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

April 20, 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

List of Figures

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.

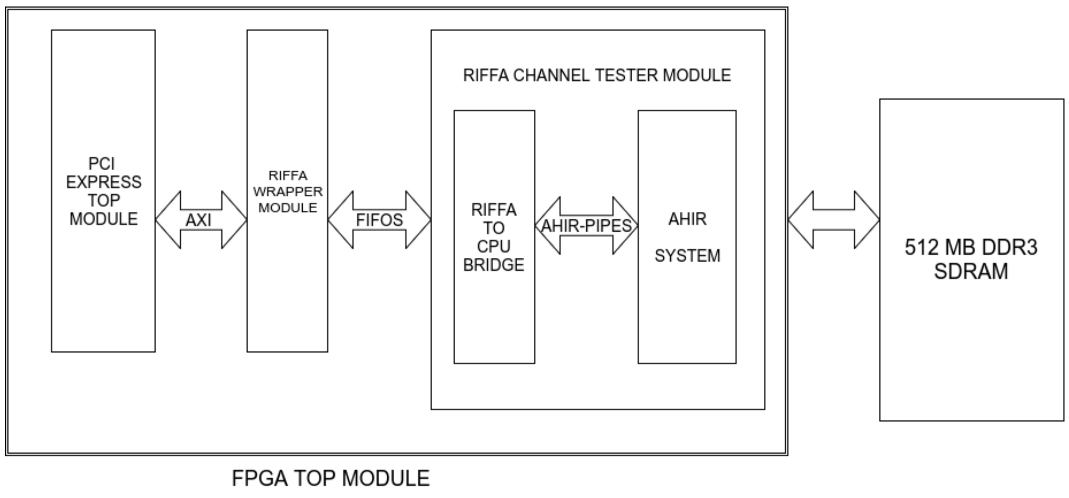


Figure 1.1: The Current Status of the Host Environment

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

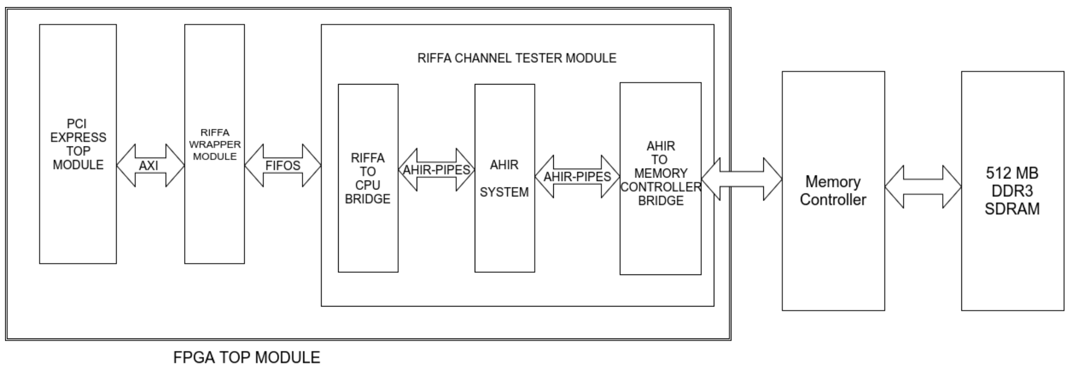


Figure 1.2: Target Status of the Host Environment

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.

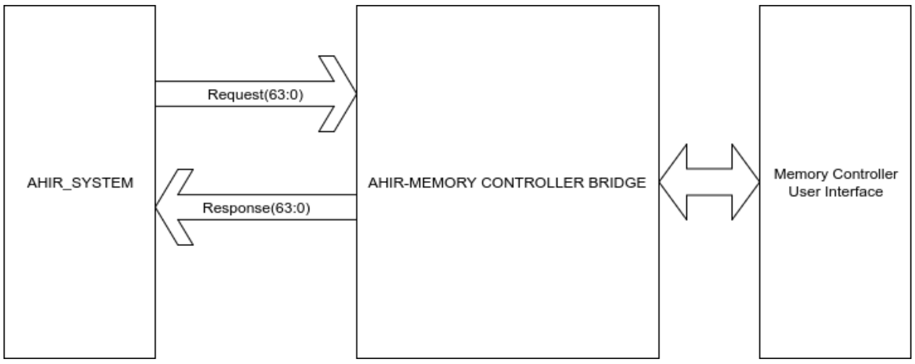

AHIR_SYSTEM
Request(63:0)
Response(63:0)
AHIR-MEMORY CONTROLLER BRIDGE
Memory Controller User Interface

