

# Evaluation-1 Report Google summer of code 2020

Apoorva Arora

July 2, 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 bidirectional 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 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). *Under development*
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.

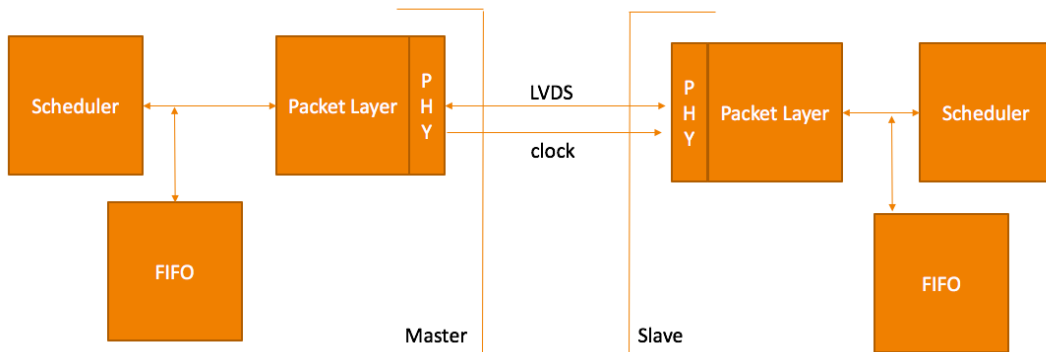


Figure 1: Top level system architecture

## 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](#)

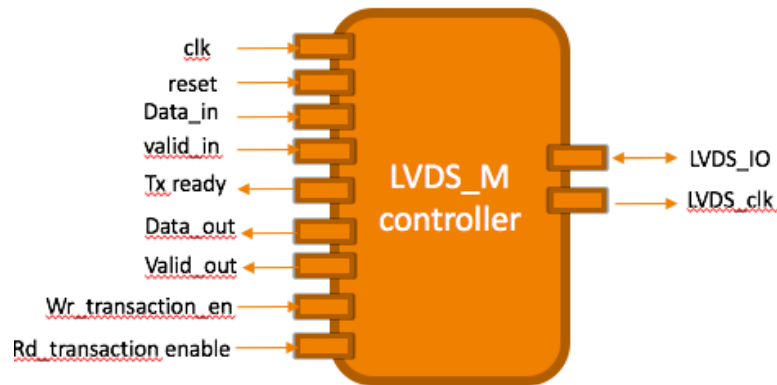


