9/24/2022

AXI Interconnect

Design and Simulation

Table of Contents

Table of Figures

Abstract

In this proposal, the architecture discussed is the implementation of AXI interconnect design using Verilog HDL to write the RTL code and ModelSim for functional verification of the system. AXI interconnect function is to build a communication “Read and Write Transactions” between an AXI master device and AXI slave one, and supports connecting Multi-Masters and Multi-Slaves combined with many features. The target is to reach an Optimized and Synthesizable Verilog code achieving the functionality of the AXI interconnect. Considering the ModelSim Simulation tool, a test-bench is built depending on a test plan with different test cases including some corner cases to validate the system operation.

Keywords: AXI Interconnect, Verilog HDL, ModelSim, Test-bench, Multi-Masters, Multi-Slaves, Read and Write Transactions.

Introduction

AMBA AXI protocol support high performance and high frequency system designs, it is an on-chip communication bus protocol developed by ARM as it meets the interface requirements of a wide range of components. It deals with two types of transactions “Read and Write”, using main 5 channels: Write Address Channel, Write Data Channel, Write Response Channel, Read Address Channel and Read Data Channel. This separation of the read and write transactions channels make the system performance better and increase the throughput of transactions. All these transactions are initiated by a master and the slave responding to the operation.

Document Description

The main points discussed in this document:

- Section 3 “AXI Protocol”: What is the AXI protocol?
  What are the features supported?

- Section 4 “Hardware Architecture”: Explaining the hardware architecture used in the AXI interconnect design “RTL Code”. Discussing how the operation of each channel will done to support the features required. Also, discuss the architecture of the slave required.

- Section 5 ”System Validation and Waveforms”: Discuss all the test cases built in the test plan supported with waveforms explain each test case from the ModelSim.

- Section 6 “Synthesis”: Show the output of the synthesis tool and discuss the results in area, power and timing reports.

- Section 7 “Glossary”: including the description of any expression or abbreviation used in the document.

AXI Protocol

The AXI protocol is burst-based and defines the following independent transaction channels: read address, read data, write address, write data, write response as shown in fig. (1).

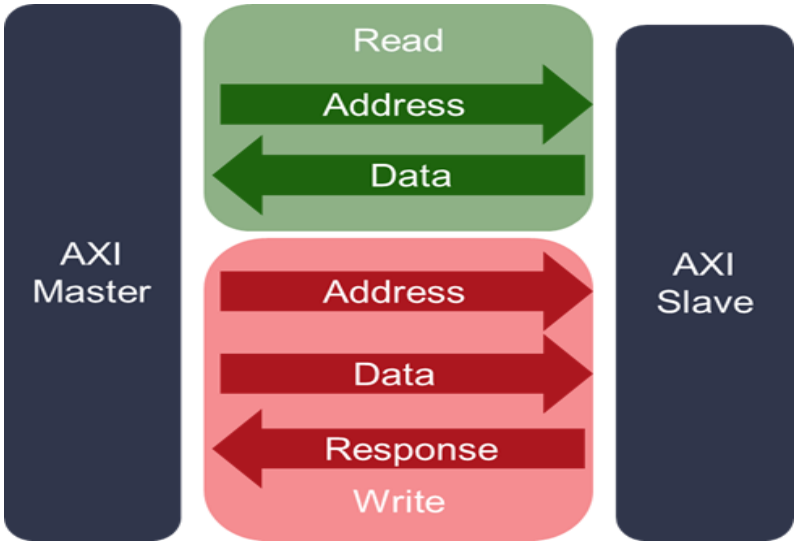

Figure 1: The main 5 channels in AXI protocol

In fig. (), notice that the source and destination of the signals in the channels are varying may be once the slave and once the master but this doesn't prevent that the master is responsible for initiating all transactions.

Channels Description:

As shown in fig. (2) and fig. (3) That AXI has two types of transactions and each transaction has its own channels that used in the operation.

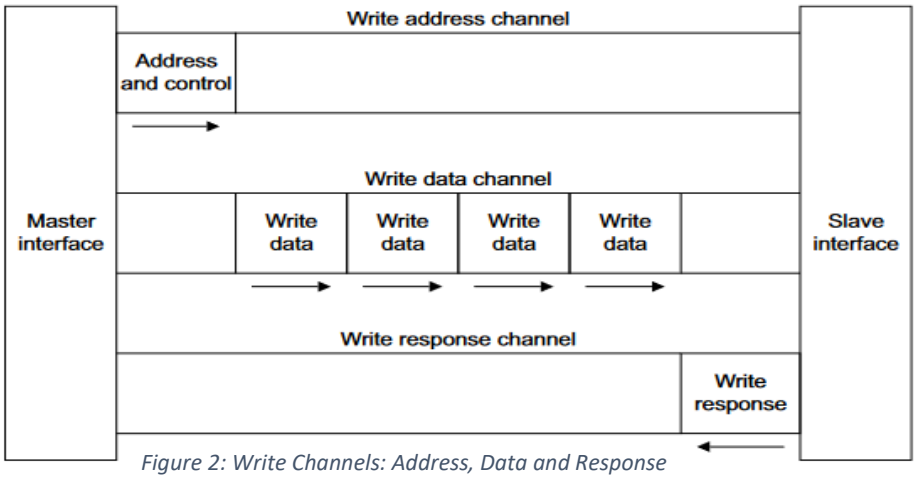

Figure 2: Write Channels: Address, Data and Response

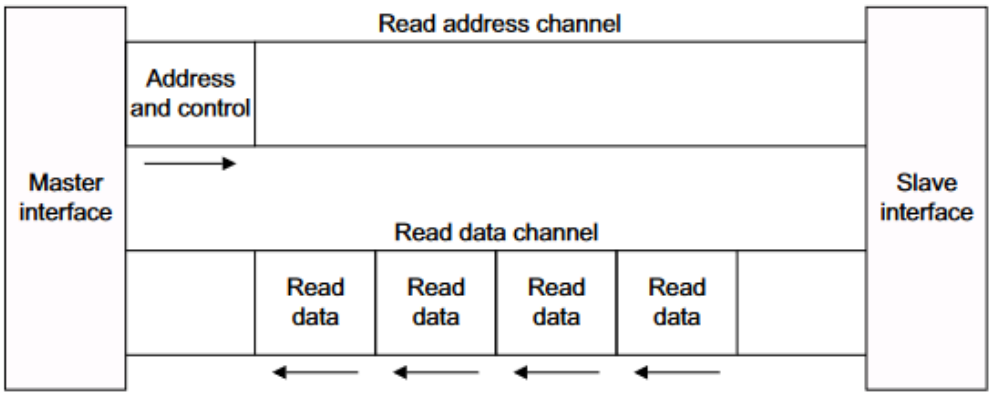

Figure 3: Read Channels: Address and Data

Address channels:

Each address channel carries the address for each transaction and the control information that describes the nature of the data to be transferred as the data is transferred between master and slave in two ways:

1. A write data channel to transfer data from the master to the slave. In a write transaction, the slave uses the write response channel to signal the completion of the transfer to the master.

2. A read data channel to transfer data from the slave to the master.

The Control Signals in Address Channels:

- Burst transactions length: determines the number of data transfers associated with the address either in Read or Write Transactions.
- Burst Size: indicates the size of each transfer in the burst.
- Burst Type: determines the type of burst transaction either Fixed, Incrementing, or Wrap “Explained in Glossary”.
- Valid and Ready Signals: Responsible for Handshaking “Discussed later”.

There are other control signals in the address channels such as “Lock”, “Cashe” and “Protection” but the above signals are the most important.

Write Data Channel:

The write data channel carries the write data from the master to the slave and it has also some control signals such as:

- Write Strobes: indicates which byte lanes hold valid data. There is one write strobe bit for each eight bits of the write data bus.
- Write Last: indicates the last transfer in a write burst transactions.
- Ready and Valid Signals: Responsible for Handshaking “Discussed later”.

Write Response Channel:

The slave uses the write response channel to respond to write transactions. All write transactions require completion signaling on the write response channel.

The control signals used beside the response itself is the Valid and Ready Signals for the Handshaking.

Read Data Channel:

The read data channel carries both the read data and the read response information from the slave to the master. Beside these signals there are some control signals used in this channel such as:

- Read last: indicates the last transfer in a read burst transactions.
- Ready and Valid Signals: Responsible for Handshaking “Discussed later”.

The channel description shows the signals and their usage during each transaction and the role of each channel in order to achieve the best performance and throughput in the transactions but how the master connected directly to the slave is it forever or there are some conditions to be satisfied to make the information that found in each channel go from master to slave and vice versa, so the next part the **Handshaking process** is explained.

Handshaking Process:

All channels us the Valid and Ready Handshake process to transfer the address, data, and control signals. As discussed before in each channel there is a source of information and there is a destination, such as in Write Address Channels the source of the address and control signals is the Master and the destination is the slave. So how the source and destination affects the Handshake process:

- Source: generates the Valid signal to indicate when the address, data or control information is available.
- Destination: generates the Ready signal to indicate that it can accept the information.

The transfer happens only when the Valid and Ready signals are high. This will be clear in the next examples.

Handshake Examples

Example 1:

The source presents the information after T1 and asserts the Valid signal. The destination asserts the Ready signal after T2, and the source must keep its information stable “Valid == 1” until the transfer occurs at T3 as shown in fig. (4).

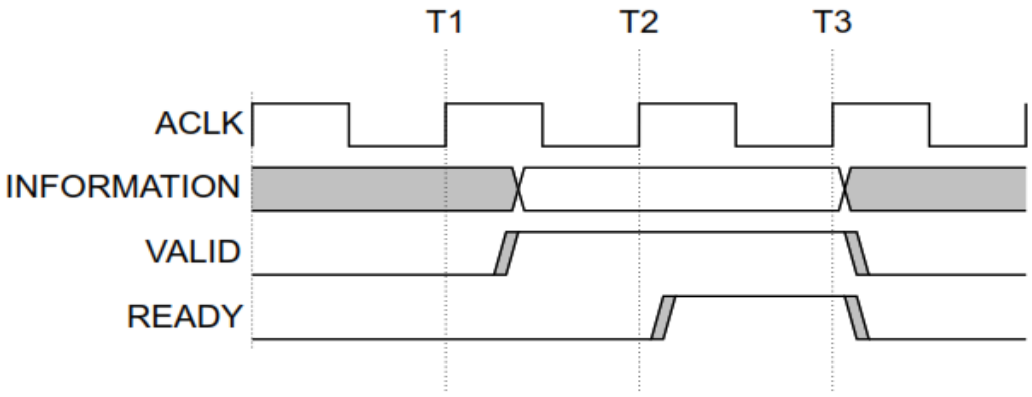

Figure 4: Handshake Example 1

Example 2:

The destination asserts Ready, after T1, before the information is valid, indicating that it can accept the information. The source presents the information, and asserts Valid, after T2, and the transfer occurs at T3 as shown in fig. (5).