Figure 2.1: AHIR-MC Bridge

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

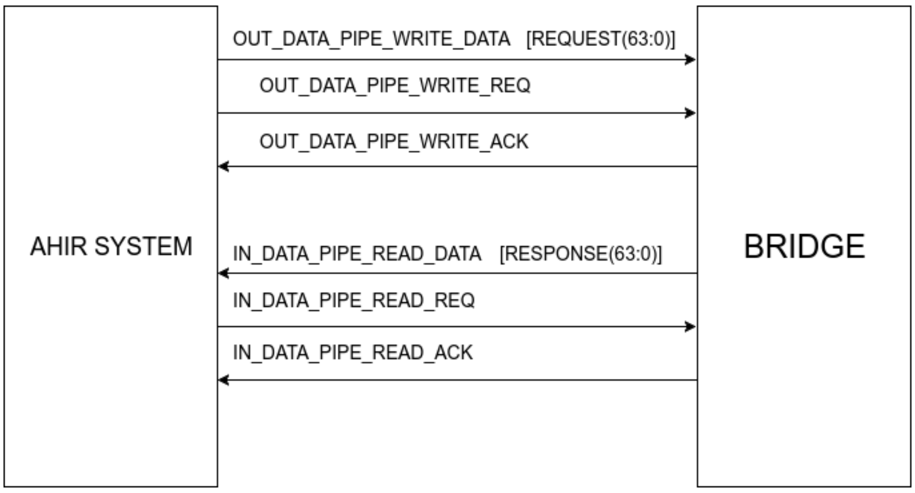


Figure 2.2: AHIR-MC Bridge

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

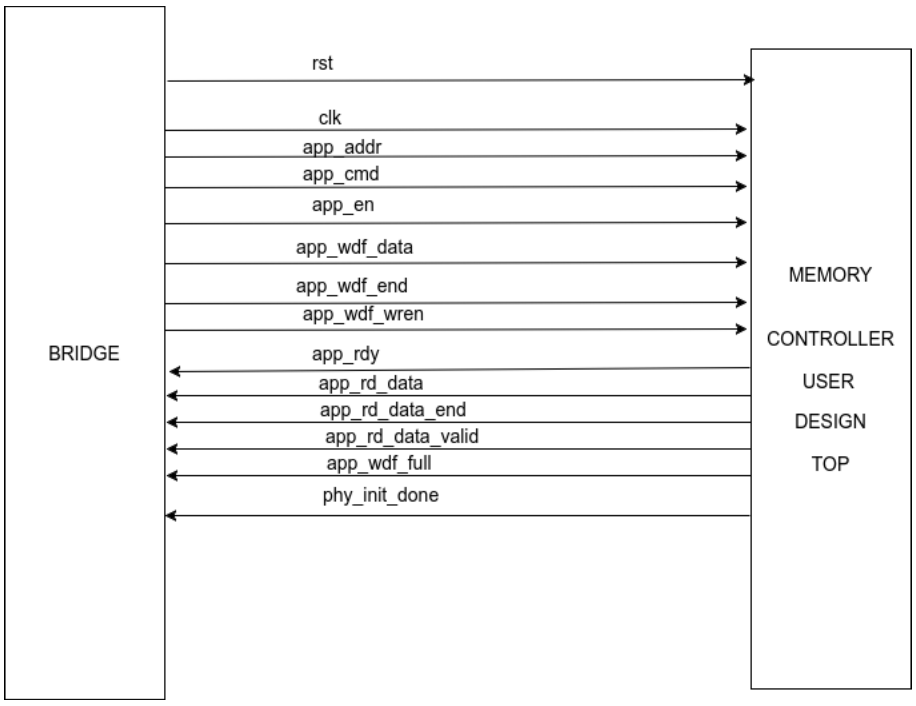


Figure 2.3: AHIR-MC Bridge

2.2 Description of the Request Packet

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

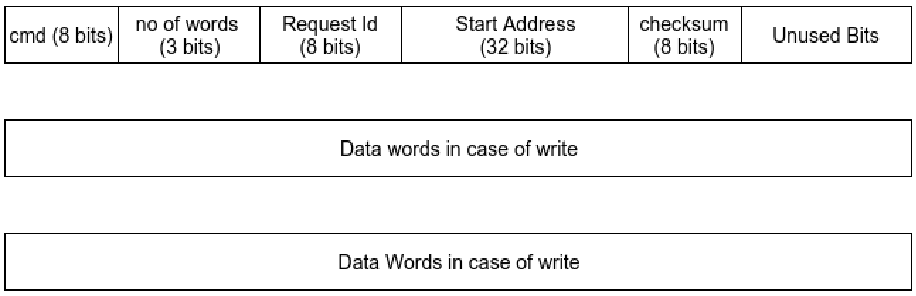


Figure 2.4: A Request Packet

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

