Faculty of Engineering
Cairo University

SPI Project

Submitted to

DR. Ihab Talkhan

Submitted by

Hussien Mustafa Hussien , sec:1 , BN : 28
Ayman Mohamed Reda , sec:1 BN : 19
Mina Emad Fakhry , sec:2 BN : 29
Mahmoud El-Sayed Mahmoud Raoff, sec:2 BN : 17

Master SPI

- The design of the master is based on 6 states (reset , idle , load, transact , unload , stslave).
- In reset state , we initialize data_out , miso_d(reg of 8 bits which the input of slave is put at index 7 of this reg) , mosi_d (reg of 8 bits which holds the data_in in load state) and count(reg indicates 8 clk cycles) with zero.
- In idle state , we initialize data_out (recent) with data_out (previous) and all of mosi_d , miso_d and count with zero.
- In load state ,we initialize data_out (recent) with data_out (previous) miso_d with zero , mosi_d with data_in and count with size(which is 8 as it's required to transfer 8 bit).
- State transact which indicates the shifting is working now.
- In state unload , we assign data_out with miso_d (miso_d finally after 8 clk cycles holds the data of slave which is sent bit by bit) , cs with 1 to stop the shifting and both miso_d and mosi_d with zero.
- There is an input called start which is the same as enable.
- There is a wire called temp indicating the master is working with which slave of 3 and consequently cs is assigned to it.
- An input called cphase which indicates with cpol in test bench which mode is working .

Process:

State in beginning in first will be reset (in master) and if(start==1) the state will be stslave (which transmit cs to slaves).

If(start==0) the state will be idle (not change)

And the state change in next positive edge to load(which load data from user (master and slave)),than the state change to transact, the state is transact (which the data master will go to slave and data slave will go to master).

<u>Note when the state =stslave the start will be don't care and if you go to initial, you should reset=1.</u>

And there is assign to mosi.

After the all data move the master check if(start==1) state will be again load and repeated.

If(start==0) the state will be idle.

When data transact there four status

Cpol=0 and cphase=0 Logic low,,, Data sampled on rising edge and shifted out on the falling

Cpol=0 and cphase=1 Logic low,,, Data sampled on the falling edge and shifted out on the rising edge

Cpol=1 and cphase=1 edge   Logic high,,, Data sampled on the falling edge and shifted out on the rising edge

Cpol=1 and cphase=0 Logic high ,,,Data sampled on the rising edge and shifted out on the falling edge

Slave SPI

- The design of the slave is based on 5 states (reset , idle , load, transact , unload).
- In reset state , we initialize data_out , mosi_d(reg of 8 bits which the input of master is put at index 7 of this reg) , miso_d (reg of 8 bits which holds the data_in in load state) and count(reg indicates 8 clk cycles) with zero.
- In idle state , we initialize data_out (recent) with data_out (previous) and all of mosi_d , miso_d and count with zero.
- In load state ,we initialize data_out (recent) with data_out (previous) mosi_d with zero , miso_d with data_in and count with size(which is 8 as it's required to transfer 8 bit).
- State transact which indicates the shifting is working now.
- In state unload , we assign data_out with mosi_d (mosi_d finally after 8 clk cycles holds the data of master which is sent bit by bit).
- An input called cphase which indicates with cpol in test bench which mode is working .

Process:

If(reset==1) the state will be reset.

If(cs==0)the state will be load (which load data and count).

If(count!=0) state will be transact (which the data master will go to slave and data slave will go to master).

But if (cs==1) the move data will be stop and the opposite is true(which the master control it(slave)).

Note:

The data out for master or salve will be zeros until the state =unload.

Integration SPI

- This module is the link between the master and the three slaves.
- The inputs data_in , data_ins0 , data_ins1 and data_ins2 are the 8 bits stored in master , slave1 , slave 2 , slave 3 respectively.
- The outputs data_out , data_outs0, data_outs1 and data_outs2 are the output of master , slave1,slave2 and slave3 respectively after shifting.
- An input called start resembles the enable signal to start shifting.
- An input called cphase which indicates with cpol in test bench which mode is working.
- CS determines which slave is working with the master.

