Problem statement: AXI Signal Manipulation

This problem should be implemented in **Verilog**.

In this problem, you will implement a simple IP that is able to change the address fields of AXI signals in real time**.** Only the Read Address Channel and Write Address Channel have address fields. Other than the address fields, other signals must propagate as is to obey AXI protocol.

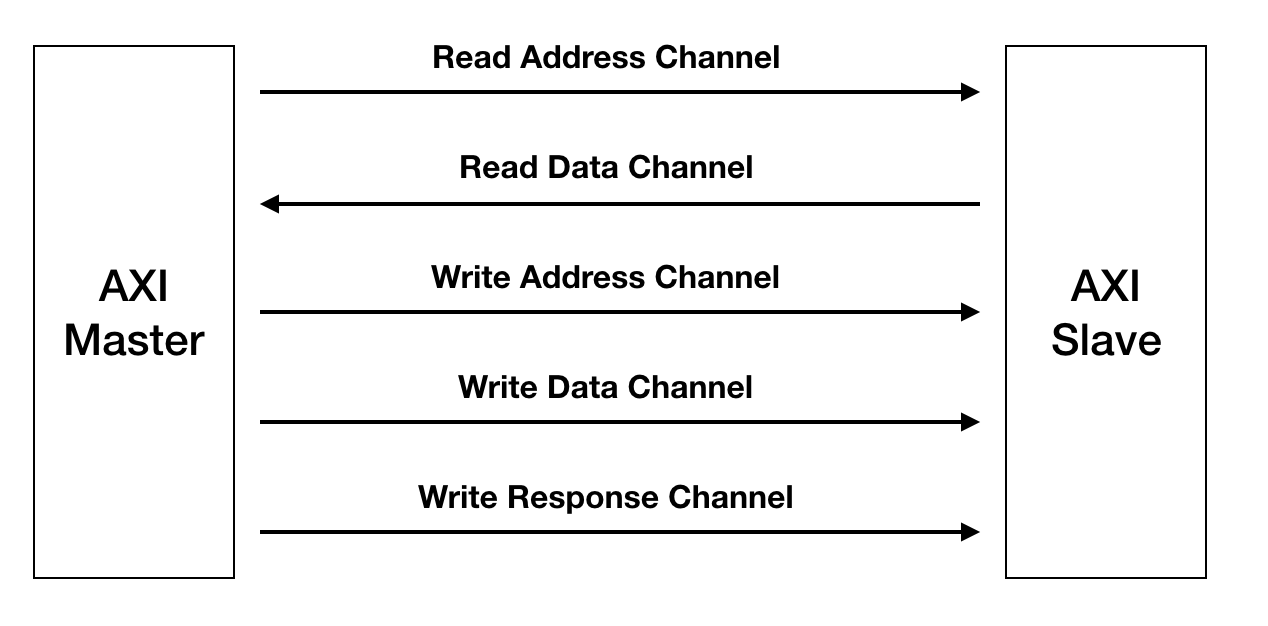
AXI is a high-performance bus that has multiple channels, namely *Read Address* Channel, *Read Data* Channel, *Write Address* Channel, *Write Data* Channel, and *Write Response* Channel. Each channel has multiple signals to perform handshake and others.  Figure 1 demonstrates the channels between an AXI master and an AXI slave. Do note, each channel has multiple signals, please check AXI spec for details [1].

Figure 1.

Your IP will be inserted in the middle between the AXI master and the slave. Figure 2 shows the block diagram. The IP is invisible to both the AXI master and the AXI slave. In its simplest form, this IP can just do a “pass-through”: your IP will just transfer whatever the AXI master transfer to your IP as is to the AXI slave. Here, we want to extend this IP to perform some basic Virtual Memory functionality, i.e., being able to change the address fields sent by AXI master. In Figure 2, we call the address fields sent by AXI master (ARADDR and WRADDR), as Virtual Address (**VA**), and we call the address fields that are sent by your IP and eventually seen by AXI slave, as Physical Address (**PA**). Do note: other than the address fields, all other signals must preserve their content and order, to obey the original AXI protocol. The difficulty here is that the translation between VA and PA may take more than one cycle. That means you need to buffer each VA’s associated signals, and output them along with the PA.

You don’t need to implement a sophisticated translation mechanism, in this task, just perform this static translation: *PA = VA + 0x1000*

However, you are REQUIRED to manually add additional latency to the above calculation to emulate the effect of long translation delay. You can build your IP with some pre-defined latency (e.g., 1 cycle, 10 cycles, 100 cycles). Try to separate the signal buffering and translation part as two different modules. Because in reality, the translation can be a full-fledged MMU.