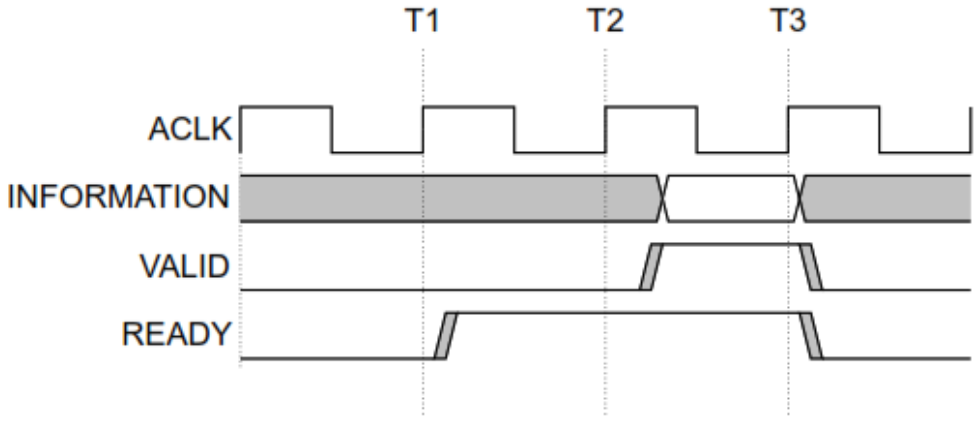

Figure 5: Handshake Example 2

Example 3:

Both the source and destination happen to indicate, after T1, that they can transfer the information. In this case the transfer occurs at the rising clock edge when the assertion of both Valid and Ready can be recognized. This means the transfer occurs at T2 as shown in fig. (6).

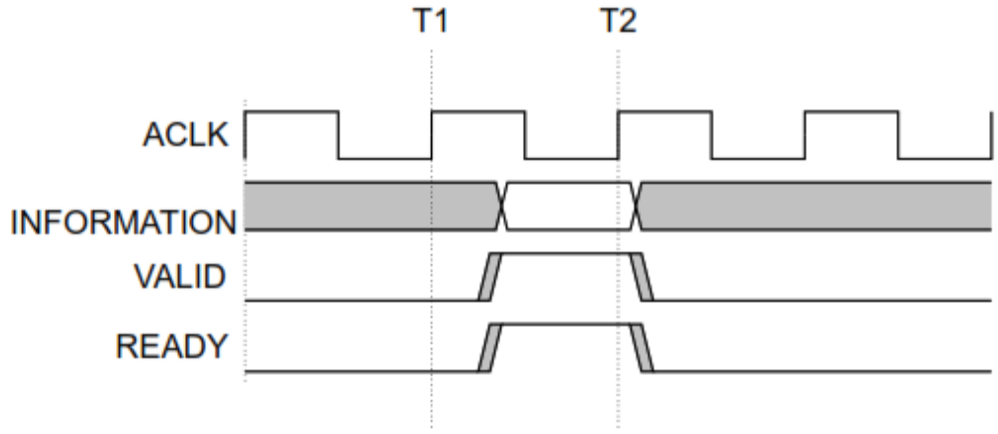

Figure 6: Handshake Example 3

As a conclusion the **Handshake process** is the major part in building the communication between the master and slave and vice versa.

Hardware Architecture

Top Module

Based on the given architecture in fig. (7), it is required to design the AXI interconnect module and a slave to connect it to IIC module based on its design requirement proposed by IIC team.

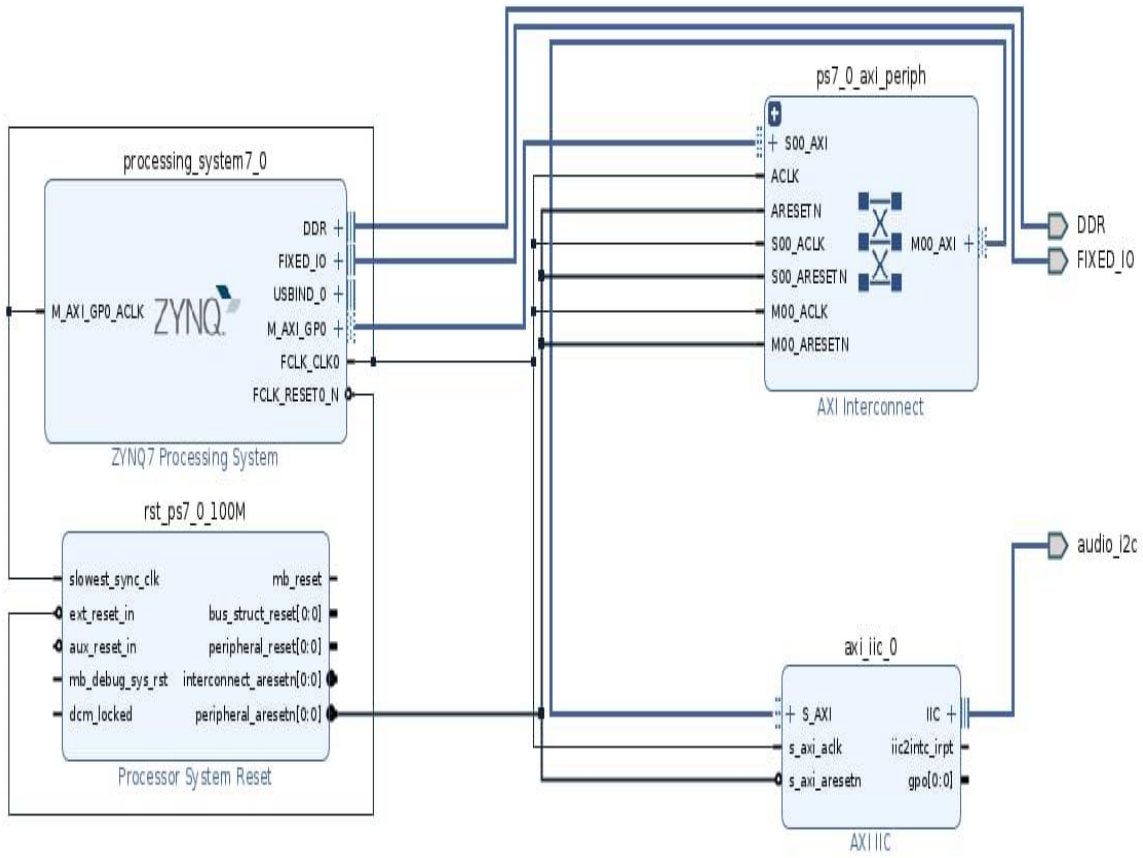

Figure 7: The design Top Module and the AXI Interconnect

AXI Interconnect architecture

Starting with the architecture of the AXI interconnect, the design of each channel will discussed separately as following:

Write Address Channel Architecture:

Note fig. (), explaining how the write address channel signal go from a master to a required slave.

The operation go as following:

- When a master want to communicate on the address channel it puts its information and assert the Valid signal.
- The **Arbiter** function is to handle which master will communicate now, so it depend on the Valid signal if only one master has asserted its Valid signal, it will be chosen to build the communication. However, if both have asserted their Valid signals, the arbiter will choose Master 1 as higher priority.
- Then, the **Decoder** role reached, one master has a specific address want to communicate but with which slave, so the decoder based on the base address found in the address information "Upper bits", it decides which slave to communicate with and has two other important outputs: Master-ID, Q-Enable that will be used in the Write Data Channel.
- Last, the **Handshake process** check on the ready signal from that slave to be asserted and make the communication built and done between a specific master and a specific slave.

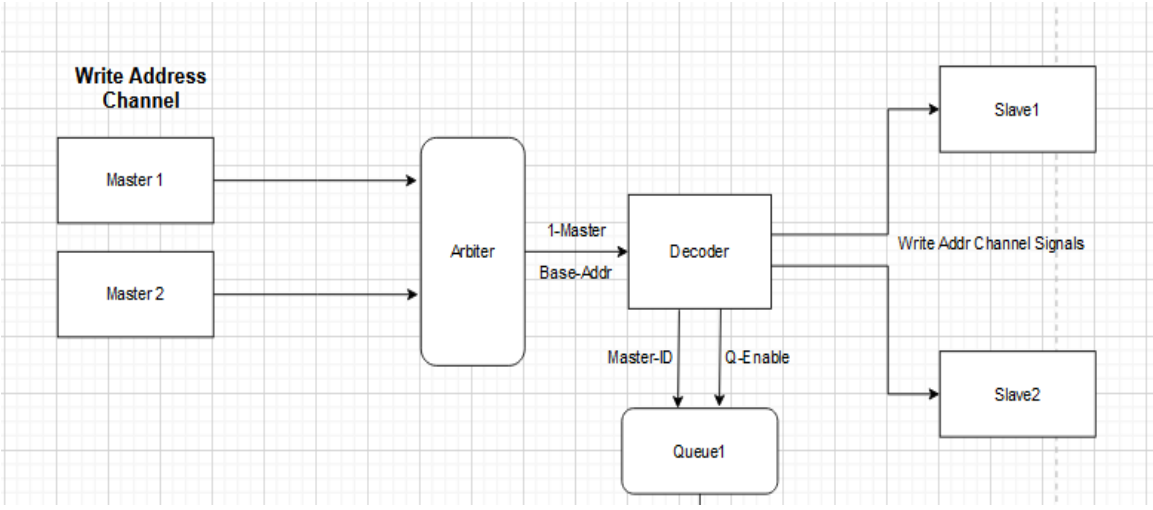

Figure 8: Write Address Channel Architecture

Write Data Channel Architecture:

In fig. (), the architecture of Write Data Channel explains how to transfer the data from a master to a slave based on the address given in the previous channel.

The operation goes as following:

- A **queue** is implemented for each slave the data wrote to this queue is the master ID that want to write in the relative slave come from the address channel.
- When reading the **queue** by using the Master ID, a specific master must be decoded "Using the **Multiplexer**" to build the communication on the Write Data Channel with this slave.
- Using a **separate queue** for each slave gives more flexibility to connect different masters with different slaves such as master 1 write in slave 2 and master 2 write in slave 1 "Simultaneously".
- Finally, the **Handshake process** check on the valid and ready signal of the chosen master and slave respectively to make the connection done.

