Evaluation Report Google summer of code 2020

Apoorva Arora

August 31, 2020

Abstract

This report gives detailed design, test bench implementation as well as simulation results of VHDL based packet protocol implementation of bidirectional serial LVDS communication between master and slave device.

Introduction

Aim of the project is to design a packet based bidirectional protocol over single LVDS link that can fully utilize the available band width based on priority based task scheduling.

- The project is mainly divided into 3 major phases. First bidirectional physical layer implementation. Second, packet layer implementation. Third, task scheduler. During the first phase of GSOC-20 bidectional physical layer and packet layer have been implemented.
- Testbench simulations have been performed to demonstrate access of SPI communication on the slave side by the master side over the packet based communication.
- The design described in this report aims to avoid cross-talk between master and slave devices over the same bidirectional link as well as provide a standard handshake based interface to the master user side to access the service on slave device via virtual address technique.

1 PHASE 1

This section describes development made during phase 1 of the project.

1.1 Methodology - Top level

1. The **Scheduler** accepts user commands including address as well as bursts size. Based on the priority of the service, a specific FIFO is used to store the data of corresponding service (address).
2. The scheduler then redirects specific command as well as FIFO link to the **packet layer** which parses the command to generate the required packets.
   - First packet transferred over the LVDS link is command packet which specifies IO address as well as burst size.
   - Then based on the command either data is written to the LVDS link or is read from it. The schedulers on both the master as well as slave sides ensures that there is no bus contention.
3. The physical layer **PHY** acts as bidirectional SERDES. The master PHY produces both clock and data while slave PHY produces/receives data on the clock provided by master.

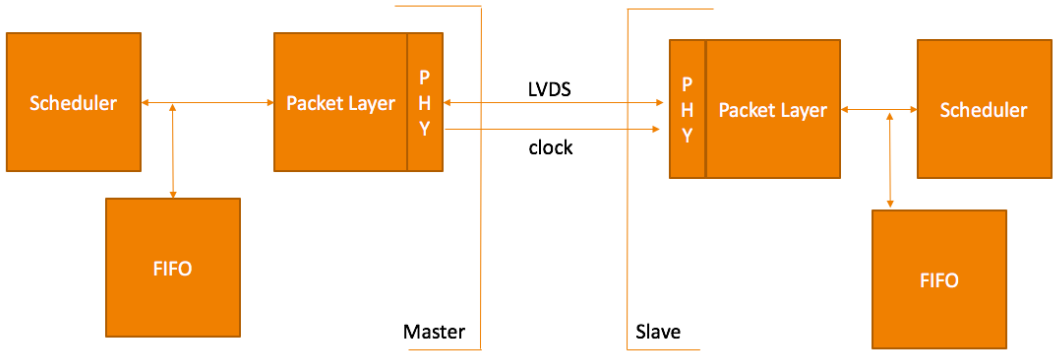

Figure 1: Top level system architecture

1.2 Bidirectional physical layer design

- This layer acts as bidirectional SERDES for the user.
- There are two major modules involved mainly master and slave.
- Master produces clock while slave transmits and receives data on that clock.

Both master as well as slave FSMs are initiated on write/read transaction enable signals from upper layers (in this case the packet layer). Hence, the upper layer has full control over the SERDES in terms of avoiding any possible cross-talks.

The schematics as well as FSM for both master and slave modules are shown below.
PHY Master code: Github link
PHY Slave code: Github link