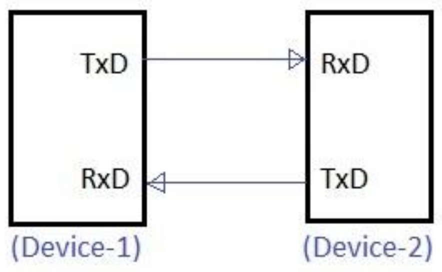
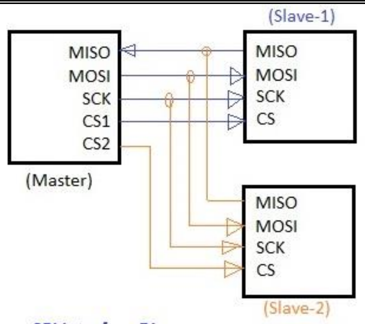
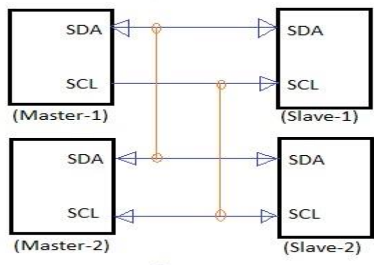
The design process of the integrate:

The module is integrated between master and 3 slaves

Notes:

Cs (output from master is 3 bits) is should be 011or 101 or 110 or 111 and every bit will move to slave example (011

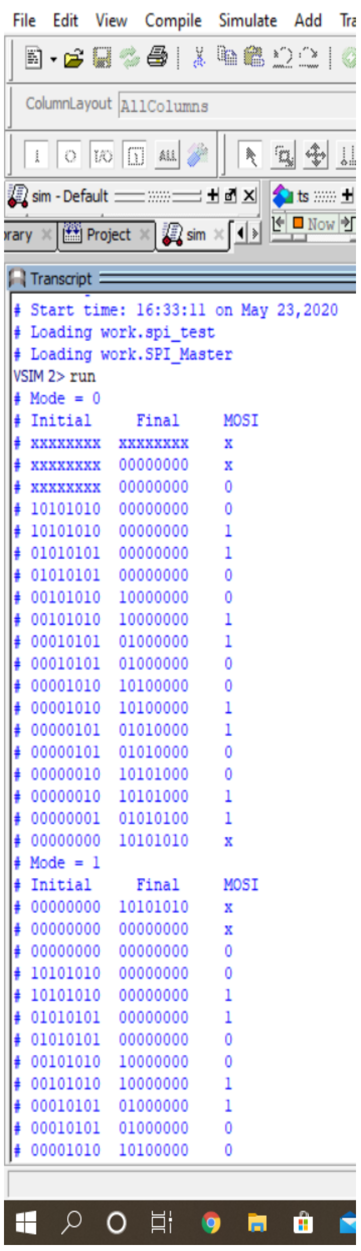
0 for slave 3 and 1 for slave 2 and 1 for slave 1)it is meaning the data will move between master and slave 3.

Features	***UART***	***SPI***	***I2C***
Full Form	Universal Asynchronous Receiver/Transmitter	Serial Peripheral Interface	Inter-Integrated Circuit
Interface Diagram	 UART Interface Diagram	 (Slave-1) MISO MOSI SCK CS1 CS2 (Master) MISO MOSI SCK CS MISO MOSI SCK CS (Slave-2) SPI Interface Diagram	 SDA SCL (Master-1) SDA SCL (Slave-1) SDA SCL (Master-2) SDA SCL (Slave-2) I2C Interface Diagram
Pin Designations	TxD: Transmit Data RxD: Receive Data	SCLK: Serial Clock MOSI: Master Output, Slave Input MISO: Master Input, Slave Output SS: Slave Select	SDA: Serial Data SCL: Serial Clock
Data rate	As this is asynchronous communication, data rate between two devices wanting to communicate should be set to equal value. Maximum data rate supported is about 230 Kbps to 460kbps.	Maximum data rate limit is not specified in SPI interface. Usually supports about 10 Mbps to 20 Mbps	I2C supports 100 kbps, 400 kbps Mbps. Some variants also supports 10 Kbps and 1 Mbps.
Distance	Lower about 50 feet	highest	Higher
Type of communication	Asynchronous	Synchronous	Synchronous
Number of masters	Not Application	One	One or more than One
Clock	No Common Clock signal is used. Both the devices will use there independent clocks.	There is one common serial clock signal between master and slave devices.	There is common clock signal be multiple masters and multiple slaves.
Hardware complexity	lesser	less	more
Protocol	For 8 bits of data one start bit and one stop bit is used.	Each company or manufacturers have got their own specific protocols to communicate with peripherals. Hence one needs to read datasheet to know read/write protocol for SPI communication to be established. For example we would like SPI communication between microcontroller and	It uses start and stop bits. It uses ACK bit for each 8 bits of data which indicates whether data has been received or not.