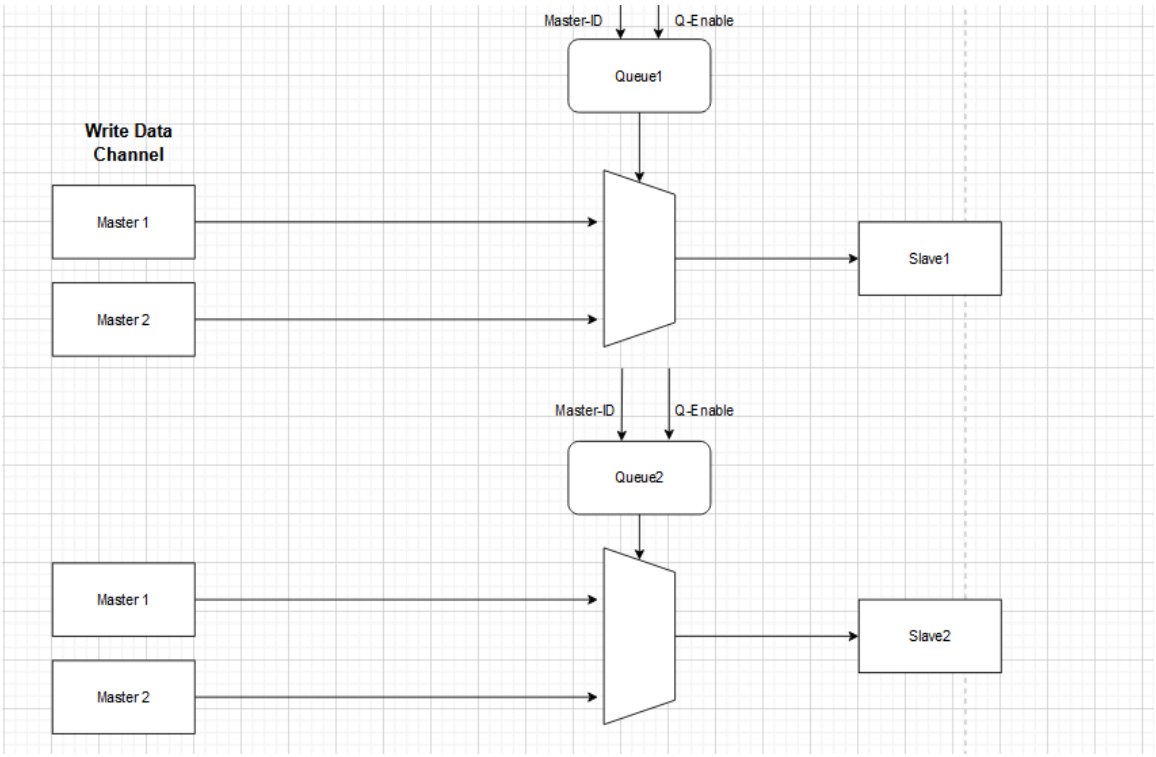

Figure 9: Write Data Channel Architecture

Write Response Channel Architecture:

In fig. (), the architecture of Write Response Channel shows how each slave responds on the write transactions to its specific master that imitated them based on its master ID.

The operation goes as following:

- First, a specific slave must be chosen to take the connection on the Write Response Channel, the **Arbiter** makes this. So it depend on the Valid signal if only one slave has asserted its Valid signal, it will be chosen to build the communication. However, if both have asserted their Valid signals, the arbiter will choose Slave 1 as higher priority.
- After the arbiter, a one slave is chosen has the master ID and the response to its write transaction. The **Decoder** uses this ID to choose which master to communicate the chosen slave with.
- Now the Handshake process, based on chosen Slave's Valid and master's Ready, the communication will be built and the information goes from specific slave to a required master.

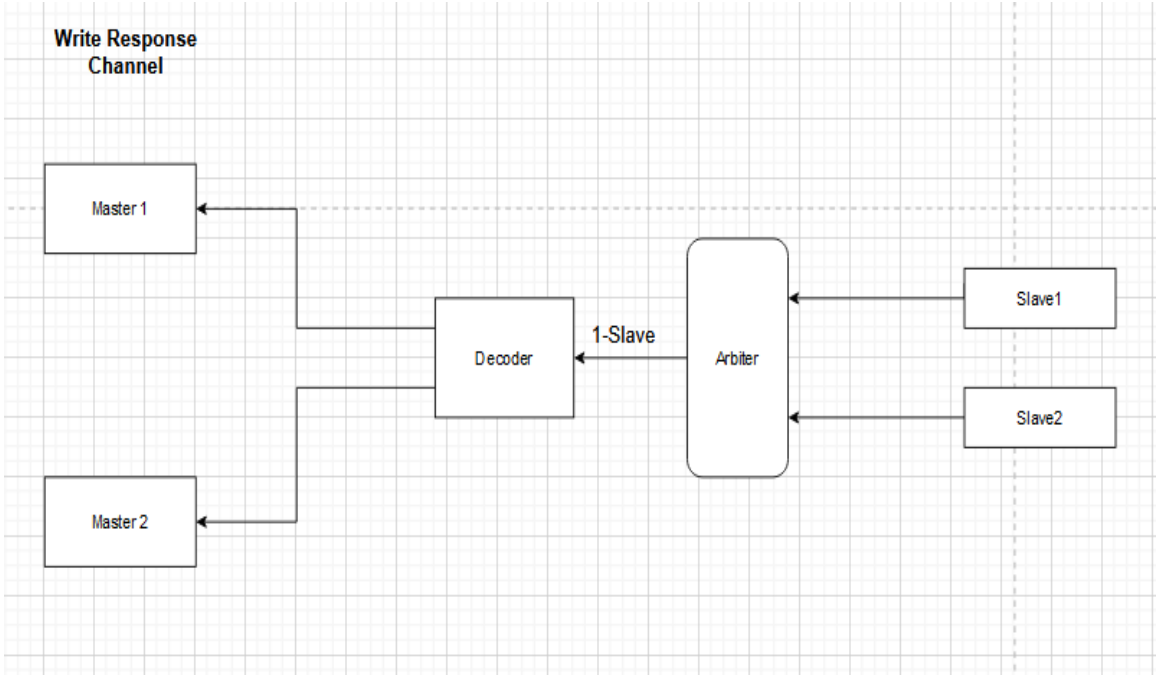

Figure 10: Write Response Channel Architecture

Read Transaction Architecture:

Final FSM shown in fig. (11).

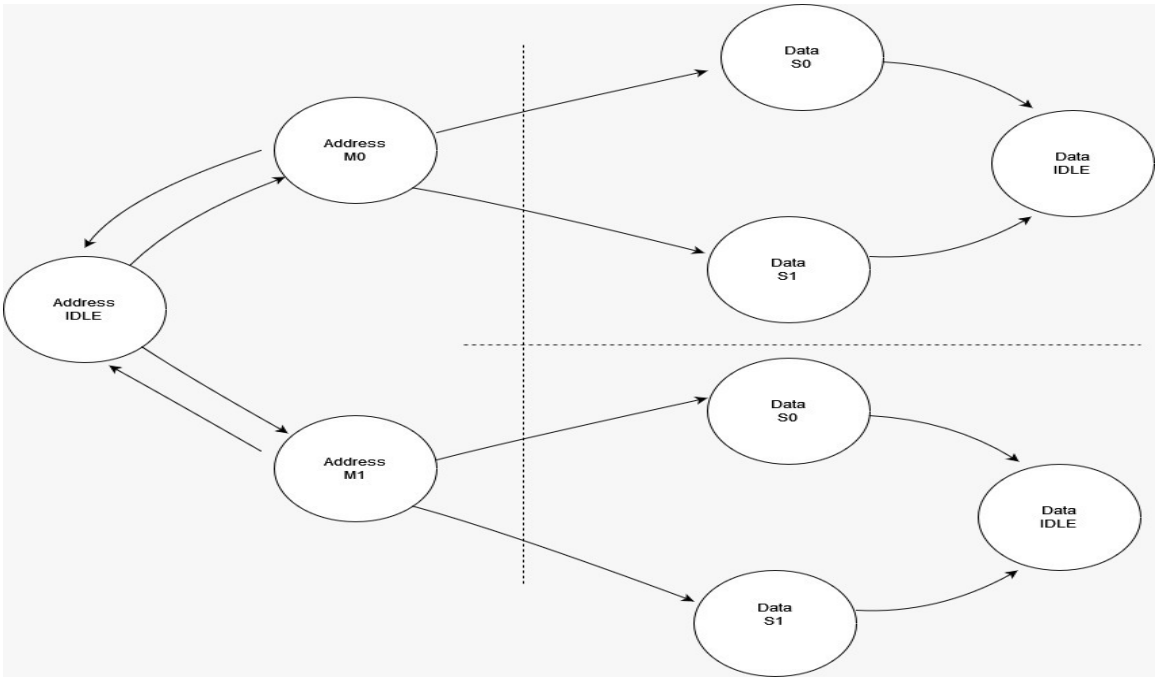

Figure 11: FSM of Read Transactions

Final Architecture shown in fig. (12), (13) and (14):