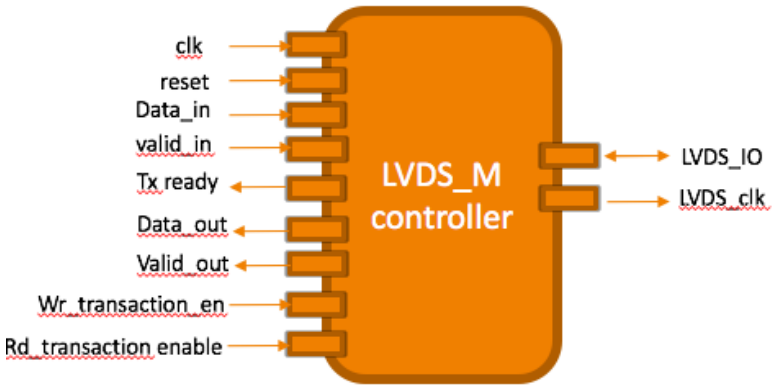

Figure 2: Master LVDS PHY SERDES

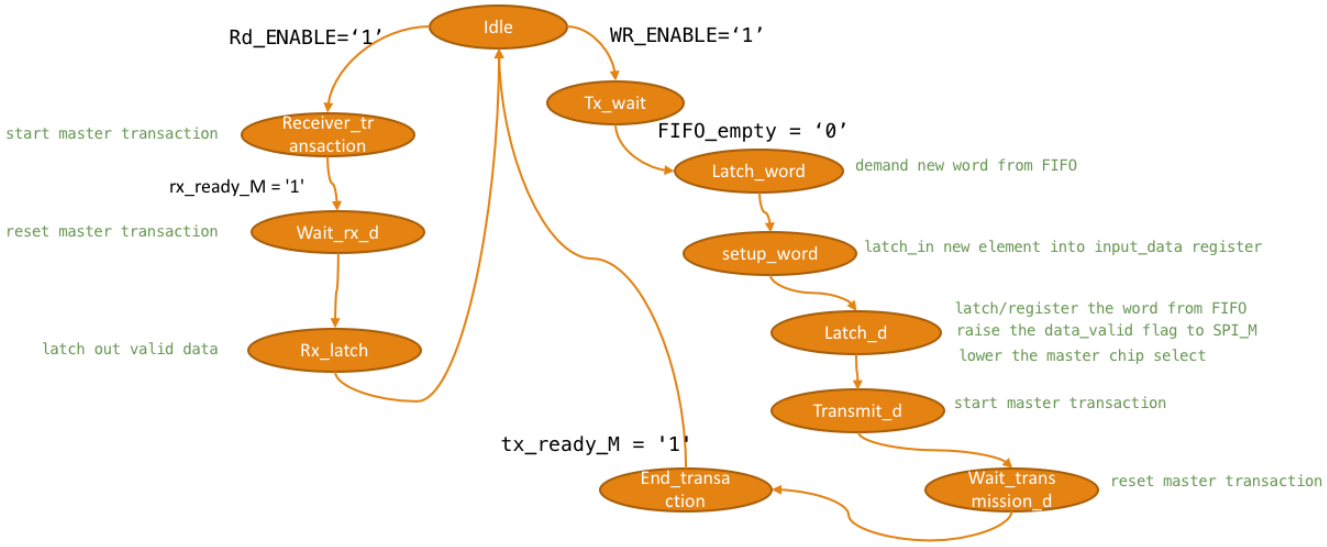

Figure 3: Master LVDS PHY SERDES-FSM

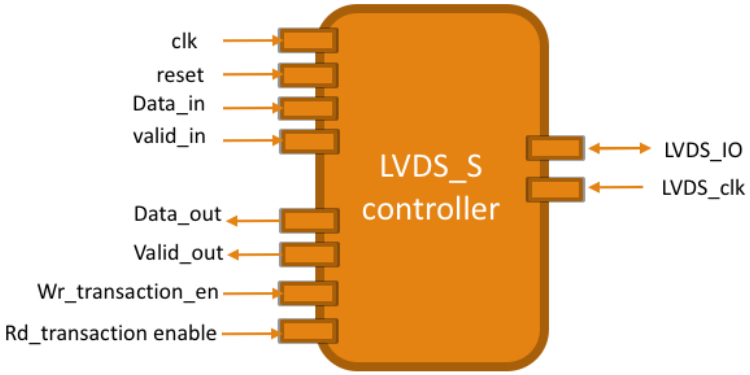

Figure 4: Slave LVDS PHY SERDES

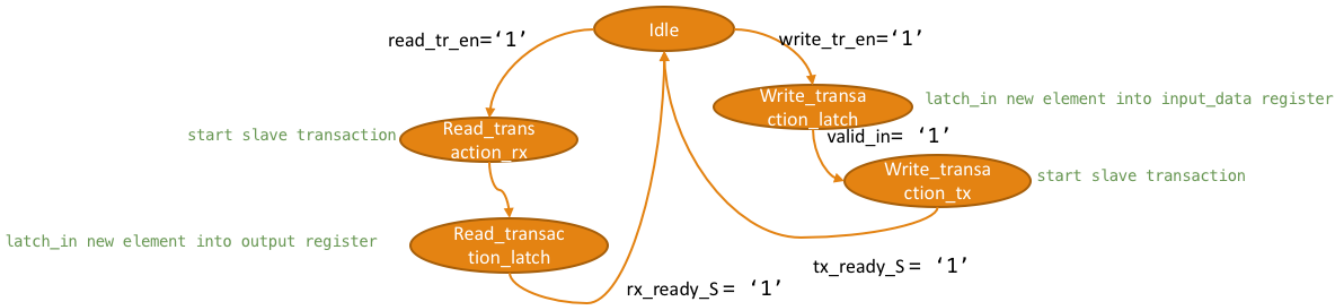

Figure 5: Slave LVDS PHY SERDES-FSM

1.2.1 Testbench construction and results

To test the PHY layer a VHDL test-bench is created where. User latches in vector data on master side and slave receives it over serial link on clock generated