Figure 2: Master LVDS PHY SERDES

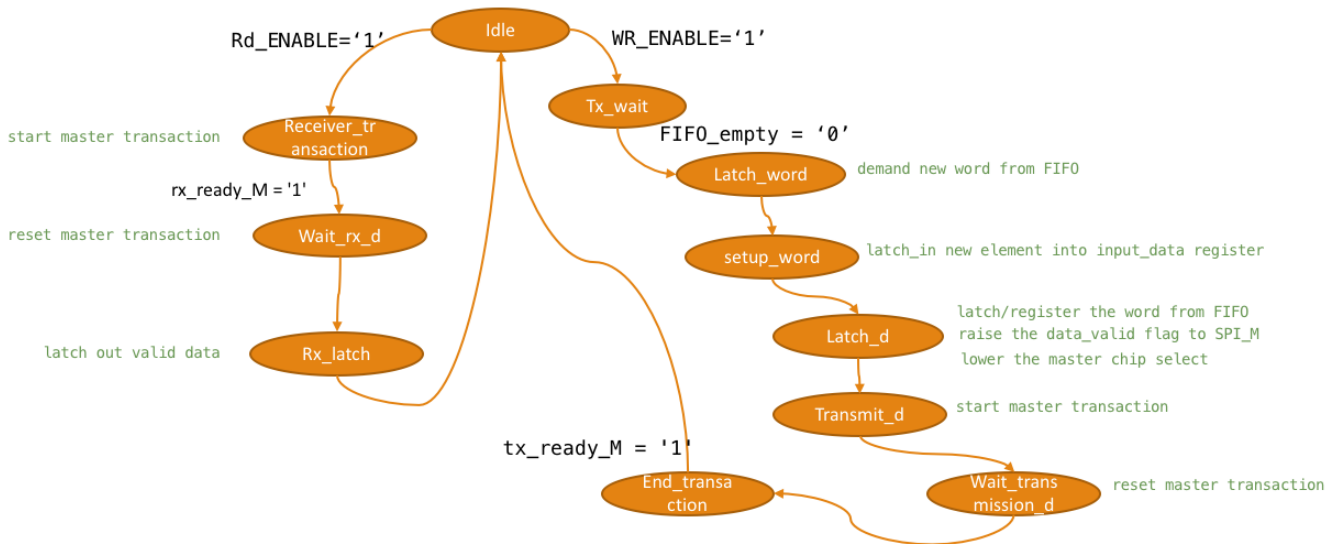


Figure 3: Master LVDS PHY SERDES-FSM

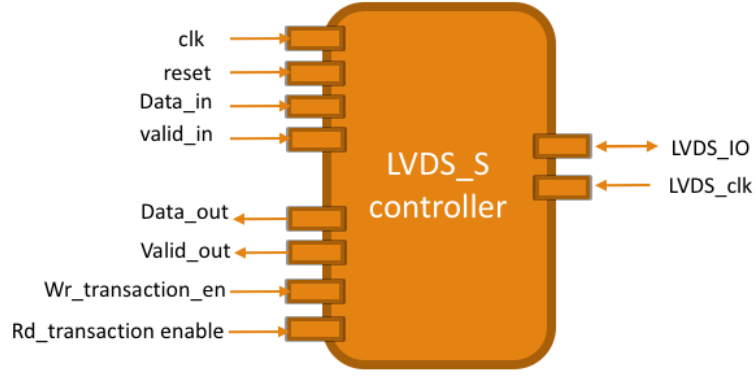


Figure 4: Slave LVDS PHY SERDES

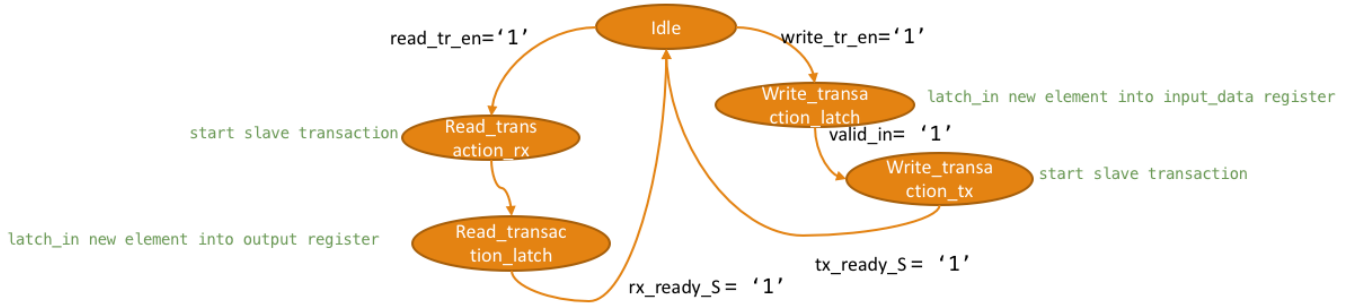


Figure 5: Slave LVDS PHY SERDES-FSM

## 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](#)

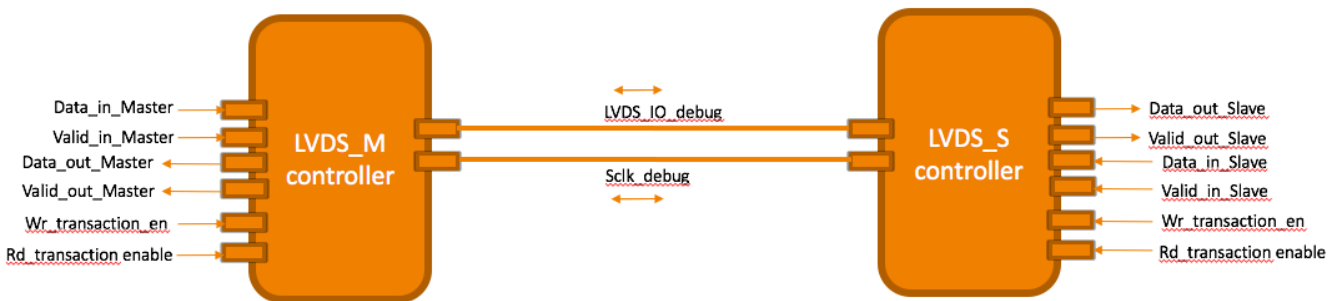


Figure 6: PHY testbench

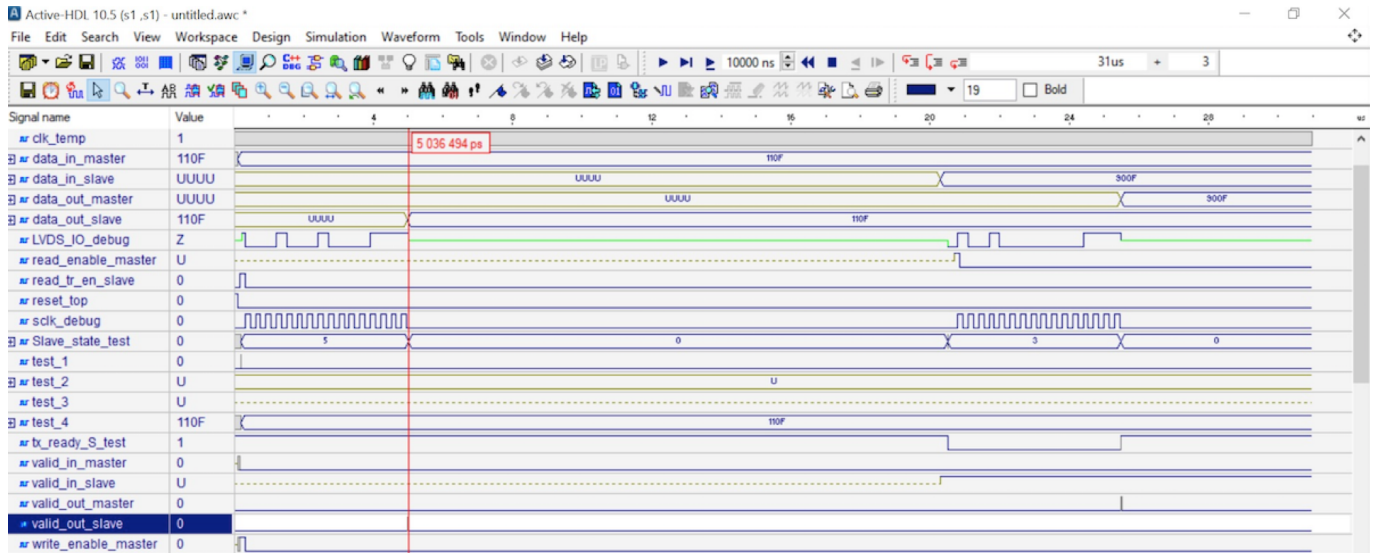


Figure 7: PHY testbench simulations screen shot