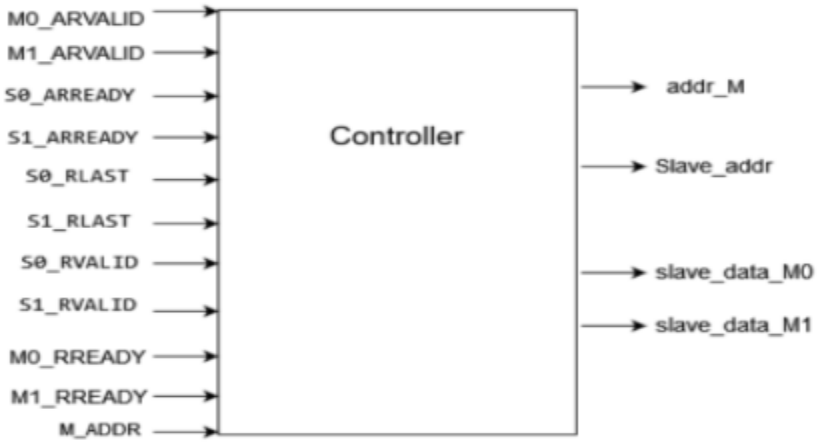

Figure 12: Final architecture block diagram

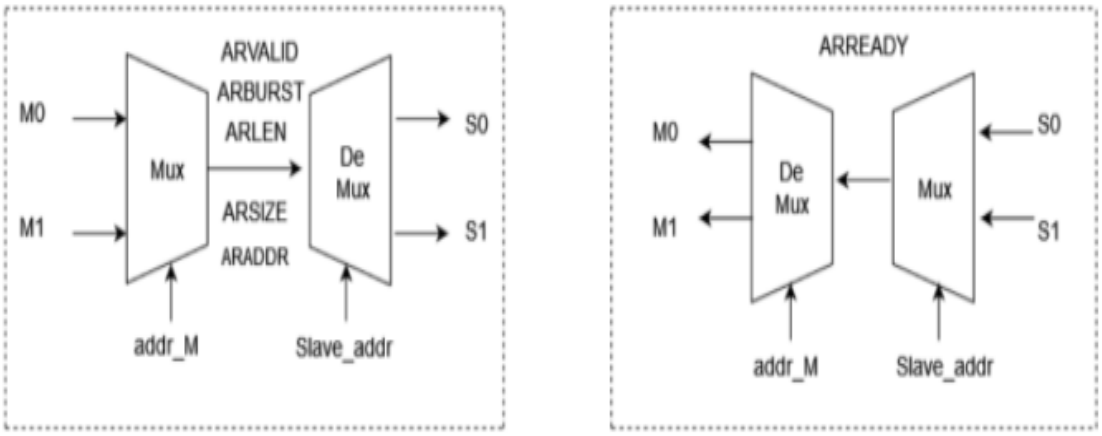

Figure 13: Final architecture block diagram

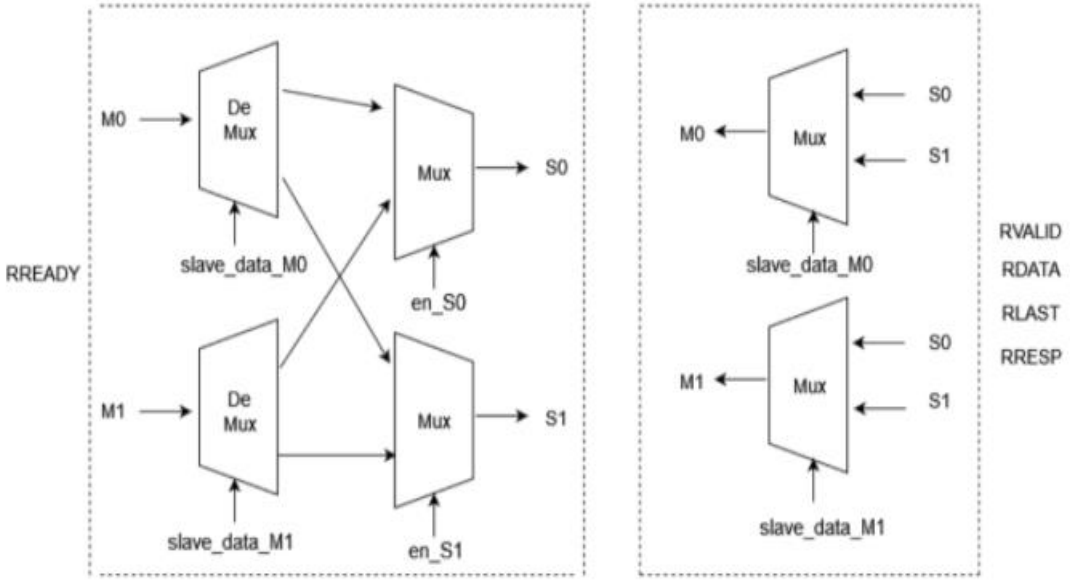

Figure 14: Final architecture block diagram

Slave architecture

FSM for the slave to be connected to IIC shown in fig. (15):

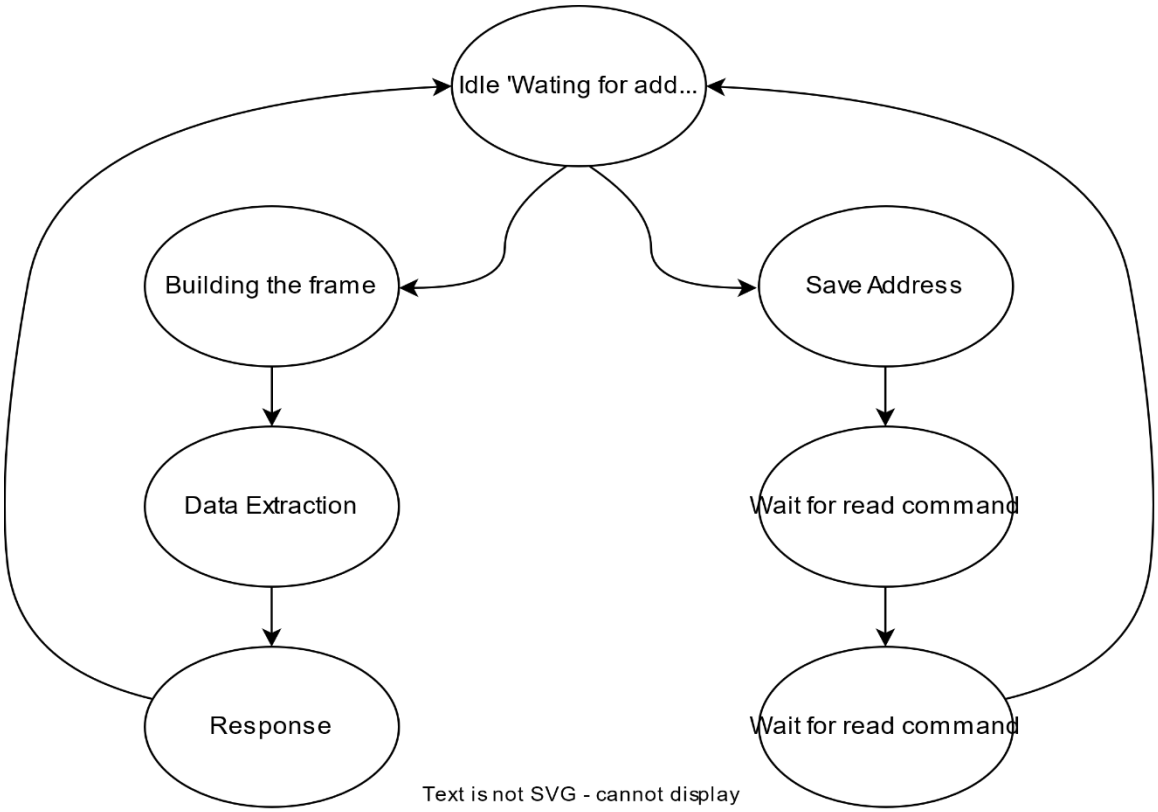

Figure 15: FSM for the slave

System Validation and Waveforms

Write Transactions Test Cases

Test Case 1:

Both Masters send a simple transfer to the same slave as shown in fig. (16).

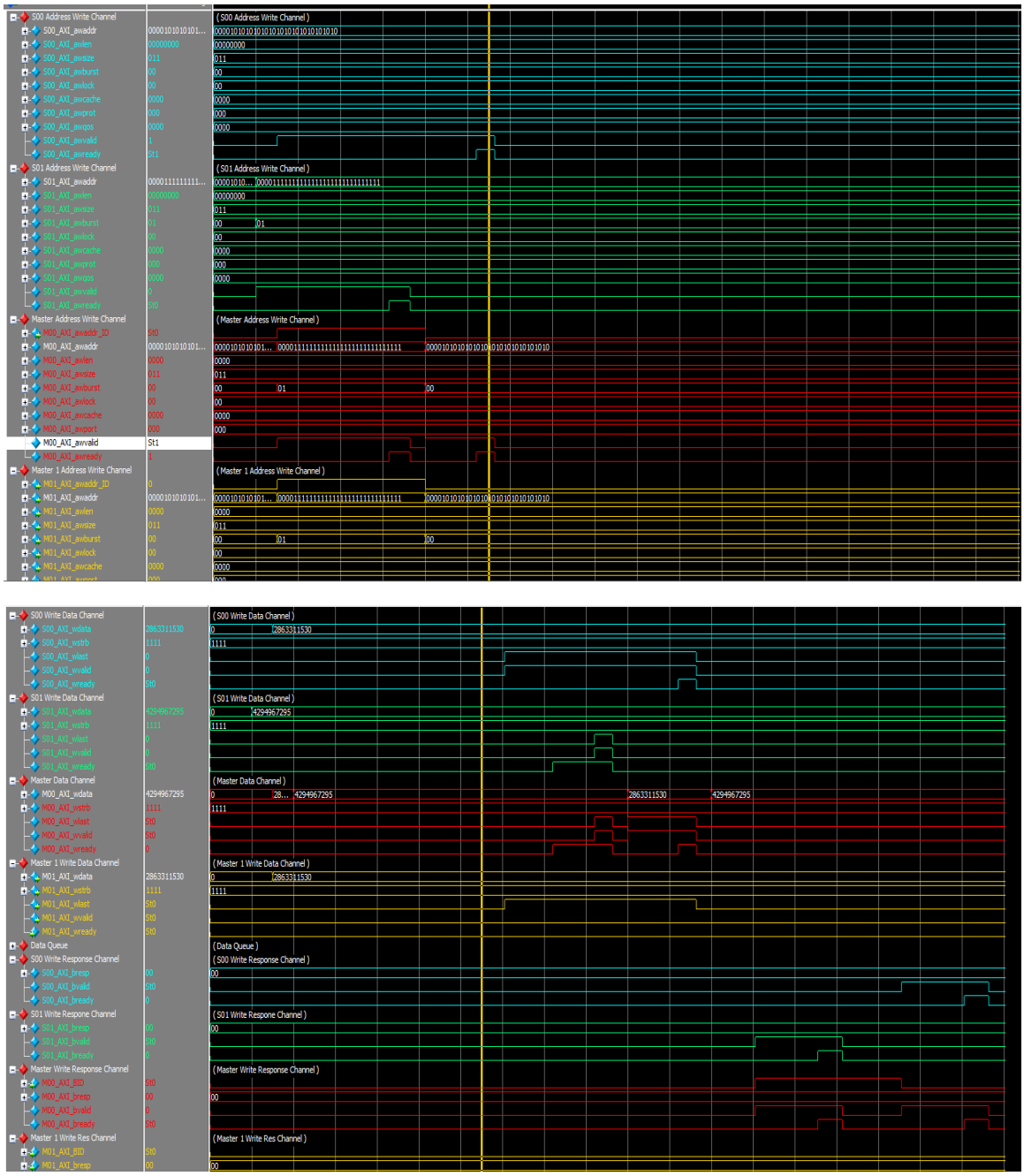

Figure 16: Write Transaction test case 1

Test Case 2:

Both Masters send a simple transfer to the other slave as shown in fig. (17).