by master and visa versa. This verifies the "Garbage in Garbage out" functionality of PHY layer. The test bench structure as well as simulation screen shot in shown below.
Test-bench code: Github link

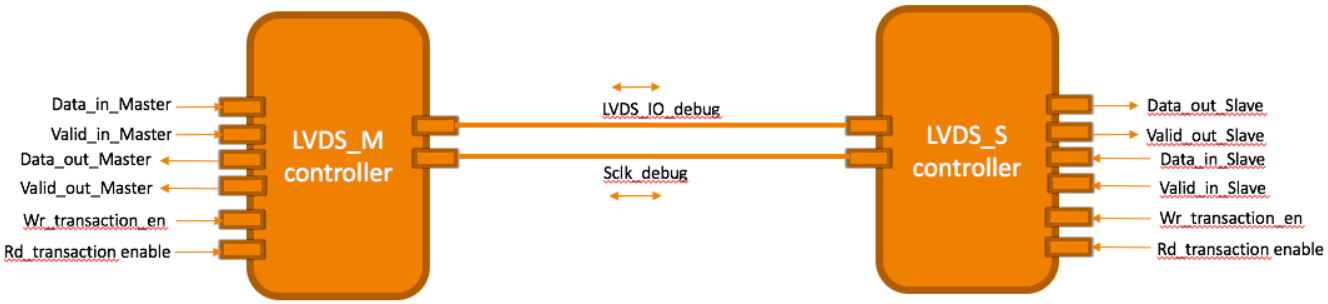

Figure 6: PHY testbench

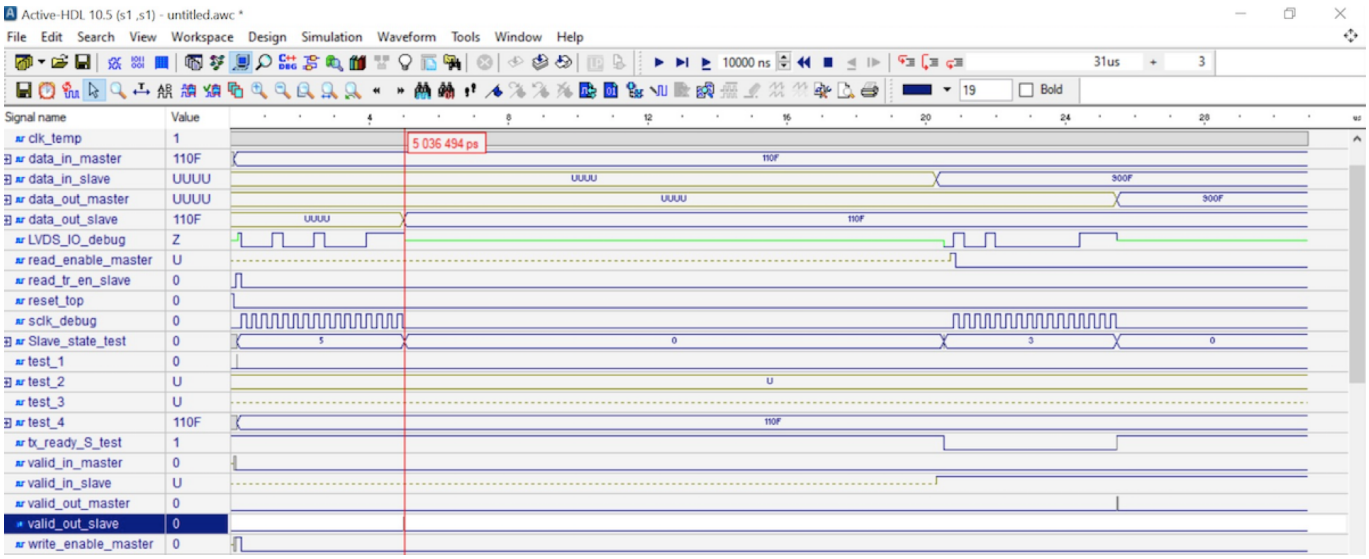

Figure 7: PHY testbench simulations screen shot

1.3 Packet Layer

This layer accepts commands from upper layers namely scheduling layer in order to control data flow of the physical layer.