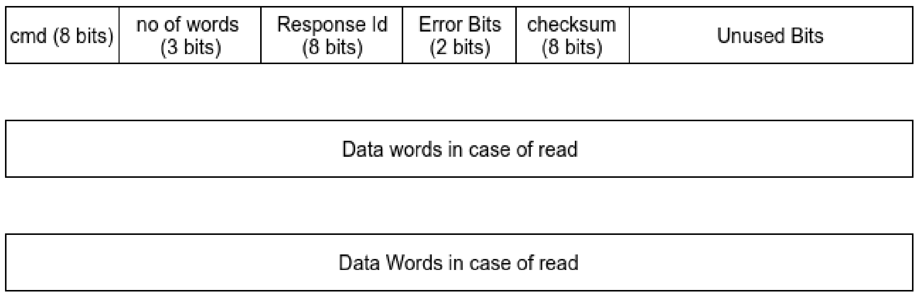


Figure 2.5: A Response Packet

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.

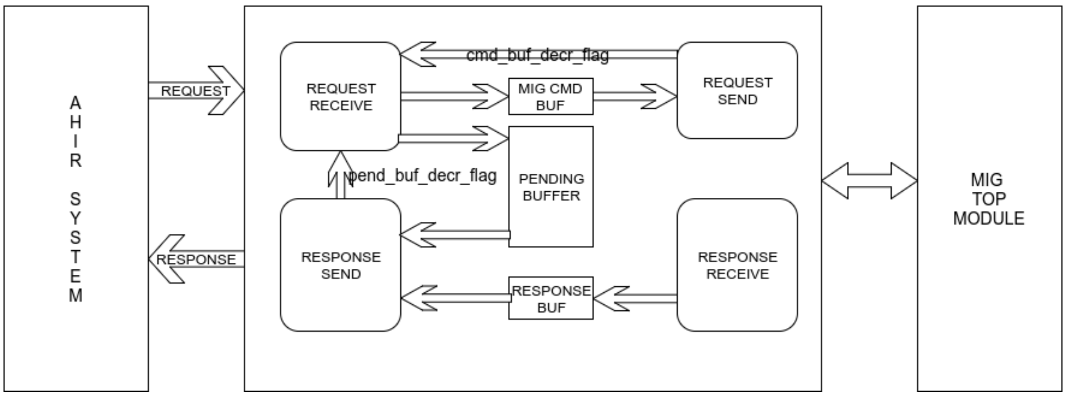


Figure 2.6: Bridge Overview

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 n staged, i.e., the depth of the buffer will be decided later on. All these three buffers are FIFO structures.

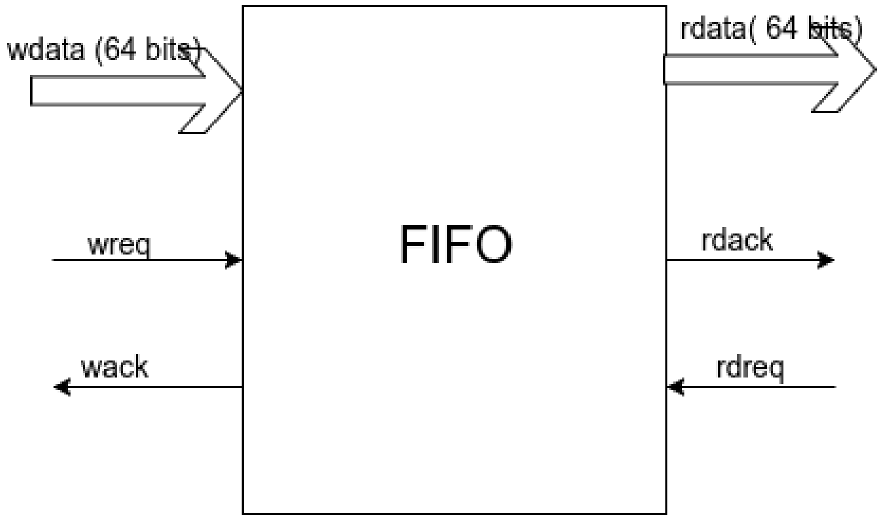


Figure 2.7: FIFO Buffers

2.4.2 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.