### 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](#)

#### Read/write command

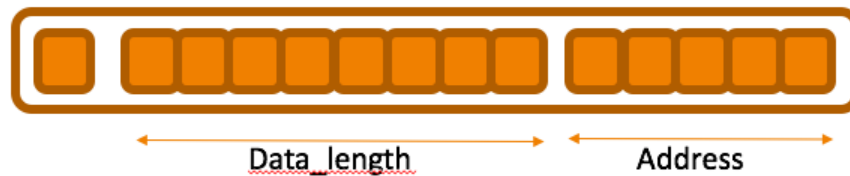


Figure 8: command packet

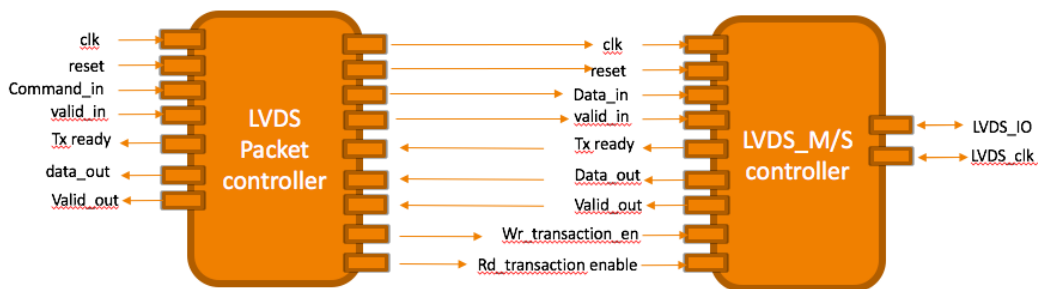


Figure 9: packet layer

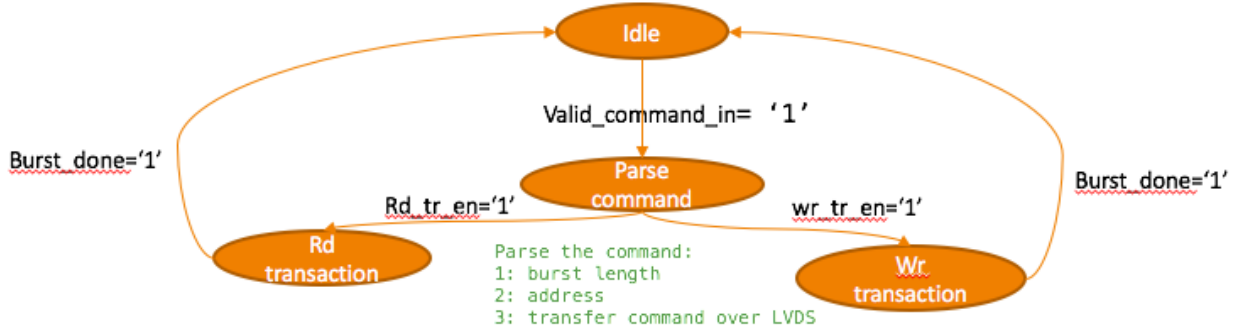


Figure 10: packet layer FSM

### 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.

## 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.

## 5 Conclusion

The simulations demonstrate that the mentioned design is able to access any slave peripheral so as to generate and receive data from it.