- The command contain virtual address, burst length and read/write information.
- The packet layer FSM ensures a "command packet" is first transmitted over the LVDS serial link so that the data flow between master and slave devices is synchronised and controlled.
- The master packet layer FSM decodes the command in order to generate/receive data packets (burst).
- The slave packet layer on the other hand reads command packet and then demands/produces data from/to the slave peripheral.

Packet layer Master code: Github link
Packet layer Slave code: Github link

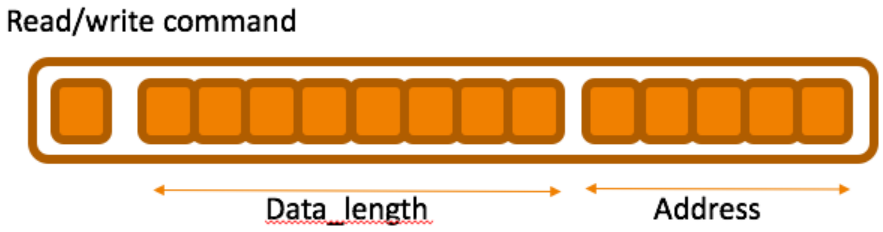

Figure 8: command packet

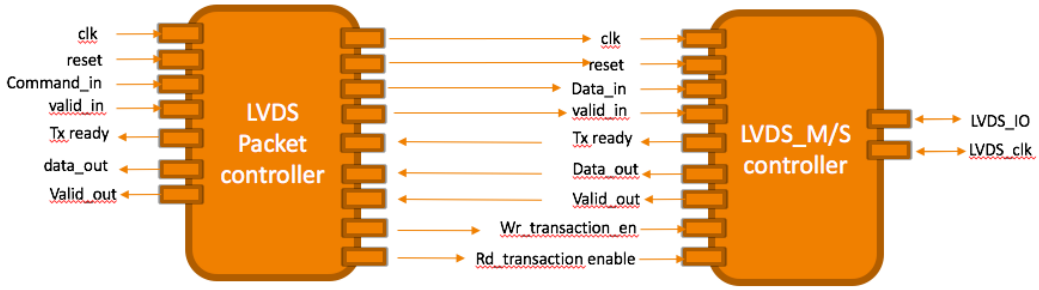

Figure 9: packet layer

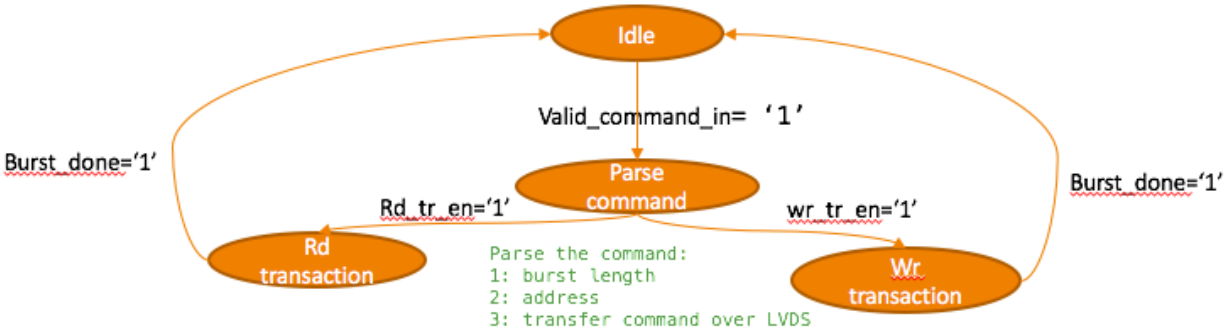

Figure 10: packet layer FSM

1.3.1 Testbench

In order to test the packet layer functionality as well as latency. A command to write 3 elements of data at x address of slave and subsequently read 3 elements from x address of slave is stimulated at command port of the master.