You only need to have a project that is: 1) synthesizable, 2) functionally correct.

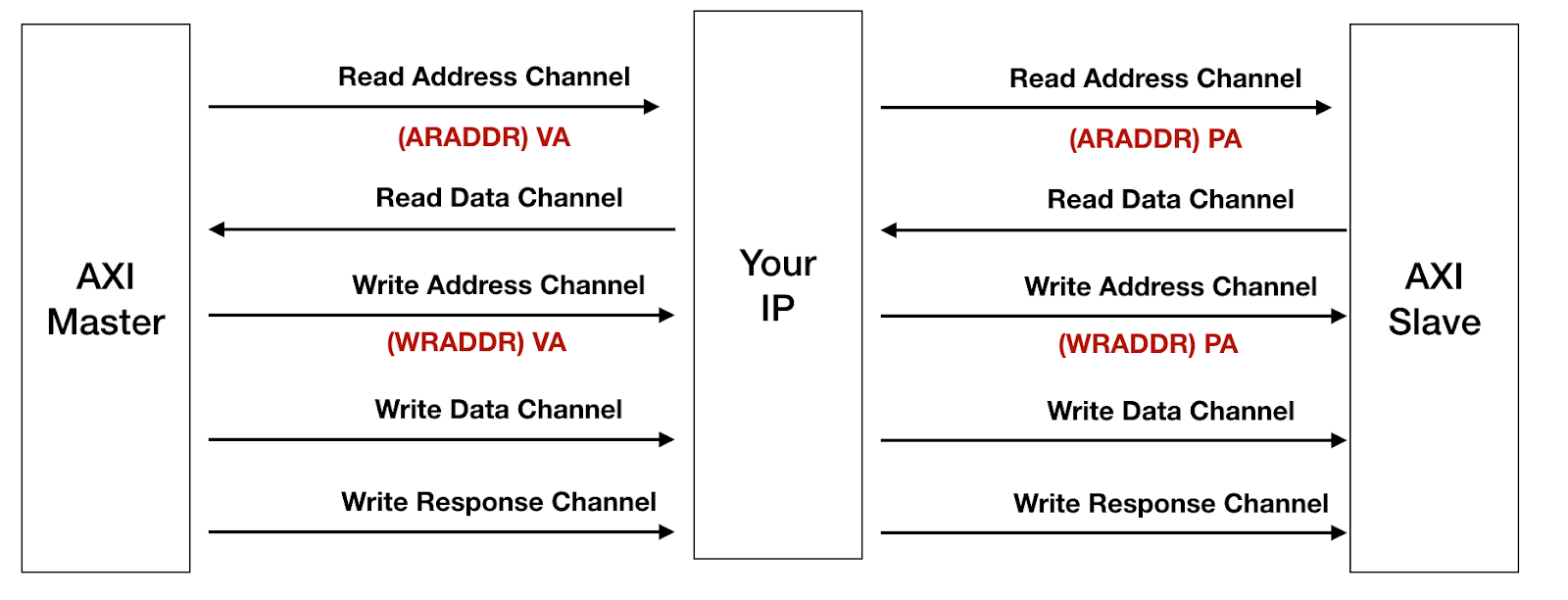


Figure 2.

References

[1] [AXI Specification](http://www.gstitt.ece.ufl.edu/courses/fall15/eel4720_5721/labs/refs/AXI4_specification.pdf)

[2] [A full VHDL implementation](https://github.com/pulp-platform/axi_rab)

Solution:

The code from [2] is used as a base and a new module axi\_timed\_buffer\_rab was written that would store all the incoming data at the slave port of the IP into the buffer entries along with a timer for each buffer entry (as latency provider) which would count down on each rising edge and the entry which has been counted down would be sent to the output port of the axi\_va\_pa\_ip IP module and the corresponding valid would be asserted and held high till AXI slave asserts the ready signal for a clock.