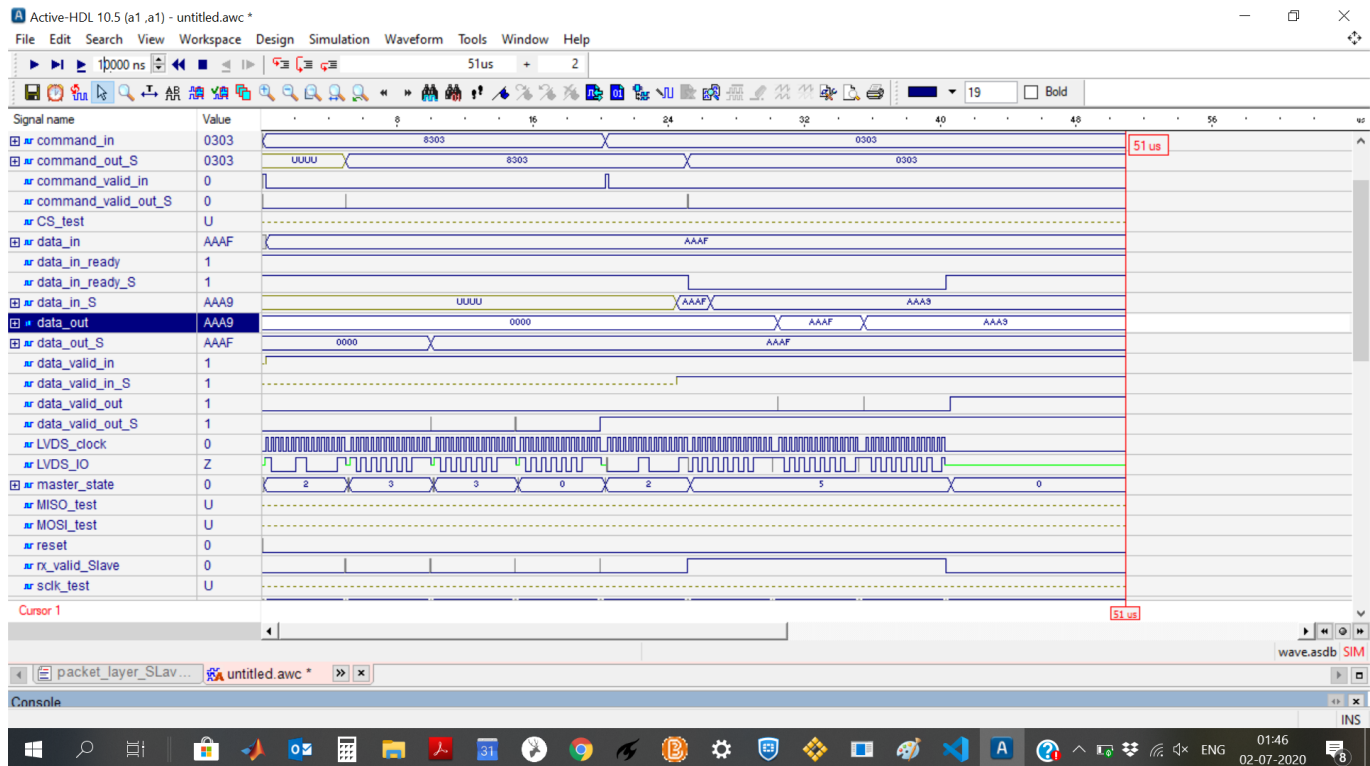


Figure 11: Packet layer read write simulations