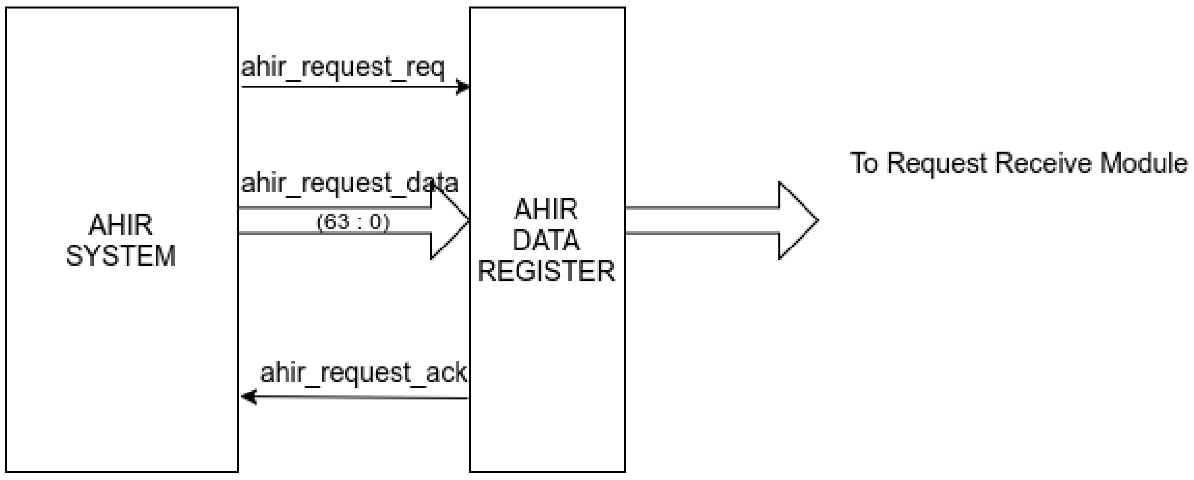


Figure 2.8: Ahir Data Register

The following fsm represents this basic idea.