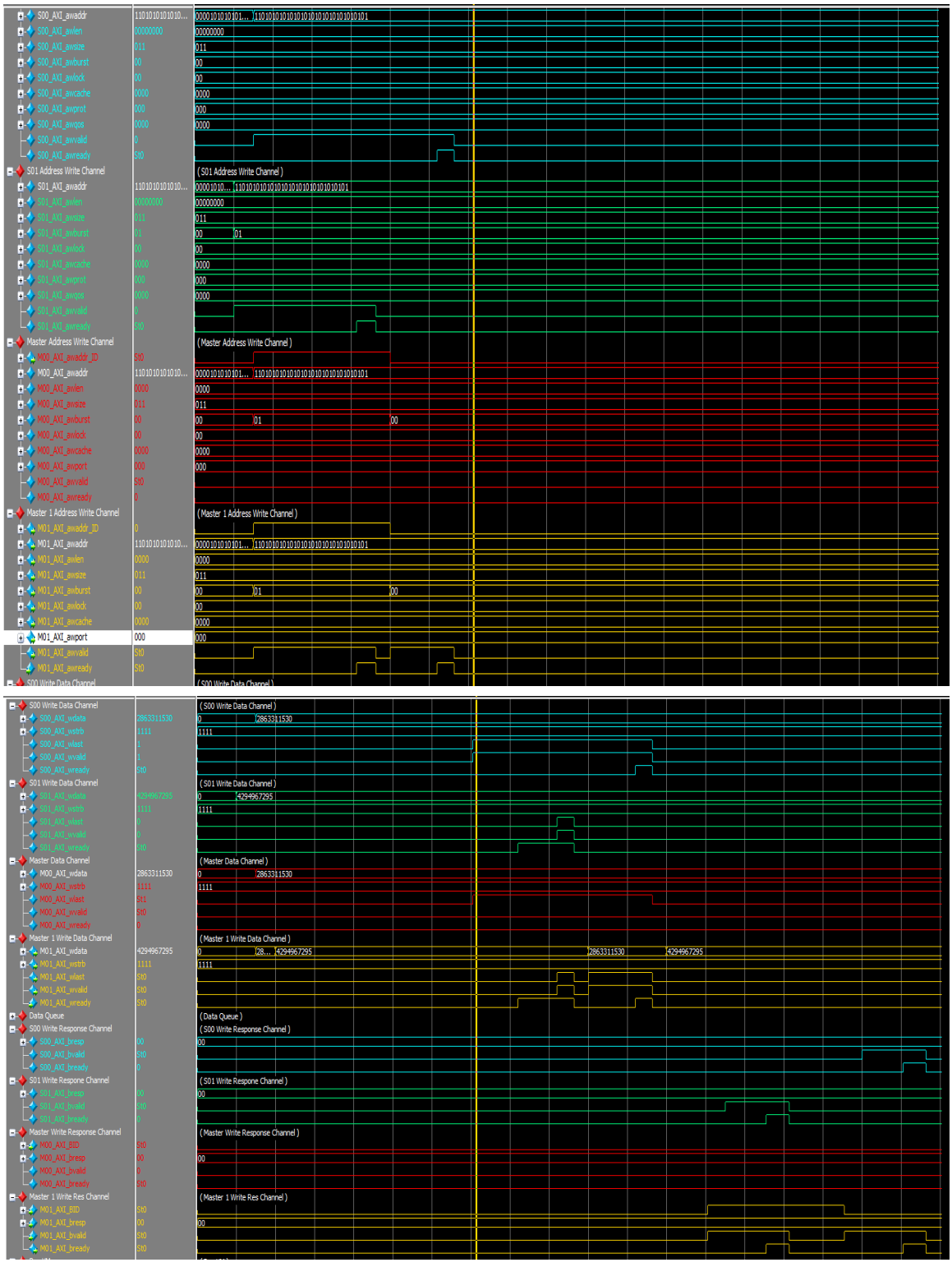

Figure 17: Write Transaction test case 2

Test Case 3:

Master 0 to Slave 0 and Master 1 to Slave 1 as shown in fig. (18).

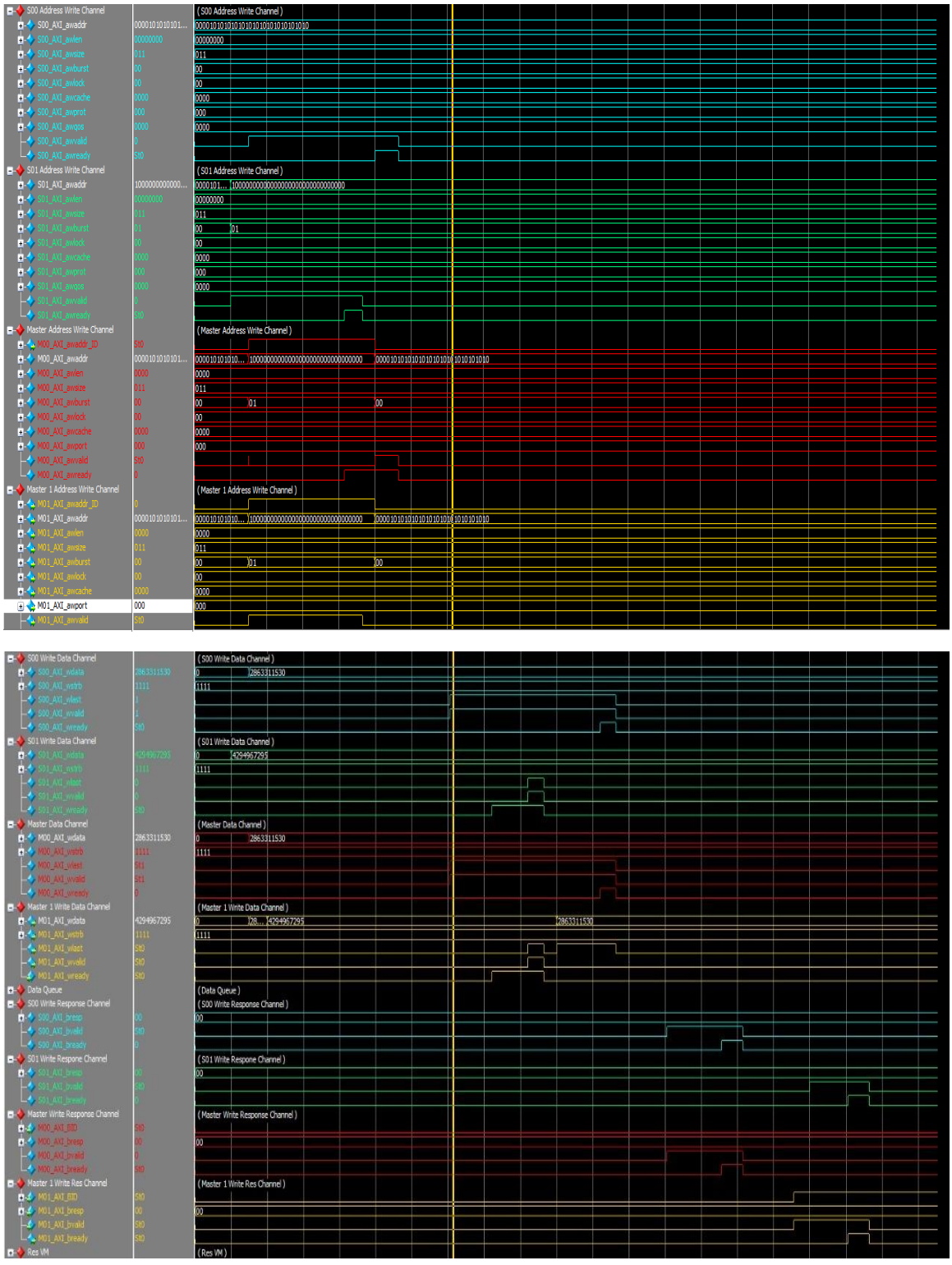

Figure 18: Write Transaction test case 3

Test Case 4:

Different QoS and One Master is sending a burst smaller than 16 as shown in fig. (19).

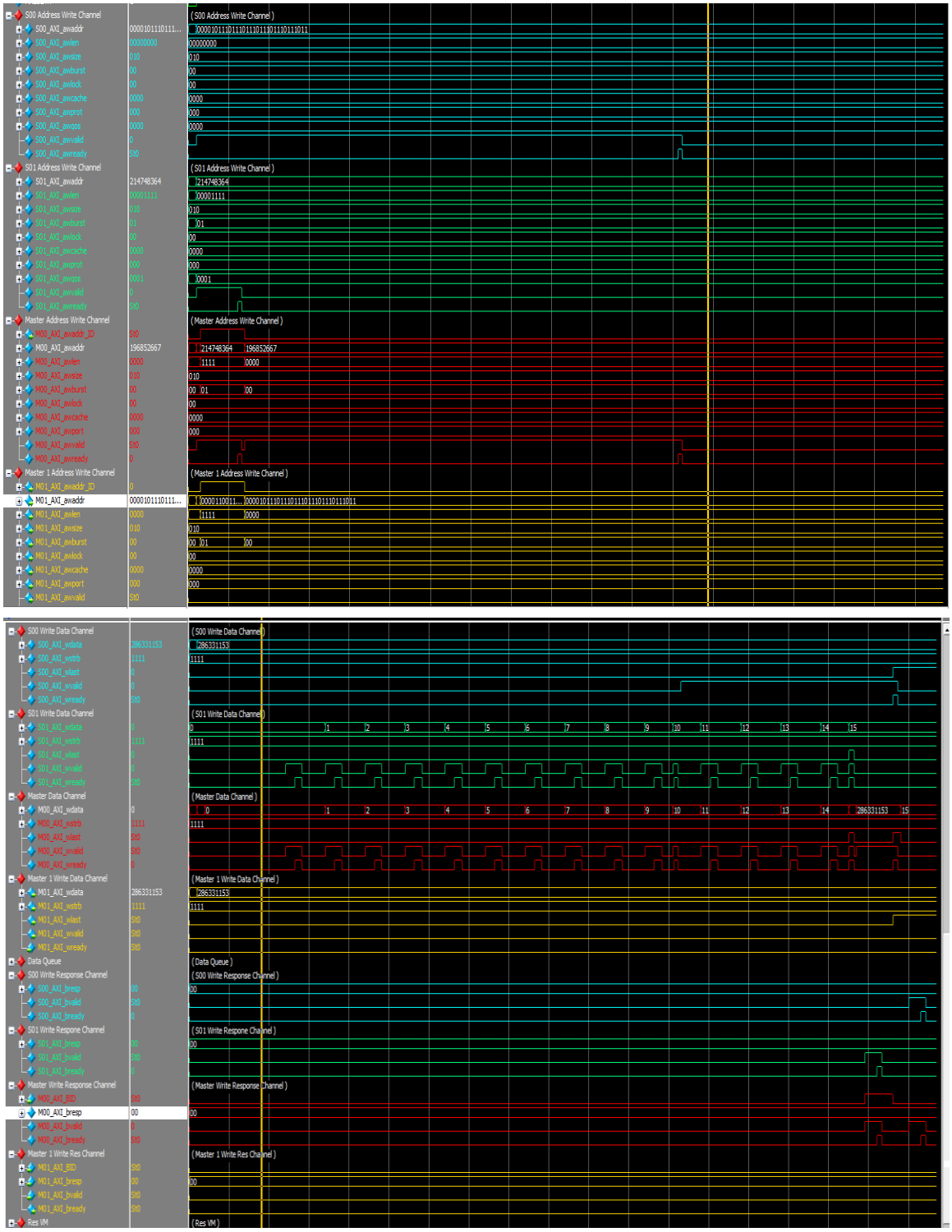

Figure 19: Write Transaction test case 4

Test Case 5:

One Master is sending a large Burst length as shown in fig. (20).