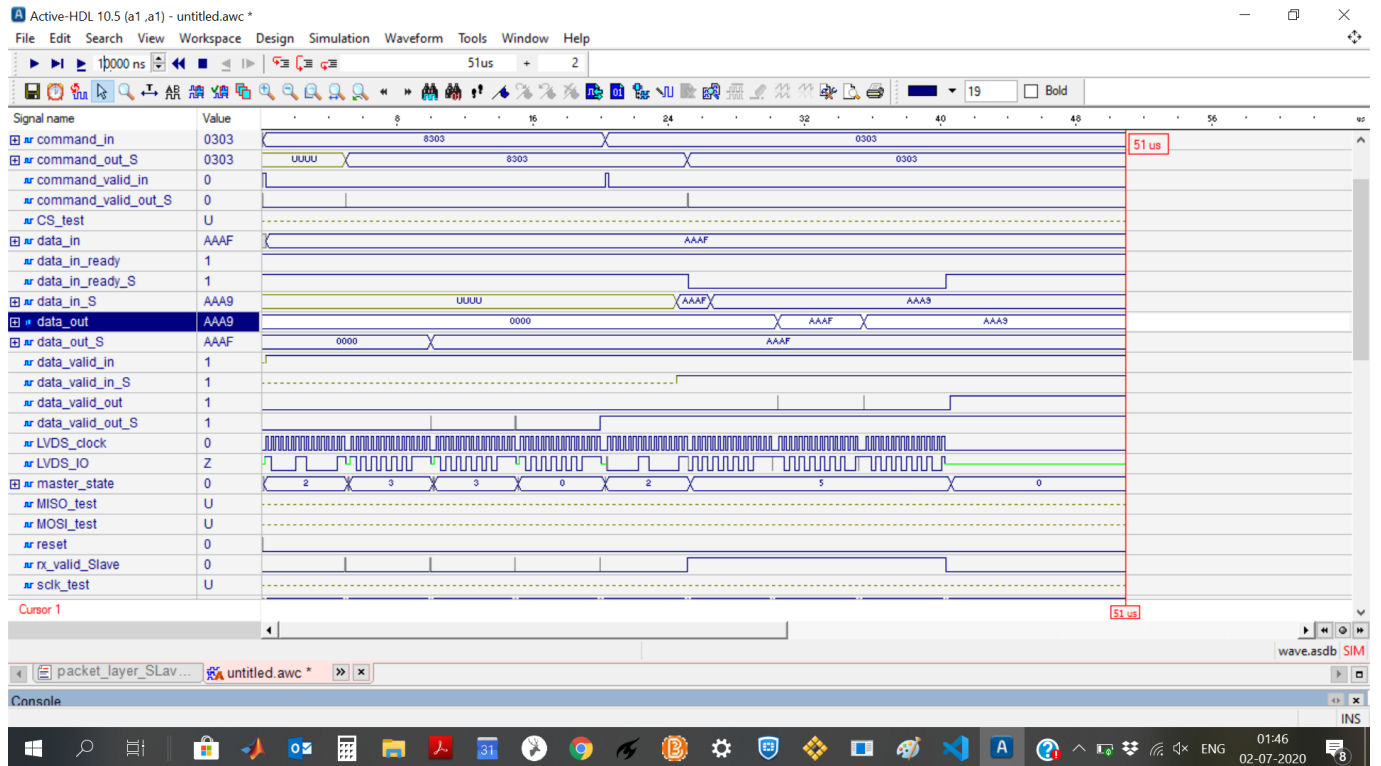


Figure 12: Packet layer SPI(M) write simulations