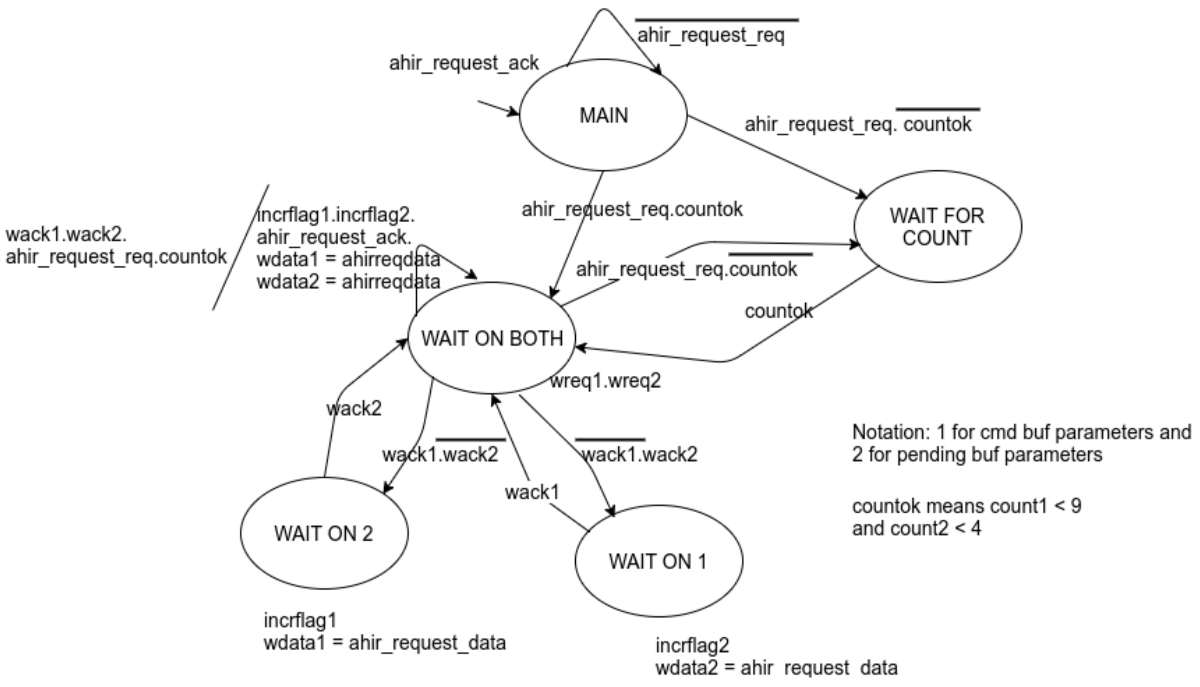

ahir_request_ack
$\overline{\text{ahir\_request\_req}}$
MAIN
ahir_request_req. $\overline{\text{countok}}$
WAIT FOR COUNT
ahir_request_req.countok
wack1.wack2.
ahir_request_req.countok
incrflag1.incrflag2.
ahir_request_ack.
wdata1 = ahirreqdata
wdata2 = ahirreqdata
ahir_request_req.$\overline{\text{countok}}$
countok
WAIT ON BOTH
wreq1.wreq2
wack2
wack1.$\overline{\text{wack2}}$
$\overline{\text{wack1}}$.wack2
wack1
WAIT ON 2
WAIT ON 1
incrflag1
wdata1 = ahir_request_data
incrflag2
wdata2 = ahir request data
Notation: 1 for cmd buf parameters and
2 for pending buf parameters
countok means count1 < 9
and count2 < 4

Figure 2.9: The Request Receive Module

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.

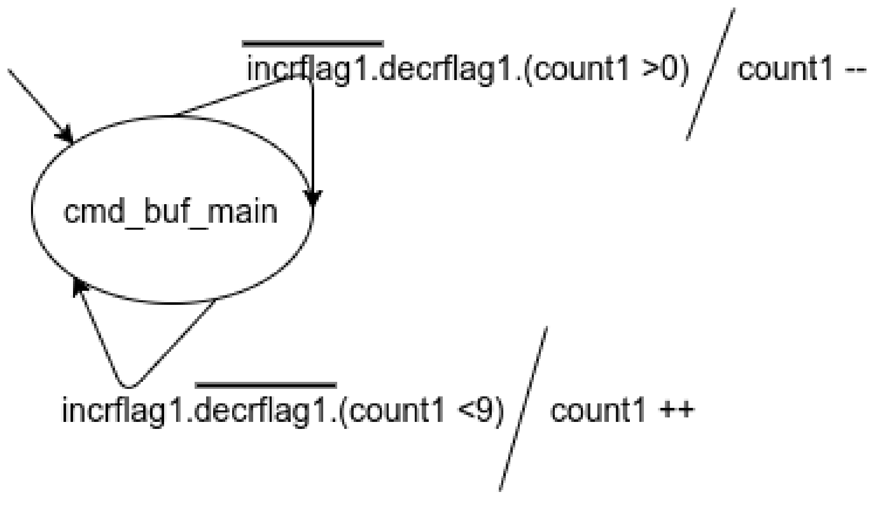
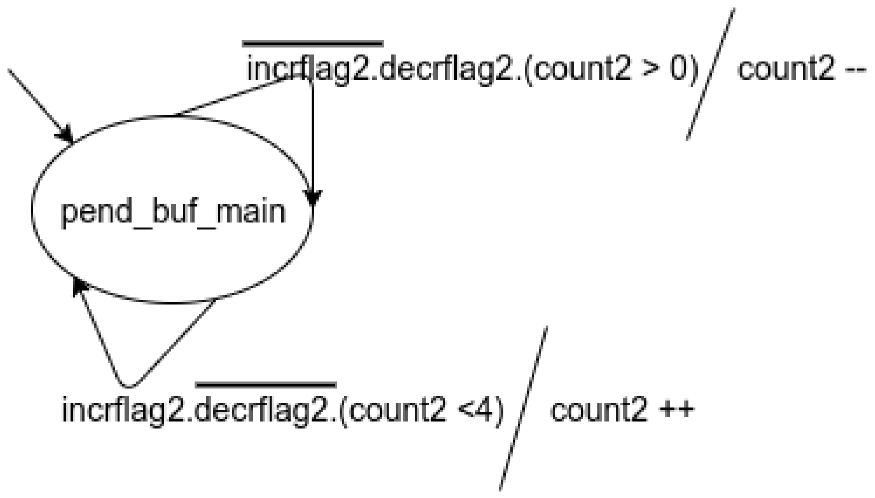