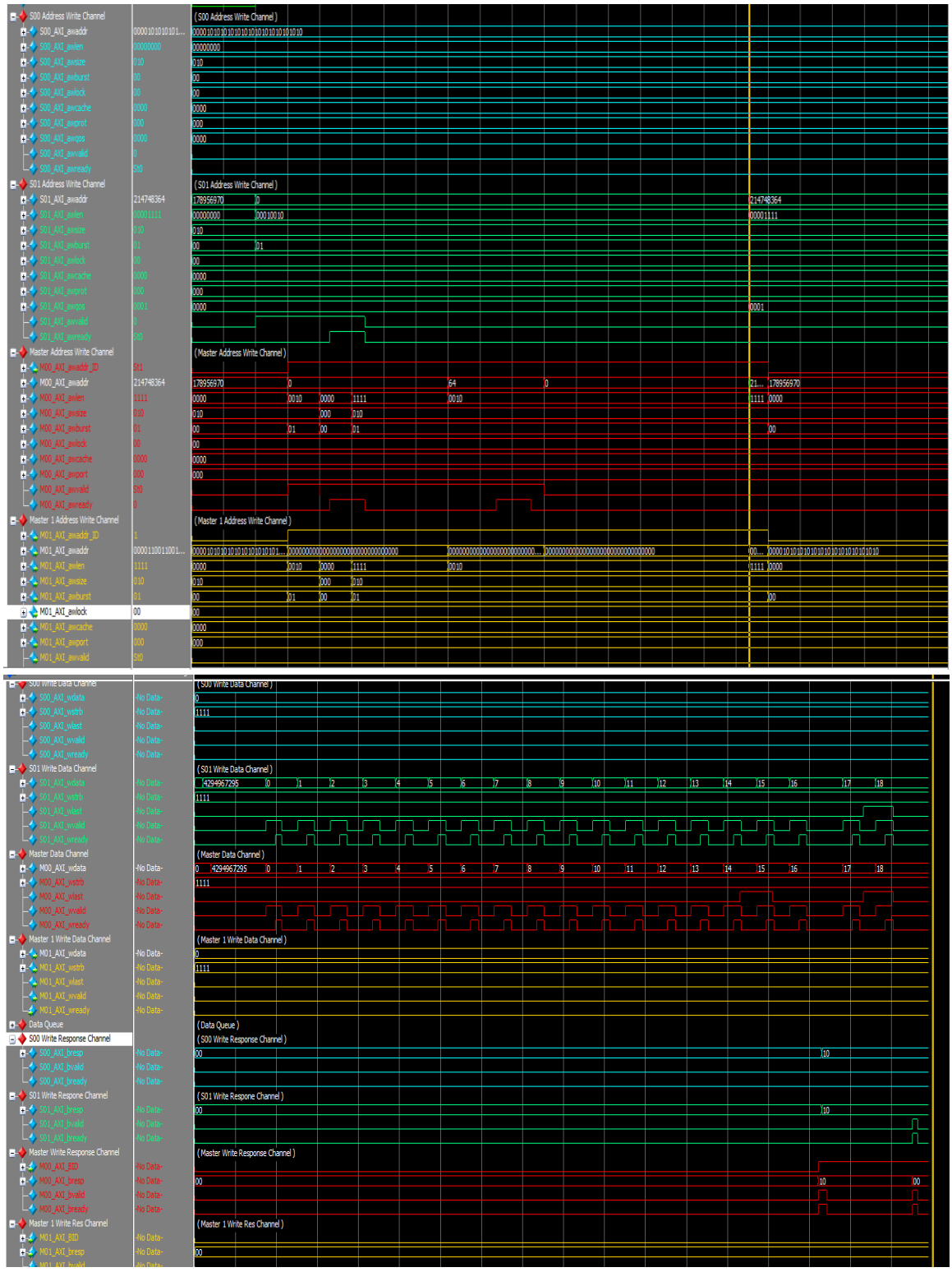

Figure 20: Write Transaction test case 5

Read Transactions Test Cases

Test Case1:

First transaction, slave 0 sends single data to master 0 shown in fig. (21).

Second transaction, slave 1 sends single data to master 0 shown in fig. (22).

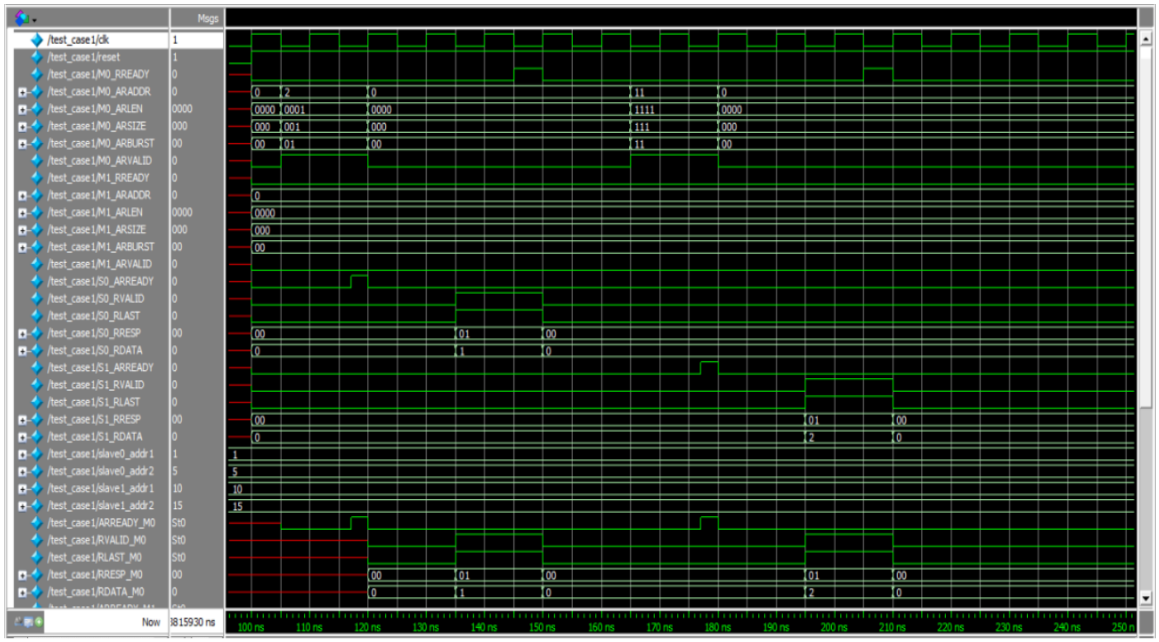

Figure 21:: Read Transaction 1 test case 1

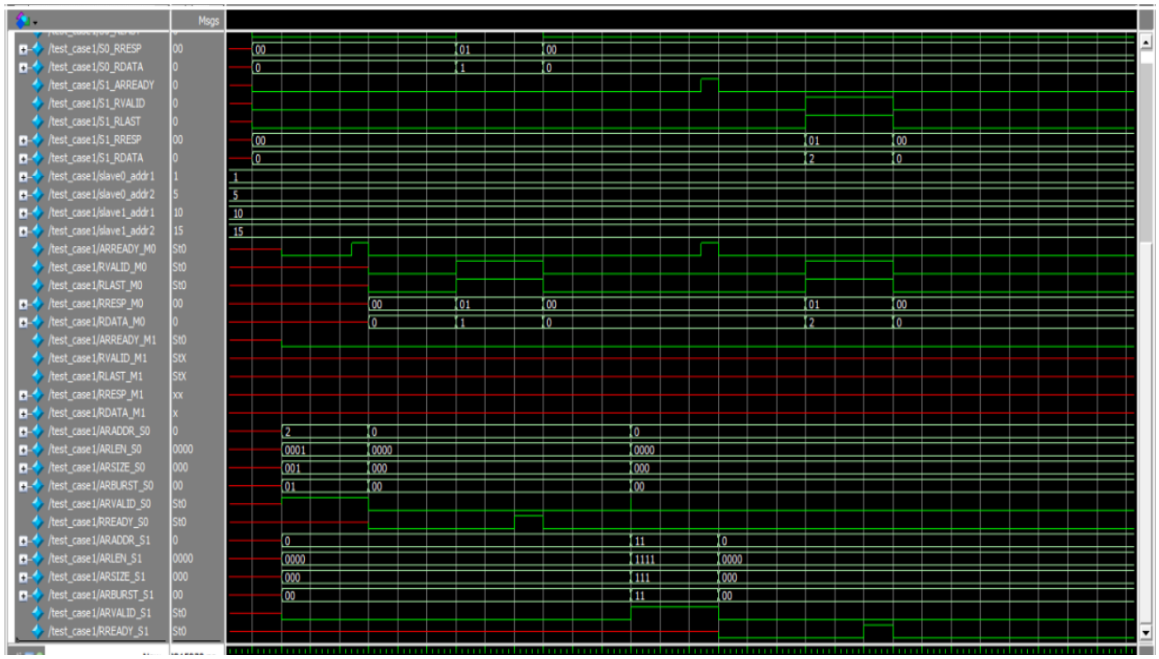

Figure 22: Read Transaction 2 test case 1

Test Case 2:

First transaction, slave 0 sends single data to master 0 as shown in fig. (23).

Second transaction, slave 1 sends single data to master 1 as shown in fig. (24).