		EPROM. Here one need to go through read/write operational diagram in the EPROM data sheet.	
Software addressing	As this is one to one connection between two devices, addressing is not needed.	Slave select lines are used to address any particular slave connected with the master. There will be 'n' slave select lines on master device for 'n' slaves.	There will be multiple slaves and multiple masters and all masters can communicate with all the slaves. Up to 27 slave devices can be connected/ addressed in the I2C interface circuit.
Advantages	It is simple communication and most popular which is available due to UART support in almost all the devices with 9 pin connector. It is also referred as RS232 interface.	•It is simple protocol and hence so not require processing overheads. •Supports full duplex communication. •Due to separate use of CS lines, same kind of multiple chips can be used in the circuit design. •SPI uses push-pull and hence higher data rates and longer ranges are possible. •SPI uses less power compare to I2C	•Due to open collector design, limited slew rates can be achieved. •More than one masters can be used in the electronic circuit design. •Needs fewer i.e. only 2 wires for communication. •I2C addressing is simple which does not require any CS lines used in SPI and it is easy to add extra devices on the bus. •It uses open collector bus concept. Hence there is bus voltage flexibity on the interface bus. •Uses flow control.
Disadvantage	• They are suitable for communication between only two devices. • It supports fixed data rate agreed upon between devices initially before communication otherwise data will be garbled.	• As number of slave increases, number of CS lines increases, this results in hardware complexity as number of pins required will increase. • To add a device in SPI requires one to add extra CS line and changes in software for particular device addressing is concerned. •Master and slave relationship can not be changed as usually done in I2C interface. •No flow control available in SPI.	•Increases complexity of the circuit when number of slaves and masters increases. •I2C interface is half duplex. •Requires software stack to control the protocol and hence it needs some processing overheads on microcontroller/ microprocessor.

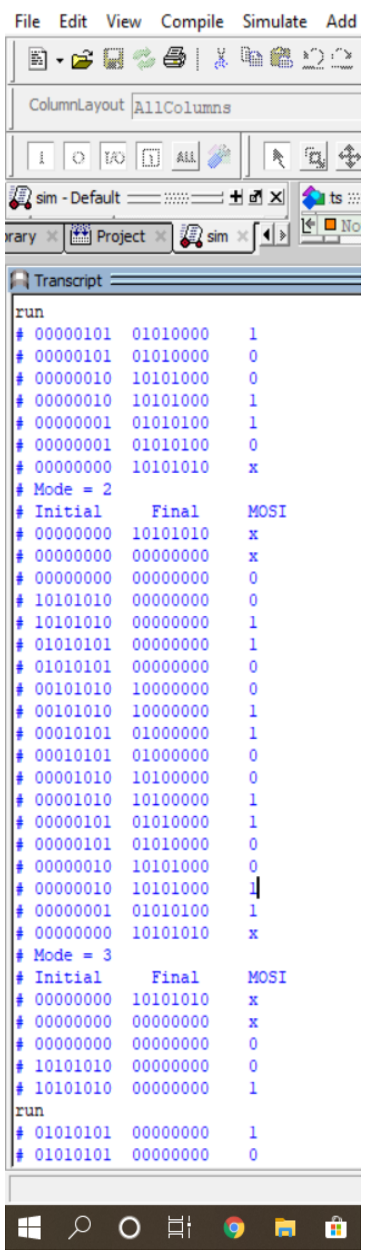
Test-Benches:

