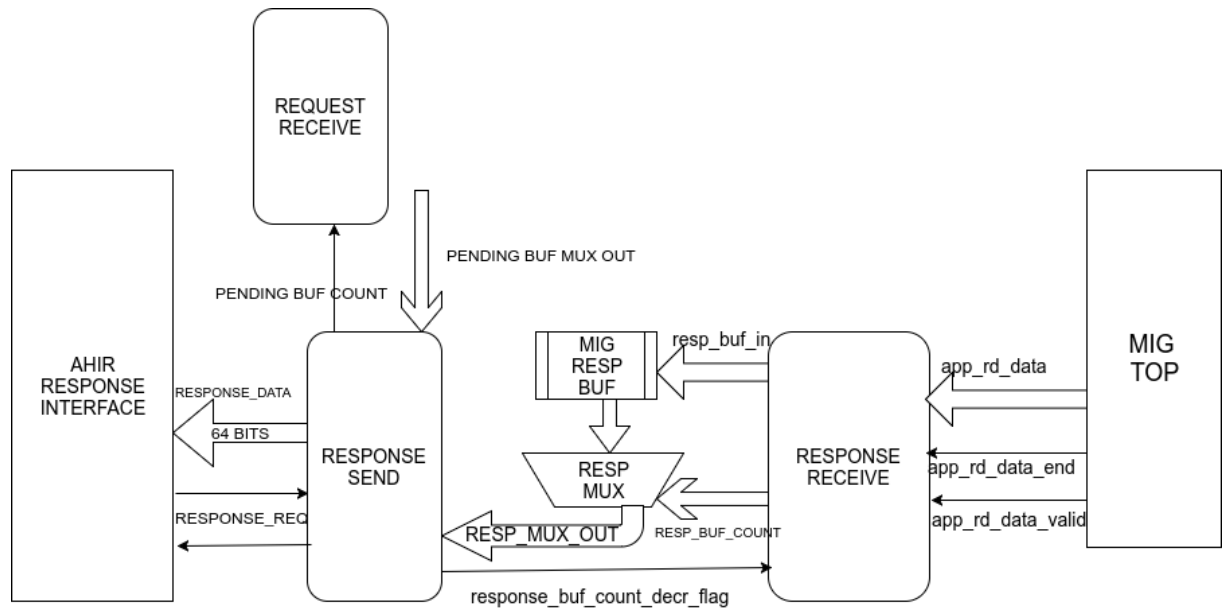


Figure 2.11: The Response Receive and Response Send Modules

The **response receive** module operates as follows :

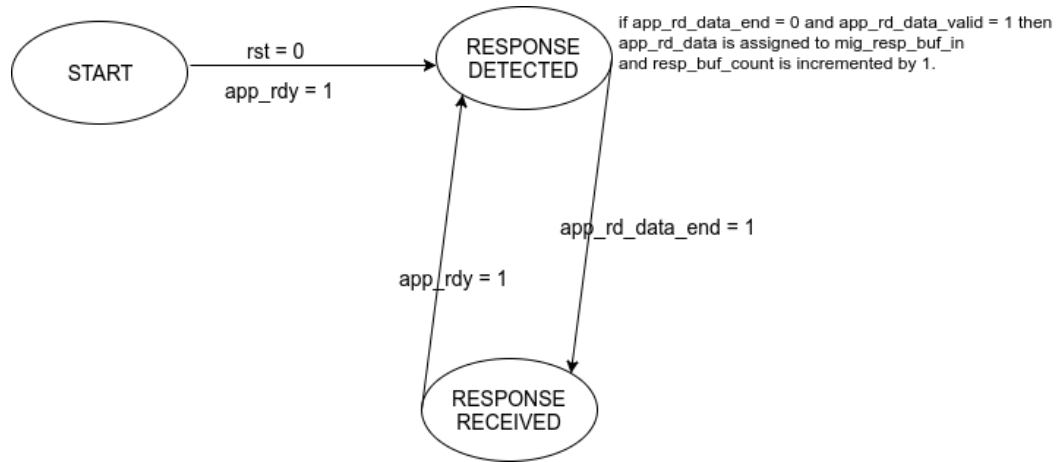


Figure 2.12: The Response receive Module

The **response send** module operates as follows :