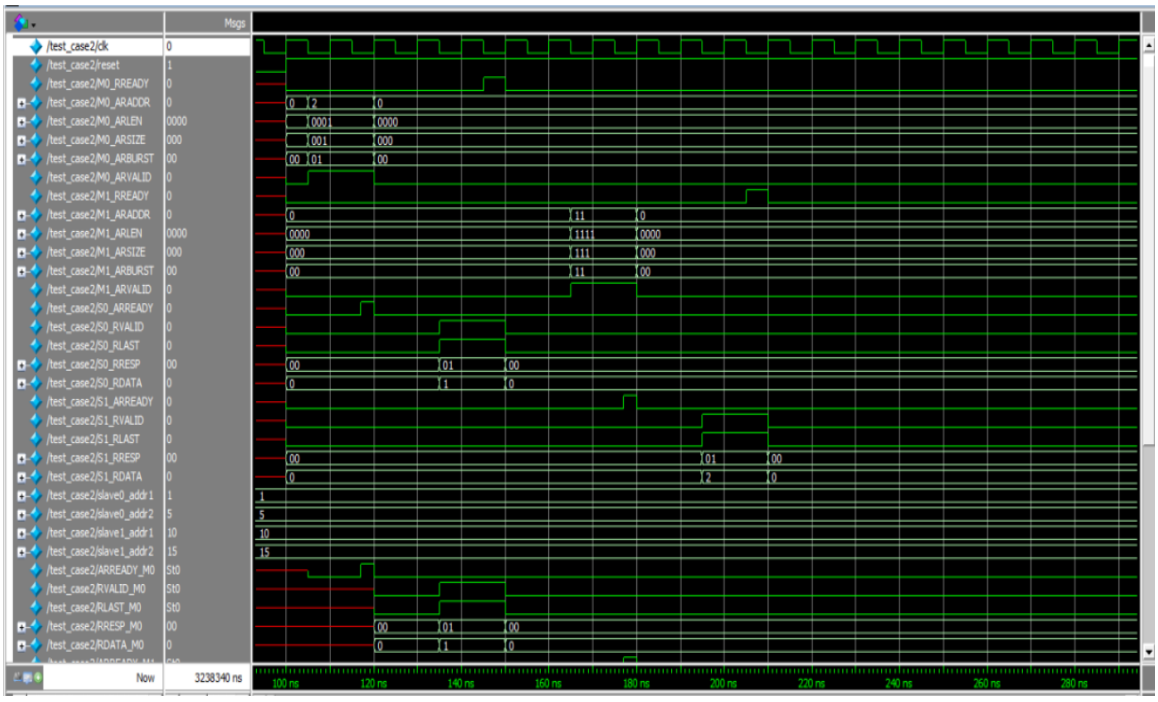

Figure 21: Read Transaction 1 test case 2

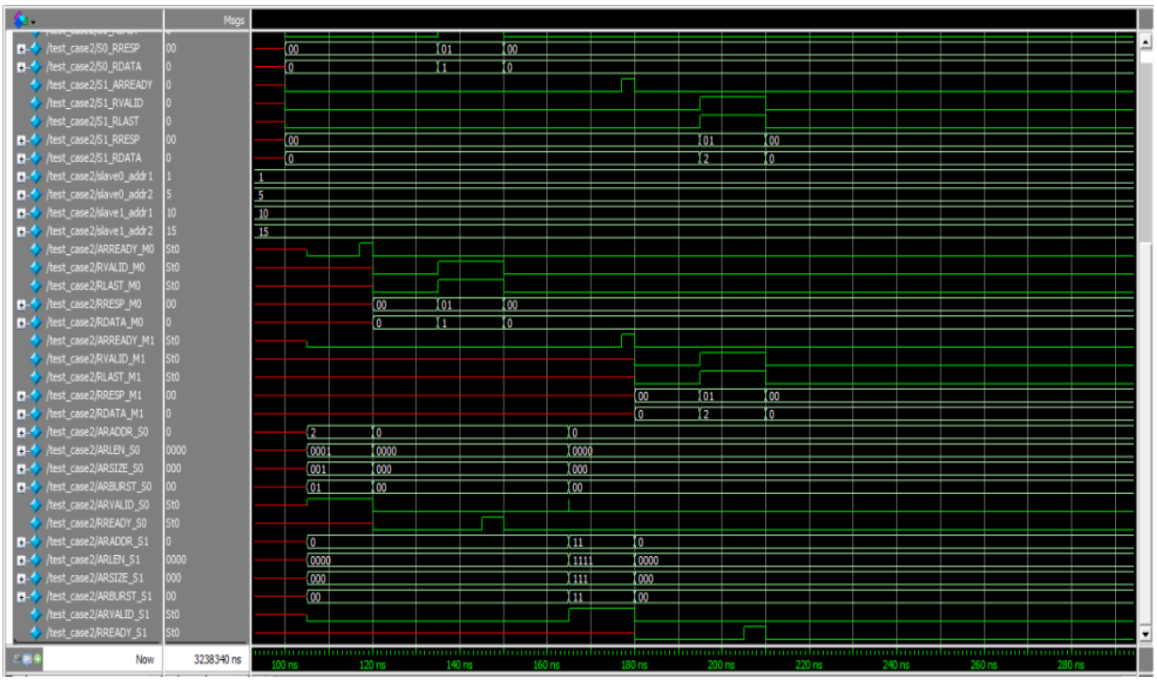

Figure 22: Read Transaction 2 test case 2

Test case 3:

First transaction, slave 1 sends single data to master 0 as shown in fig. (25).

Second transaction, slave 0 sends single data to master 1 as shown in fig. (26).

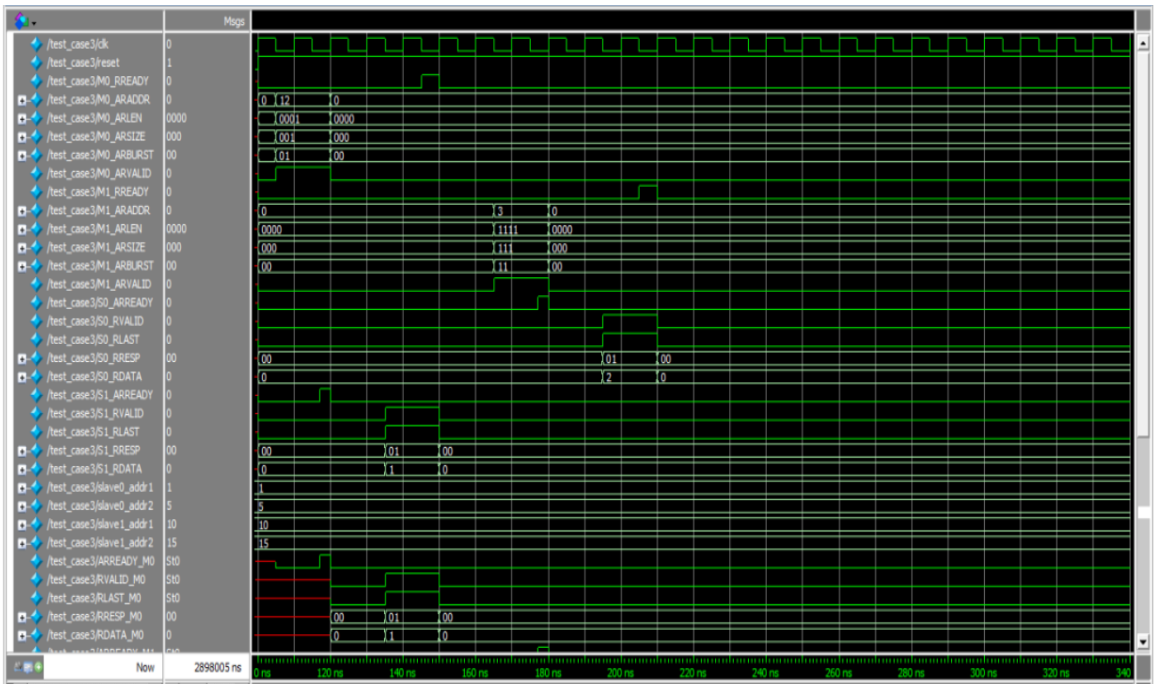

Figure 23: Read Transaction 2 test case 3

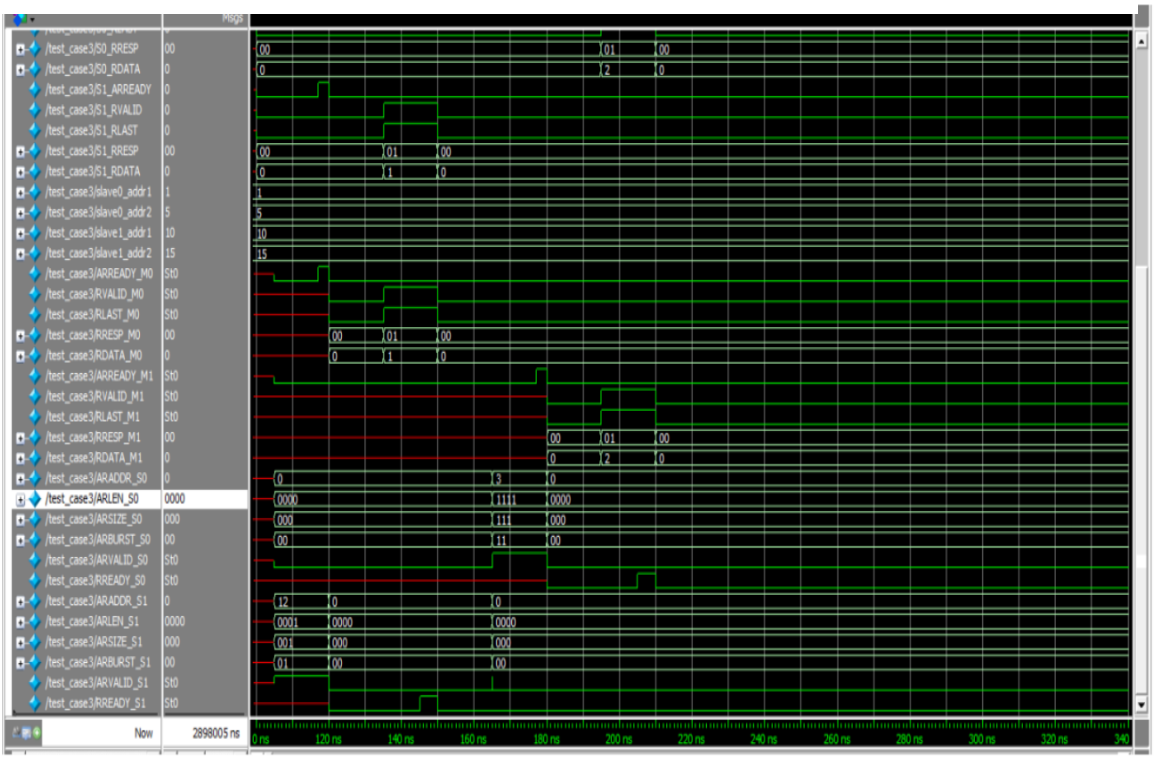

Figure 24: Read Transaction 1 test case 3

Test case 4:

First transaction, slave 1 sends burst data to master 0 as shown in fig. (27).

Second transaction, slave 0 sends burst data to master 1 as shown in fig. (28).