- The test bench simulation results clearly shows transfer of three elements (16 bit) from master to slave and then reading of three elements from slave by the master. Testbench code: Github link
- The slave produces valid command data as well as valid data on successful reception while master is able to produce valid data element vector on successful reception. Hence, this testbench (Fig: 11) clearly verifies the packet layer functionality. Testbench code: Github link
- In order to simulate an application. An SPI master module is attached to the slave. Then, The master is forced to generate command and transfer three elements via the serial link to the SPI peripheral of the slave.
- The simulation results (Fig: 12) clearly show that the SPI master is able to detect data from the LVDS master and is able to produce it on the SPI line.

1.4 Challenges and solutions

- The major hurdle was to come with a plan that can avoid cross talk on the bidirectional link. Hence, Master Slave based communication was adapted and extra packet layer was introduced to control the bidirectional burst data flow.
- Handshaking signals between various steaming modules sometimes stalled the state machines because of deadlock conditions. To solve this handshaking signals were carefully re-timed (whenever required) in order to avoid stalling to corresponding state machines.

2 PHASE 2

This section describes developments during phase-2 of the project. It mainly involves two parts:-

- Scheduler design
- Hardware testing

2.1 Scheduler Design

2.1.1 Algorithm and RTL construction

The main idea behind scheduling is to ensure that high priority tasks get served before lower priority tasks. AS shown in figure: 13, input command is captured from software via AXI lite register. Based on the priority of the service mentioned in the scheduler, the data latch FSM stores the command word in respective priority FIFO and the subsequent data is further stored in relevant FIFO based on the priority.

The master user FSM concurrently checks availability of data in command FIFOs based on priority (hence the command from higher priority is parsed first). After extracting and latching in the command, the burst information, read/write transaction information as well as data (from relevant FIFO) is passed on the packet layer to continue the transaction.