1. Master Test-Bench:



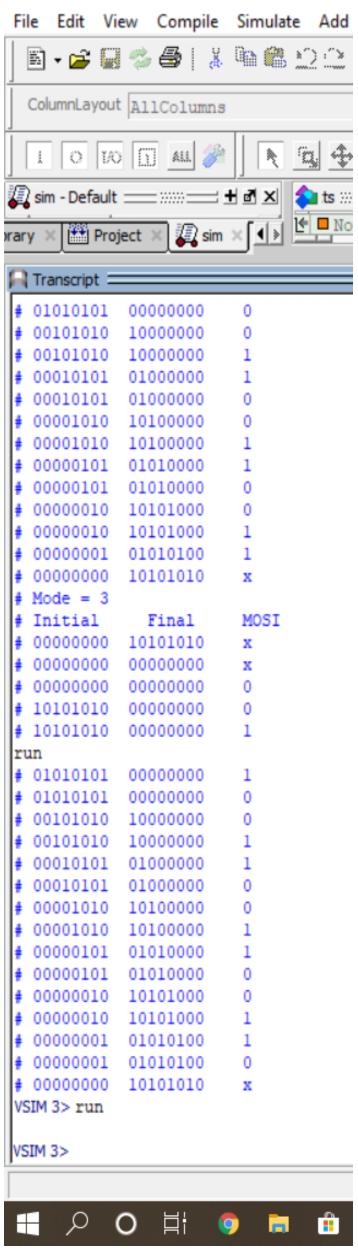
```
# Start time: 16:33:11 on May 23,2020
# Loading work.spi_test
# Loading work.SPI_Master
VSIM 2> run
# Mode = 0
# Initial     Final     MOSI
# xxxxxxxx  xxxxxxxx    x
# xxxxxxxx  00000000    x
# xxxxxxxx  00000000    0
# 10101010  00000000    0
# 10101010  00000000    1
# 01010101  00000000    1
# 01010101  00000000    0
# 00101010  10000000    0
# 00101010  10000000    1
# 00010101  01000000    1
# 00010101  01000000    0
# 00001010  10100000    0
# 00001010  10100000    1
# 00000101  01010000    1
# 00000101  01010000    0
# 00000010  10101000    0
# 00000010  10101000    1
# 00000001  01010100    1
# 00000000  10101010    x
# Mode = 1
# Initial     Final     MOSI
# 00000000  10101010    x
# 00000000  00000000    x
# 00000000  00000000    0
# 10101010  00000000    0
# 10101010  00000000    1
# 01010101  00000000    1
# 01010101  00000000    0
# 00101010  10000000    0
# 00101010  10000000    1
# 00010101  01000000    1
# 00010101  01000000    0
# 00001010  10100000    0
```

Figure 1 : master test-bench 1



```
run
# 00000101  01010000    1
# 00000101  01010000    0
# 00000010  10101000    0
# 00000010  10101000    1
# 00000001  01010100    1
# 00000001  01010100    0
# 00000000  10101010    x
# Mode = 2
# Initial     Final     MOSI
# 00000000  10101010    x
# 00000000  00000000    x
# 00000000  00000000    0
# 10101010  00000000    0
# 10101010  00000000    1
# 01010101  00000000    1
# 01010101  00000000    0
# 00101010  10000000    0
# 00101010  10000000    1
# 00010101  01000000    1
# 00010101  01000000    0
# 00001010  10100000    0
# 00001010  10100000    1
# 00000101  01010000    1
# 00000101  01010000    0
# 00000010  10101000    0
# 00000010  10101000    1
# 00000001  01010100    1
# 00000000  10101010    x
# Mode = 3
# Initial     Final     MOSI
# 00000000  10101010    x
# 00000000  00000000    x
# 00000000  00000000    0
# 10101010  00000000    0
# 10101010  00000000    1
run
# 01010101  00000000    1
# 01010101  00000000    0
```

Figure 3 : master test-bench 2



```
# 01010101  00000000    0
# 00101010  10000000    0
# 00101010  10000000    1
# 00010101  01000000    1
# 00010101  01000000    0
# 00001010  10100000    0
# 00001010  10100000    1
# 00000101  01010000    1
# 00000101  01010000    0
# 00000010  10101000    0
# 00000010  10101000    1
# 00000001  01010100    1
# 00000000  10101010    x
# Mode = 3
# Initial     Final     MOSI
# 00000000  10101010    x
# 00000000  00000000    x
# 00000000  00000000    0
# 10101010  00000000    0
# 10101010  00000000    1
run
# 01010101  00000000    1
# 01010101  00000000    0
# 00101010  10000000    0
# 00101010  10000000    1
# 00010101  01000000    1
# 00010101  01000000    0
# 00001010  10100000    0
# 00001010  10100000    1
# 00000101  01010000    1
# 00000101  01010000    0
# 00000010  10101000    0
# 00000010  10101000    1
# 00000001  01010100    1
# 00000001  01010100    0
# 00000000  10101010    x
VSIM 3> run

VSIM 3>
```