$\overline{\text{incrflag1}}$.decrflag1.(count1 >0) / count1 --
cmd_buf_main
incrflag1.$\overline{\text{decrflag1}}$.(count1 <9) / count1 ++

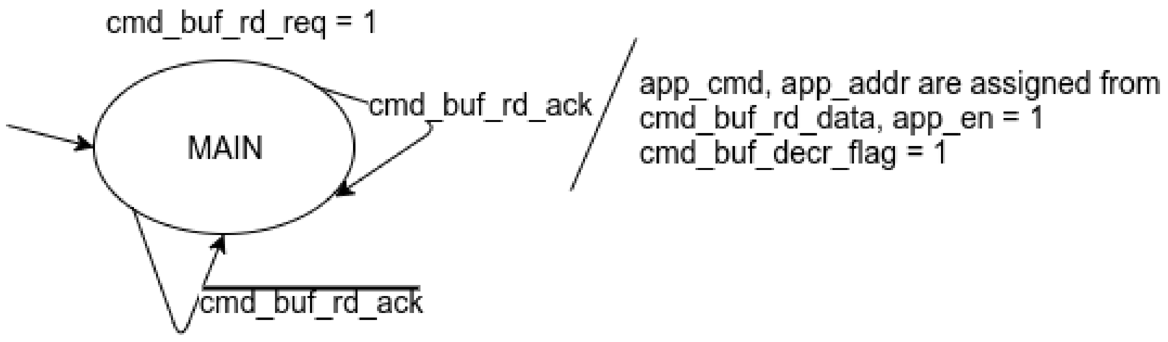
Figure 2.10: The Cmd Buffer Count Module



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

Figure 2.11: The Pending Buffer Count Module

The request-send module operates as follows:



cmd_buf_rd_req = 1
MAIN
cmd_buf_rd_ack / app_cmd, app_addr are assigned from cmd_buf_rd_data, app_en = 1 cmd_buf_decr_flag = 1
$\overline{\text{cmd\_buf\_rd\_ack}}$

Figure 2.12: The Request-Send Module

2.4.3 Design of the Response Module

The response-module interface is shown below:
The **response receive** module operates as follows :