The module can be broken into 5 parts:

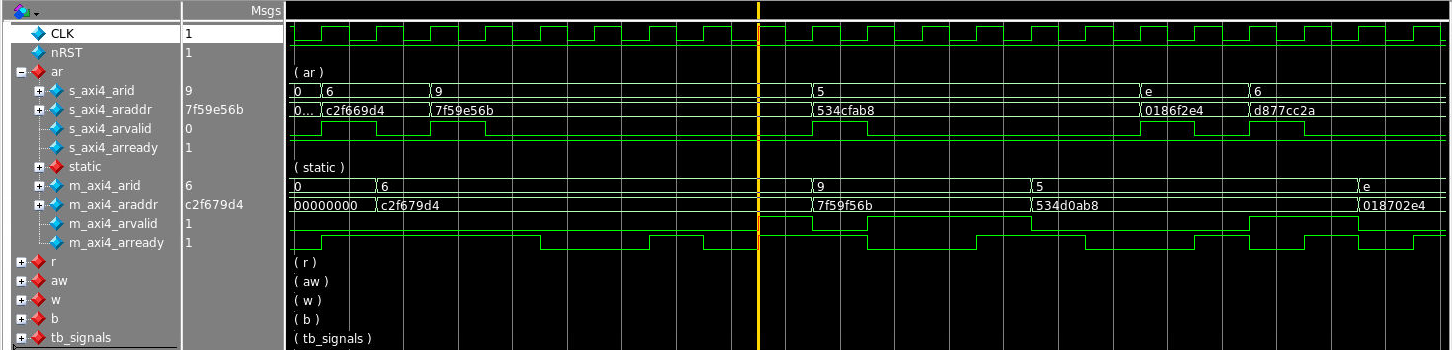
1. Read Address port which acts as a slave and receives the read address data from AXI Master and stores it into the buffer axi\_timed\_buffer\_rab of depth = 4 (parameterized). While storing the VA is converted to PA and the PA is stored as the address field. When the timer has been counted down for any of the entry it is sent as output from the master port of the IP and is accepted by the AXI slave whenever ready.
2. Write Address port which acts as a slave and receives the write address data from AXI Master and stores it into the buffer axi\_timed\_buffer\_rab of depth = 4 (parameterized). While storing the VA is converted to PA and the PA is stored as the address field. When the timer has been counted down for any of the entry it is sent as output from the master port of the IP and is accepted by the AXI slave whenever ready.
3. Write Data port which would receive the data from Master AXI for the address sent on Write address channel. The data is stored in buffer and put out to Master AXI on the next rising edge of the clock as this has no VA to PA conversion and hence wouldn’t take any time.
4. Read Data Port which acts as slave to receive the data from AXI slave. The data is stored in buffer and put out to master AXI on the next rising edge of the clock as this has no VA to PA conversion and hence wouldn’t take any time.
5. Write Response port which receives acknowledgment from the AXI Slave that a write has been performed successfully for a particular ID. The data is stored in buffer and put out to master AXI on the next rising edge of the clock as this has no VA to PA conversion and hence wouldn’t take any time.

A comprehensive testbench axi\_va\_pa\_ip\_tb has been written to test the above mentioned ports. To maintain simplicity and just test the functionality most sideband informational signals from each port have been kept at static value 0 and only functional signals like address, data, id etc have been modified. All the ports are being tested to see if the correct value which was received by the IP at its slave port is being sent out of the master ports after the required delay values (decided by the latency set by the block, in our case we are setting it to 7 clocks in our TB) with exception of Address ports which would spit out a PA of VA + ‘h1000. The order of the data being sent is also monitored to check if the buffer is acting correctly.

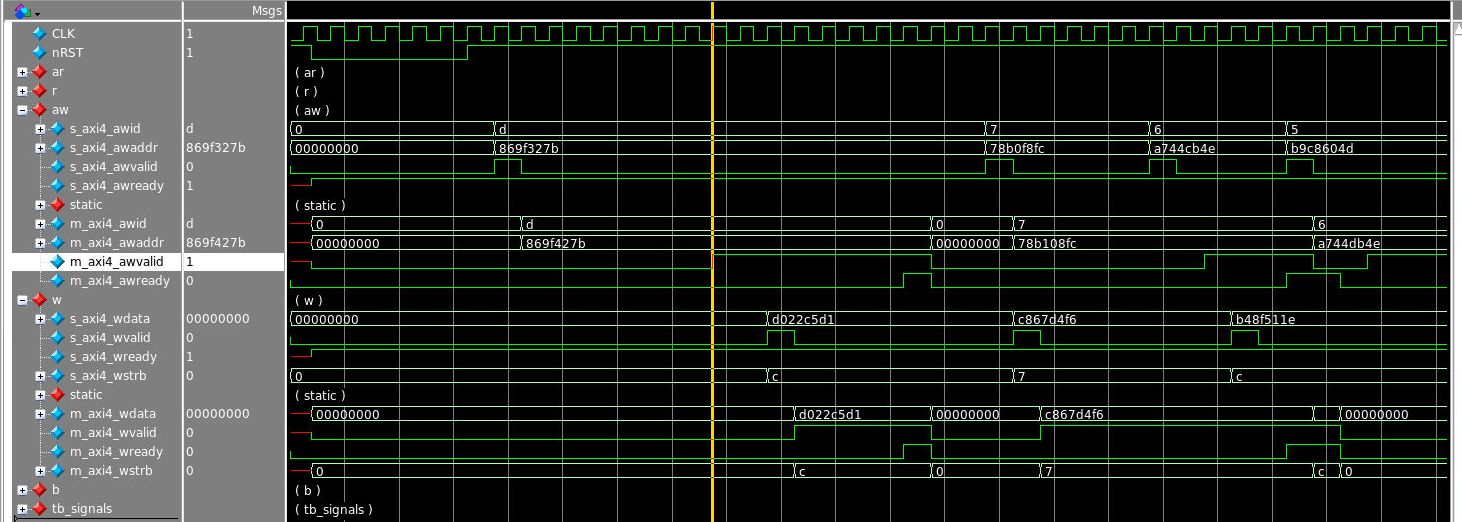
The simulation was run using Modelsim and waveforms viewed in Questasim. The file axi\_va\_pa\_ip\_tb.wlf can be used to view the waveforms. Altera Quartus 2 was used for synthesis and the information can be found in the file axi\_va\_pa\_ip.log.