Figure 2 : master test-bench 3

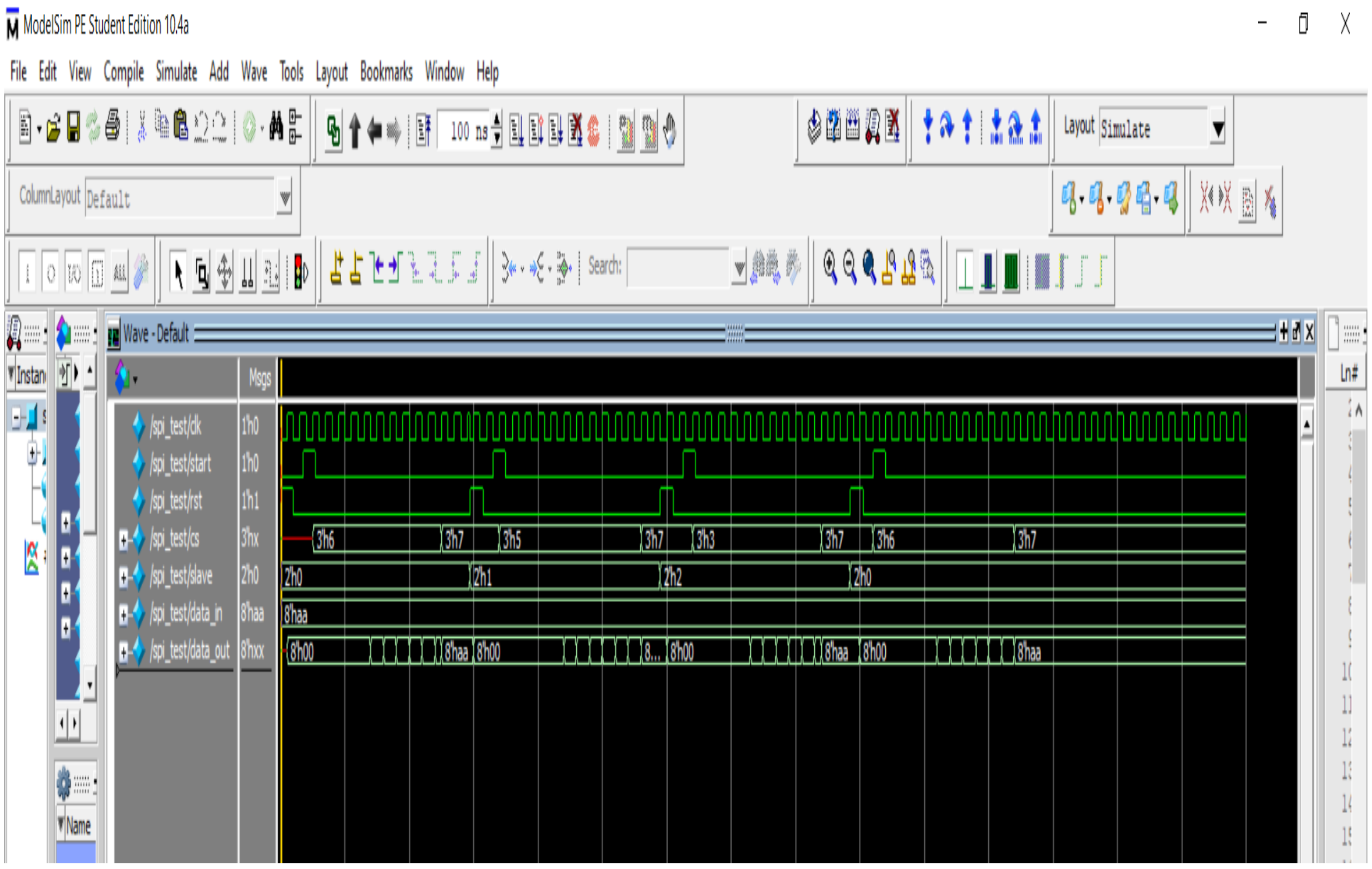
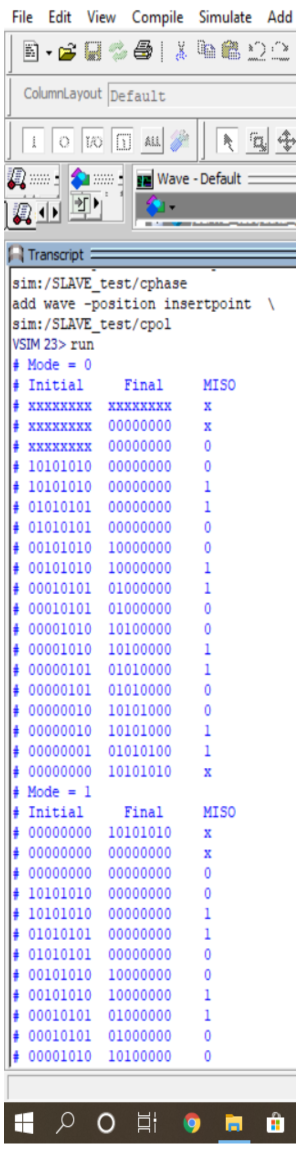