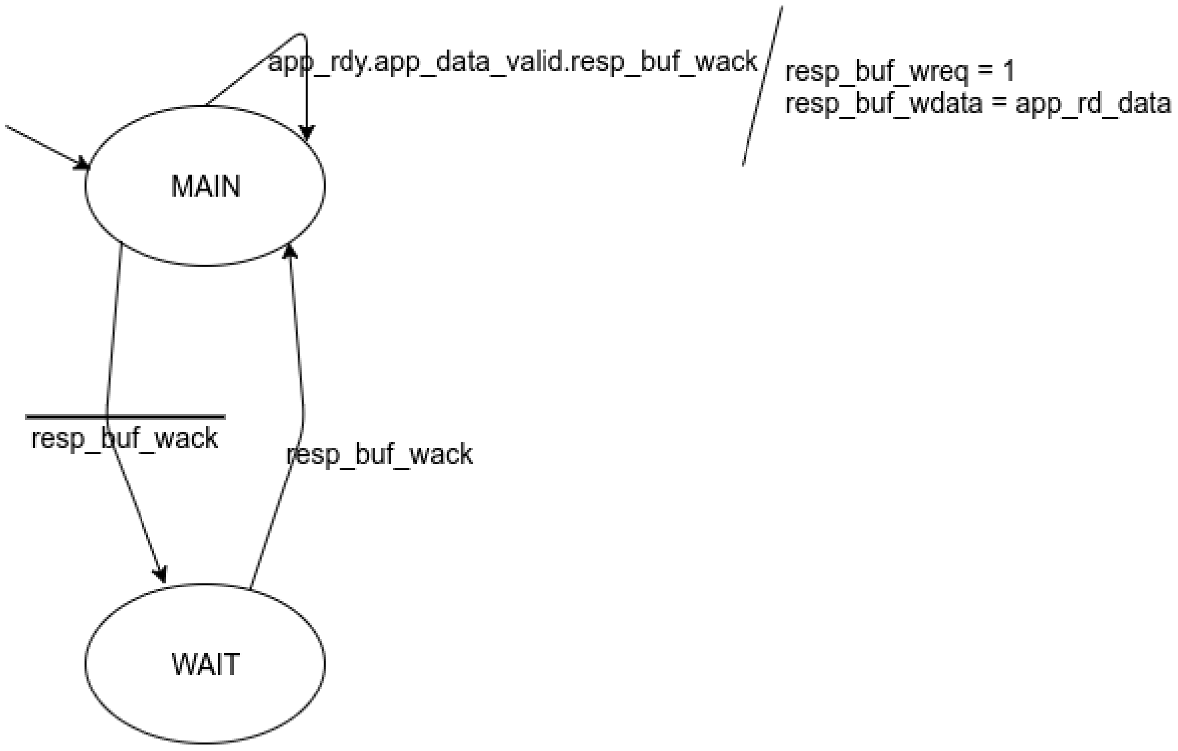

app_rdy.app_data_valid.resp_buf_wack
resp_buf_wreq = 1
resp_buf_wdata = app_rd_data
MAIN
$\overline{\text{resp\_buf\_wack}}$
resp_buf_wack
WAIT

Figure 2.13: The Response receive Module

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

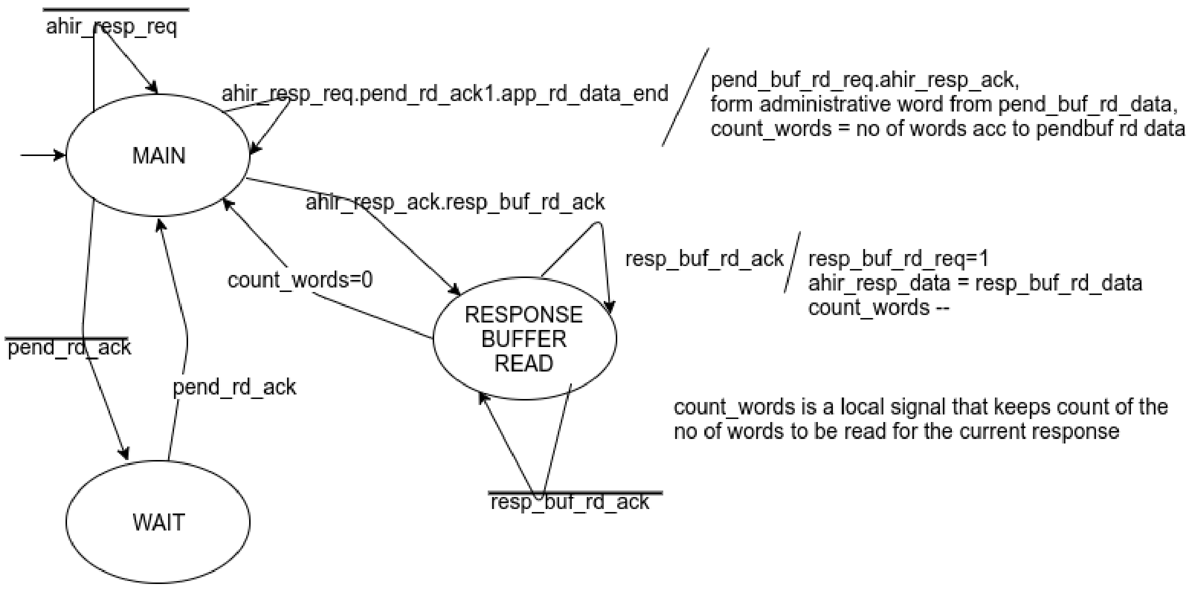

$\overline{\text{ahir\_resp\_req}}$
ahir_resp_req.pend_rd_ack1.app_rd_data_end
pend_buf_rd_req.ahir_resp_ack,
form administrative word from pend_buf_rd_data,
count_words = no of words acc to pendbuf rd data
MAIN
ahir_resp_ack.resp_buf_rd_ack
count_words=0
resp_buf_rd_ack
resp_buf_rd_req=1
ahir_resp_data = resp_buf_rd_data
count_words --
RESPONSE BUFFER READ
$\overline{\text{pend\_rd\_ack}}$
pend_rd_ack
count_words is a local signal that keeps count of the no of words to be read for the current response
$\overline{\text{resp\_buf\_rd\_ack}}$
WAIT

Figure 2.14: The Response Send Module

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).