Pasting some screenshots of the waveforms:

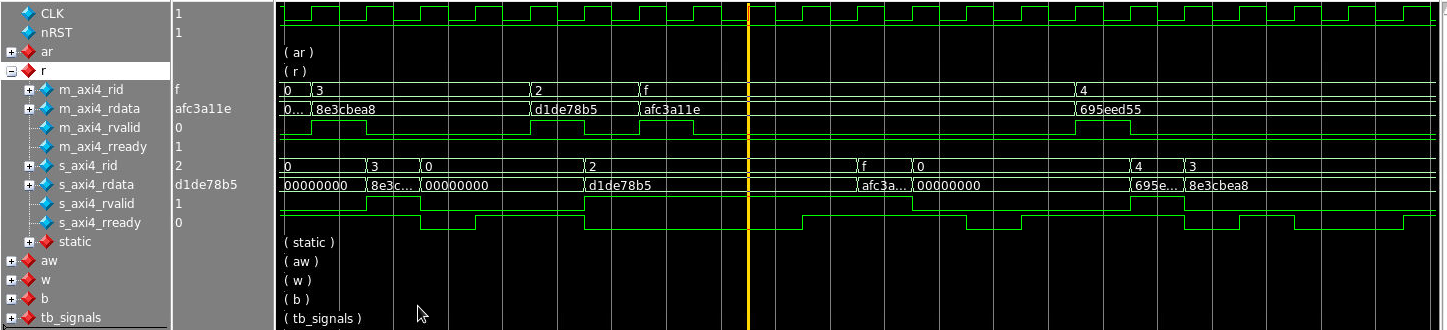
1. Read Address:



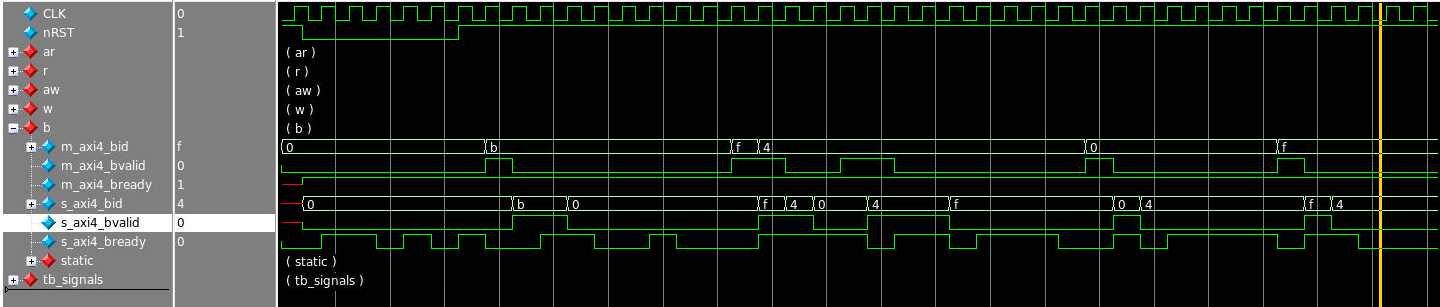
1. Write Address and Data:



1. Read Data:



1. Write Response:



Please contact for more information. Thanks!