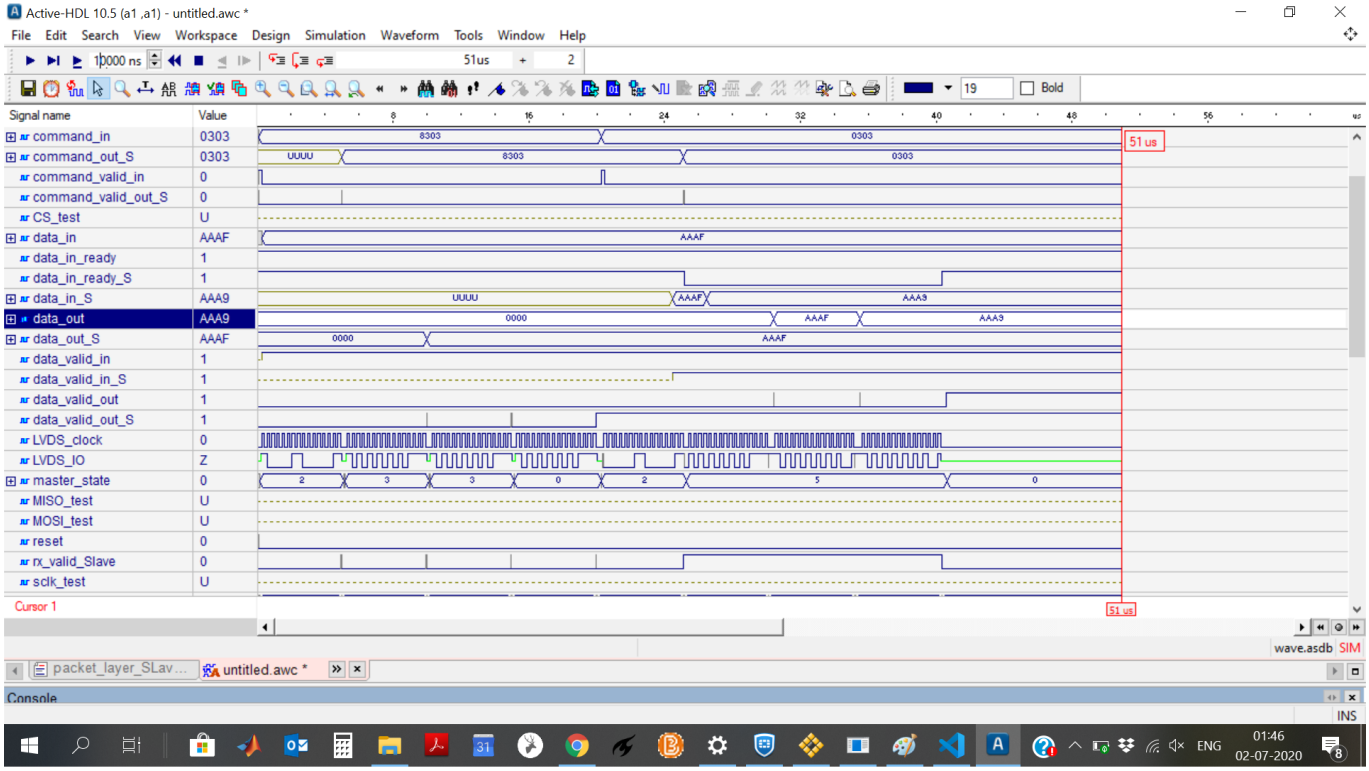

Figure 11: Packet layer read write simulations

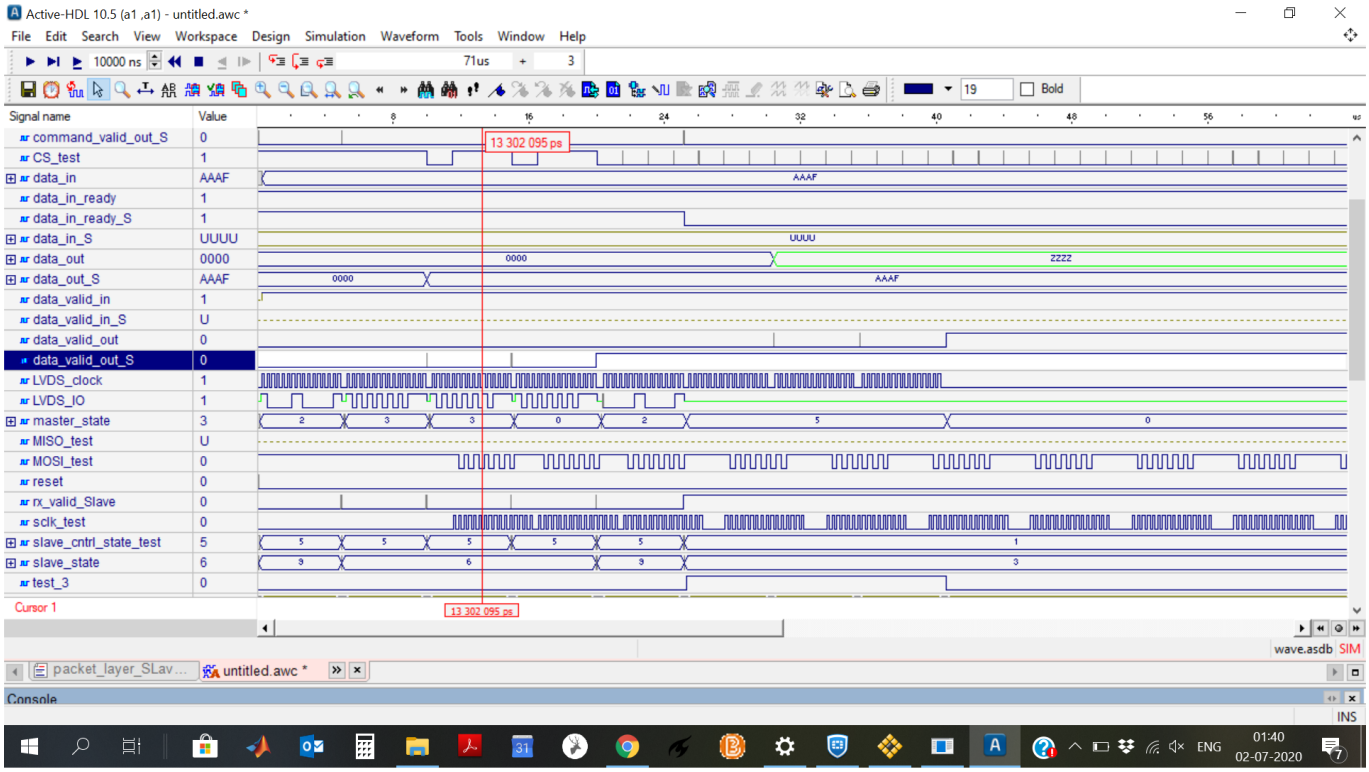

Figure 12: Packet layer SPI(M) write simulations

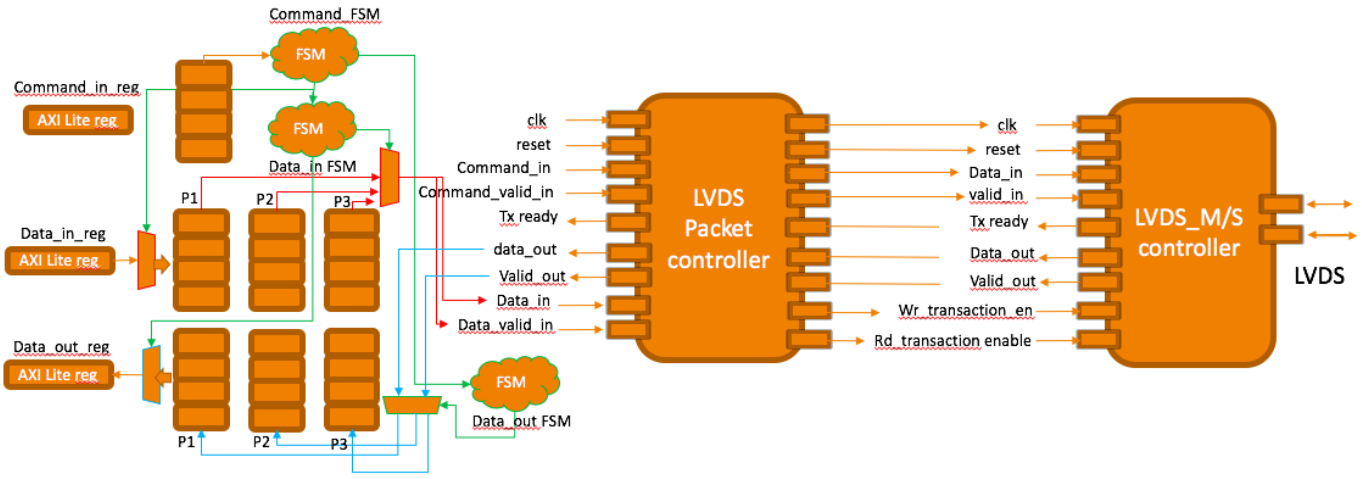

Figure 13: Scheduler

The code for scheduler as well as testbench can be found Github link. For testbench write transaction was performed with back to back commands flowing into the command register on Scheduler Master with high priority command latched after lower priority command. The output shows that scheduler first schedules high priority command and then lower priority command.

2.2 Hardware test setup on AXIOM-Beta

Most of the time was spend in this part. Specially learning and understanding of AXI bus communication system. For final testing a piece of software running on ZYNQ PS will be demanding a service from the designed scheduler.

Hence, for step by step learning and testing of the system the following steps were taken:-

- First, understand basic AXI read write transaction (PS-PL) via apertus remote setup.
- Second, designing custom AXI slave peripheral so as to instantiate my IP into the top level vivado block design.

Steps to Flash FPGA bit files on both ZYNQ and MACHXO2 are described in details here.
here.

3 PHASE 3

This section mainly deals with hardware testing on AXIOM Beta. the tests were conducted in two phases: First, loop back tests were conducted on ZYNQ-SOC i.e the main board. Both Master as well as Slave packet layers were implemented on ZYNQ PL to functionally verify the system. Second, The Slave module was flashed onto external MACHXO2 FPGA, Master module was flashed onto ZYNQ and the data link was established via LVDS channel.

3.1 Loopback tests

The steps mentioned here were followed to load bit stream to ZYNQ PL and run specific memory write/read operations in order to access the ports of Master and Slave modules. The results show that the software running on ZYNQ-PS is able to perform both read as well as write transactions to virtual slave address.

3.2 Tests with MACHO2 as Slave device

The steps mentioned here were followed to load bit stream onto both ZYNQ-PL as well as MACHXO2. Both read and write operations were conducted as mentioned in above subsections. The AXI transaction halted during both the operations.THERE could be multiple reasons for the issue:-

- Incorrect LVDS port assignment and IO buffer instantiatiion.
- Issues with Physical LVDS connection between MACHXO2 and ZYNQ PL.
- Incorrect bitstream flashing technique for MACHXO2.