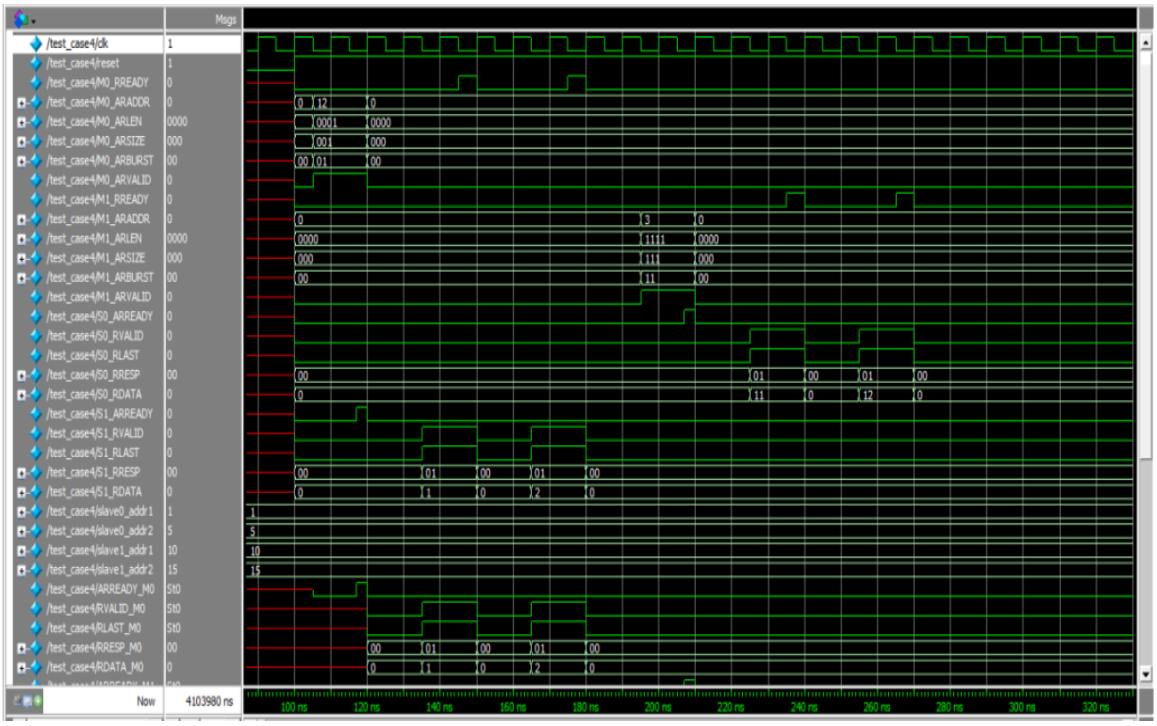

Figure 25: Read Transaction 1 test case 4

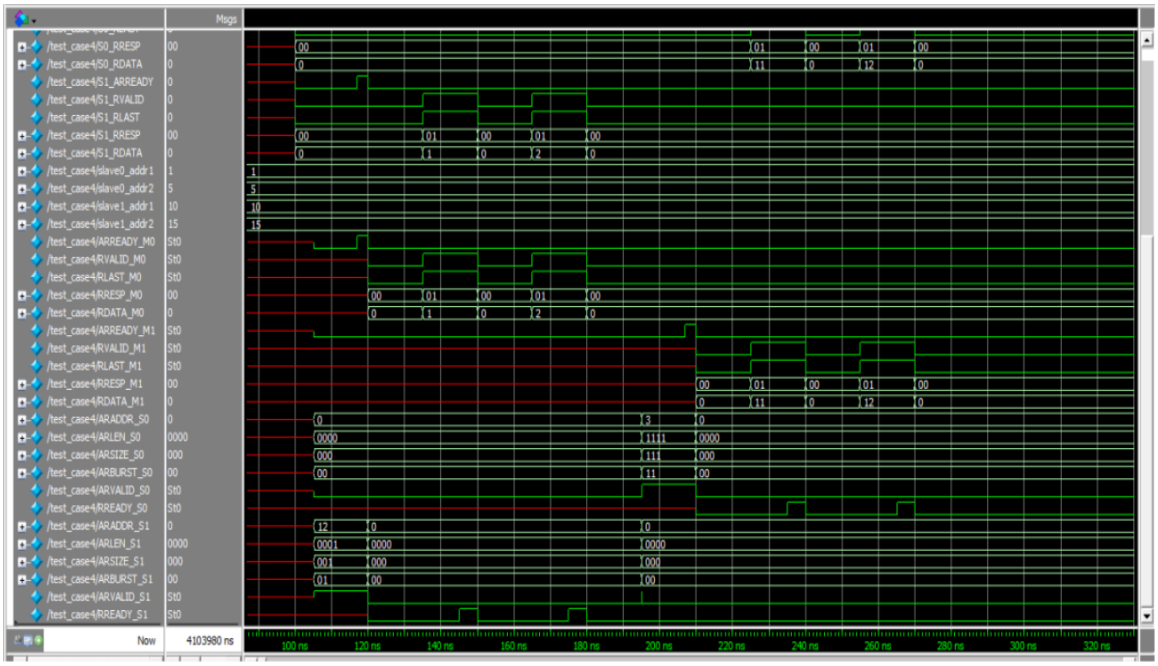

Figure 26: Read Transaction 2 test case 4

Test case 5:

First transaction, slave 1 sends burst data to master 0 as shown in fig. (27).

Second transaction, slave 0 sends burst data to master 1 as shown in fig. (28).

The difference here is that the second address is sent before first transaction ends and both transactions intersect at a point where we have two masters reading from two different slaves.

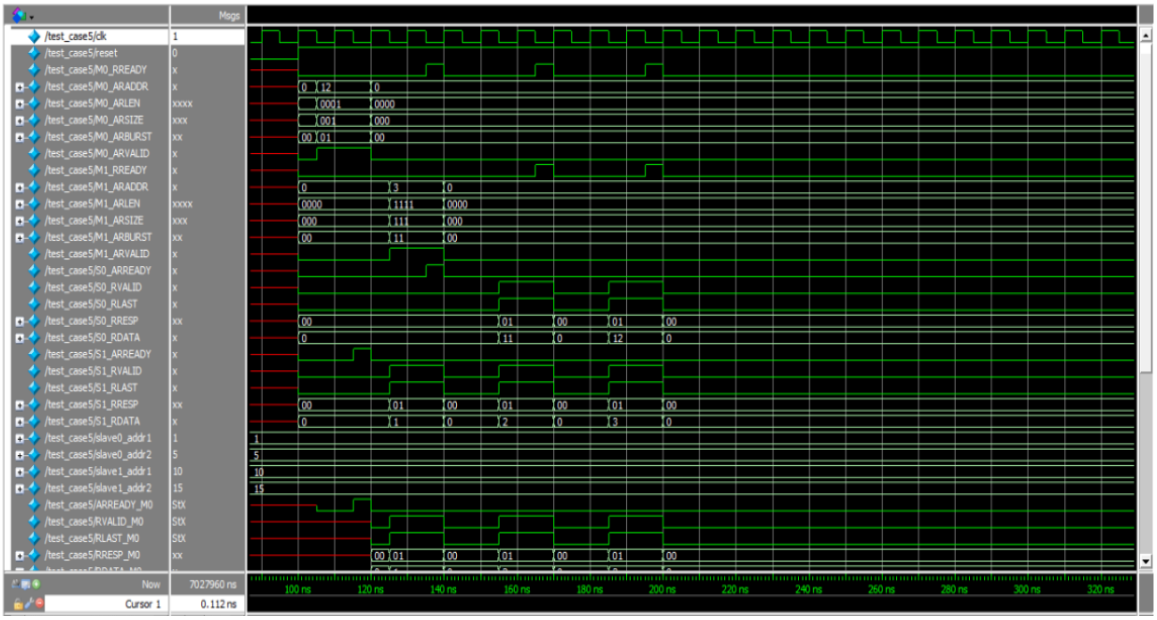

Figure 27: Read Transaction 1 test case 5

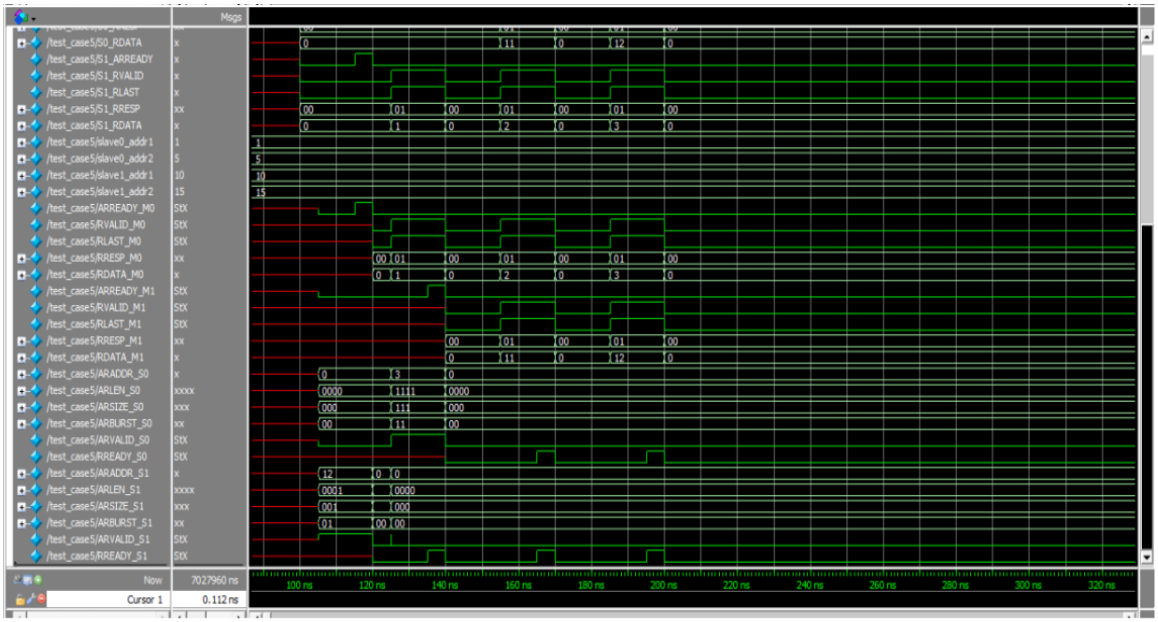

Figure 28: Read Transaction 2 test case 5

Synthesis

Constrain	Value	Constrain	Value
Clock Frequency	100MHz	Input and Output Delay	20% of the clock period
Input driving cell	BUF2X	Generated Clocks	All the other clocks with div ratio of 1

Timing report:

Worst Slack for the setup is 3.3 and for the Hold is 0.32, you can see the whole reports at the drive.

Area report:

```
Number of ports:                         3112
Number of nets:                          4426
Number of cells:                         1793
Number of combinational cells:           1612
Number of sequential cells:               126
Number of macros/black boxes:               0
Number of buf/inv:                        677
Number of references:                      30

Combinational area:               12435.365703
Buf/Inv area:                      4657.378643
Noncombinational area:             3106.488018
Macro/Black Box area:                 0.000000
Net Interconnect area:           208974.388397

Total cell area:                  15541.853721
Total area:                      224516.242118
```

Power report:

Power Group	Internal Power	Switching Power	Leakage Power	Total Power (%)	Attrs	Cell Count
io_pad	0.0000	0.0000	0.0000	0.0000 (0.00%)		0
memory	0.0000	0.0000	0.0000	0.0000 (0.00%)		0
black_box	0.0000	0.0000	0.0000	0.0000 (0.00%)		0
clock_network	0.0000	0.0000	0.0000	0.0000 (0.00%)		0
register	0.2275	3.6994e-03	6.7027e+05	0.2318 (16.16%)		110
sequential	7.9123e-04	1.3716e-04	5.2266e+04	9.8066e-04 (0.07%)		10
combinational	0.1271	1.0718	3.2632e+06	1.2022 (83.78%)		1612
Total	0.3554 mW	1.0756 mW	3.9858e+06 pW	1.4350 mW		

Glossary

Expression	Definition
ARM	"Advanced RISC Machine" is an architectural firm with offices in Melbourne, Sydney, and Adelaide, Australia. The firm was founded in 1988 and has completed internationally renowned design work.
AMBA	"Advanced Microcontroller Bus Architecture" is a freely available, open standard for the connection and management of functional blocks in a System-on-Chip (SoC).
AXI	"Advanced eXtensible Interface" is an on-chip communication bus protocol developed by ARM.
Burst Transaction	It means transmitting data repeatedly without going through all the steps required to transmit each part of data in a separate transaction with only the address of the start one only and it will go through others.
Fixed Burst	In Fixed, the address is the same for every transfer in the burst. This burst type is used for repeated accesses to the same location such as when loading or emptying a FIFO.
Incrementing Burst	In incrementing, the address for each transfer in the burst is an increment of the address for the previous transfer. The increment value depends on the size of the transfer. This type is used for accesses to normal sequential memory.
Wrap Burst	A wrap burst is similar to an incrementing burst, except that the address wraps around to a lower address if an upper address limit is reached.

Conclusion

In this design document, AXI protocol is well explained then designing the AXI interconnect with many supported features and implemented using Verilog HDL combined with the design and implementation of the AXI slave to be connected to the IIC module. All test cases are declared and simulated using ModelSim. Then, applying Synthesis to the whole design at frequency 100 MHz and met all timing constraints with a positive slack. Finally, extract Timing, Area, and Power reports and discussing them.

References

AMBA AXI and ACE Protocol Specification - Arm