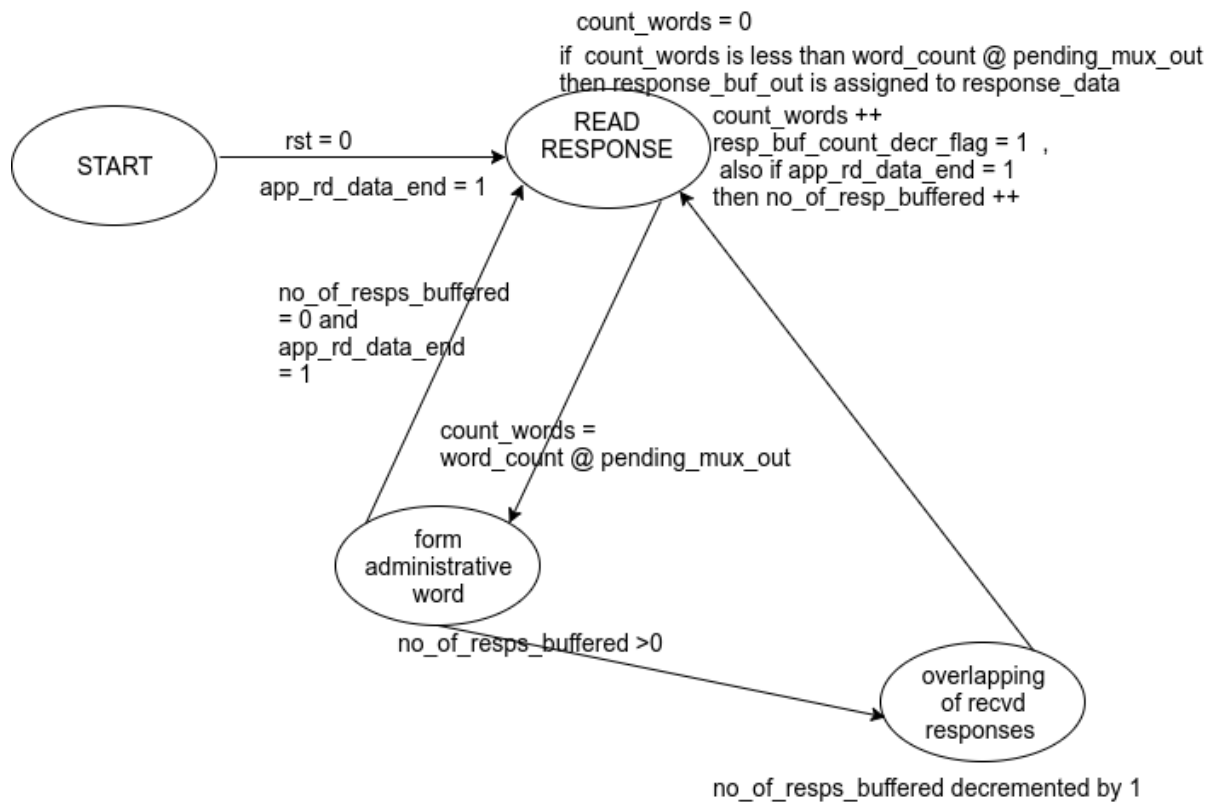


Figure 2.13: The Response Send Module

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)

no\_of\_resp\_buffered is also a local flag which keeps track of the other responses which are fully received (in resp\_buf) which the current req is being sent

Note here that the no\_of\_resp\_buffered flag is initilized to 0 at the start state itself while the count\_words is initilized to 0 for each read\_response state iteration. *In the above proposed design, the bridge can only serve requests **in-order**, that is, in order of their arrival from the ahir side.*