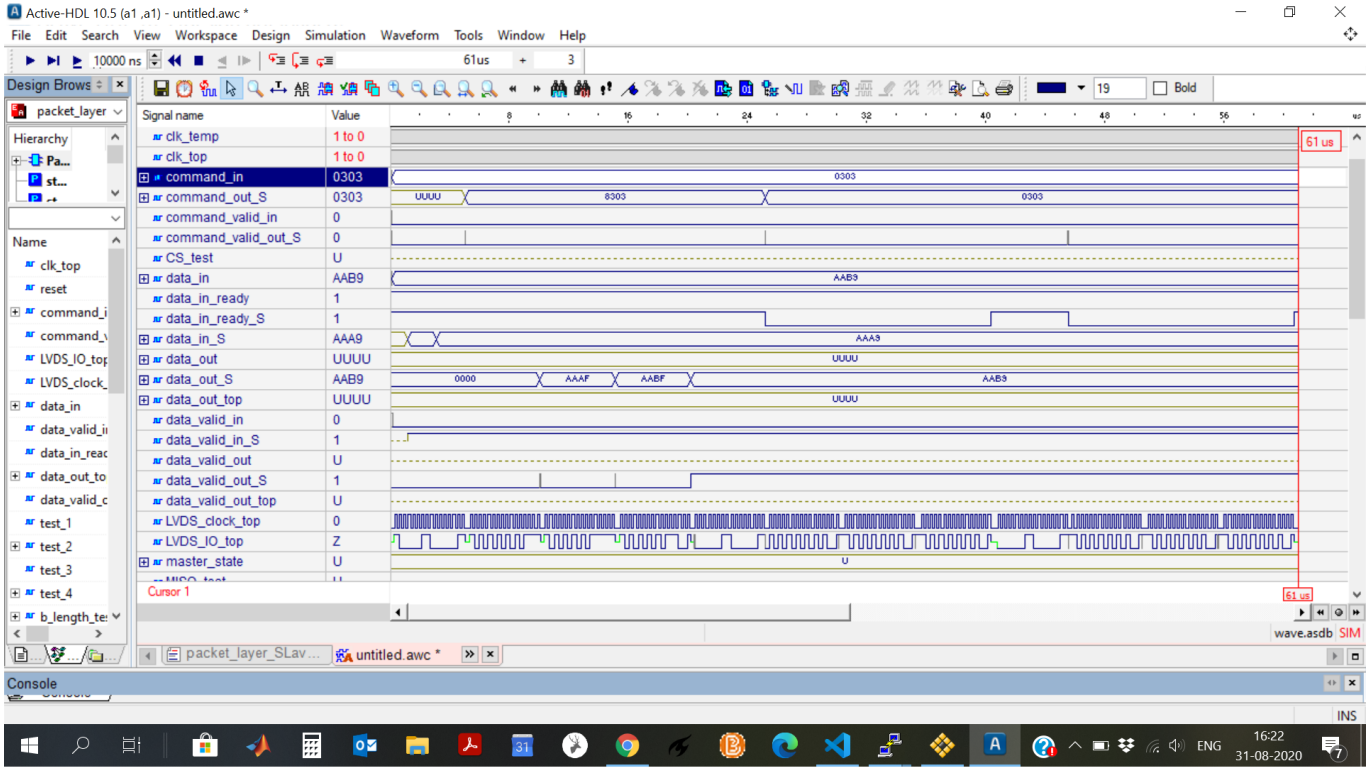

Figure 14: Scheduler test

Multiple tests were conducted during Phase 3 of the project in order to find the solution to the problem but unfortunately the bug was not fixed. This issue will be taken up after the GSOC 2020 submissions.

3.3 Conclusion and Future work

Following major contributions were made during the coding period:-

- Bidirectional link design following Master-Slave methodology that is able to perform half duplex communication.
- Packet protocol design that utilises bidirectional physical layer to perform read and write ttransactions without possibilities of any cross talk.
- Scheduler design that is able to schedule commands and coresspoding data tansactions based on priority of the service demanded by the software.
- Implementation of Packet protocol in ZYNQ AXI environment in order to easily control the bidirectional protocol packet transactions via software (Linux) running on ZYNQ=PL.

Following work still need to be done :-

- Perform hardware verification with MACHXO2 as Slave device with both read/write burst transactions controlled via Linux running on ZYNQ-PS.
- The currect protocol is dependent on Clock controlled my Master Module. In future the protocol needs to be upgraded in order to use free running clock between master and slave modules.

In conclusion the project can be incorporated with other projects that require by directional half duplex communication between different modules specially in AXIOM Beta for using MACHXO2 as IO extension device for ZYNQ.