Figure 4 : master test-bench wave form

In figures 1,2,3 the left column represents the initial value given from the controller to the master ,It begins with this value. And then the slave's data gets shifted to the right column bit by bit through the miso which represents the final form of data inside the master. We display the mode before each transfer process; the mode is determined by the c-pol and c-phase .in the master test bench there is no slave to transfer the data to ;so we made the data transfer again from the master to the master by assigning mosi to miso "assign miso=mosi". In this test bench we tried the 4 modes by changing c-phase and c-pol as displayed in the first 3 figures. The mosi is also displayed it is the least significant bit of the master that will be transmitted to the slave.

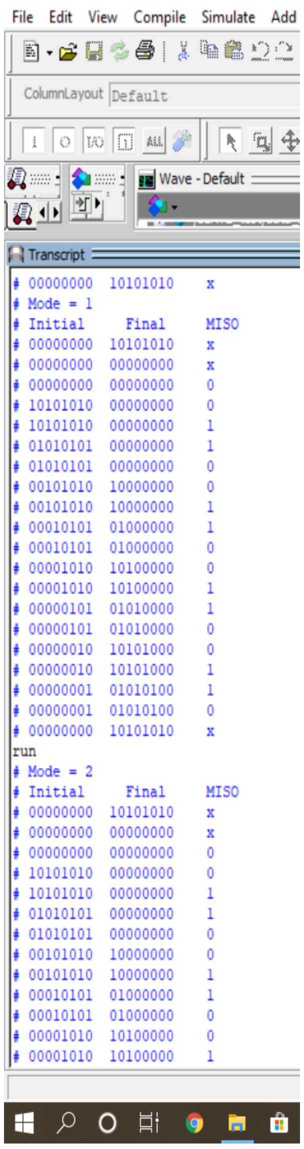
In the wave form we display first the clock initialized before each transfer process according to c-pol. Then the start control witch moves the master from the idle state to the load state to begin the transfer process. Then the rst control that moves the master to the reset state that resets all the data to its initial state. Then the cs that represents the selected slave if there was one. The cs is low driven so "6->110" means that slave 0 is selected and "7->111" means no slave is selected at that time, and so on. Then the slave which is an input to the master from the controller, it describes the address of the wanted slave. Then the data-in of the master witch is the initial value of the data inside the master inserted to it from the controller. Finally, the data-out is the final value inside the master after shifting in the data of the selected slave.

2. Slave Test-Bench:



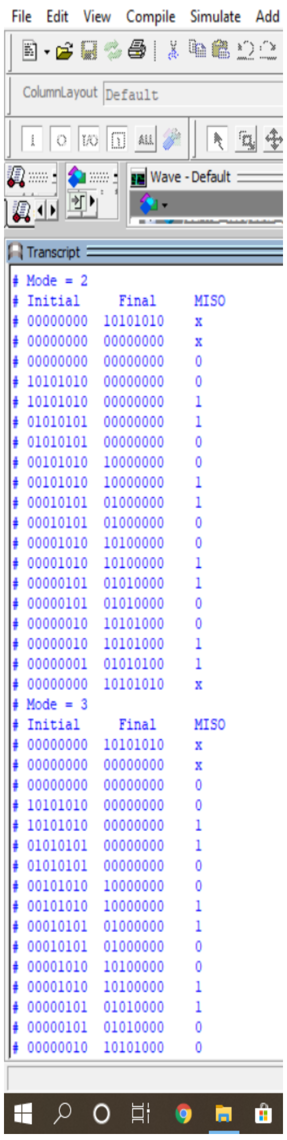
```
sim:/SLAVE_test/cphase
add wave -position insertpoint  \
sim:/SLAVE_test/cpol
VSIM 23> run
# Mode = 0
# Initial     Final     MISO
# xxxxxxxx  xxxxxxxx    x
# xxxxxxxx  00000000    x
# xxxxxxxx  00000000    0
# 10101010  00000000    0
# 10101010  00000000    1
# 01010101  00000000    1
# 01010101  00000000    0
# 00101010  10000000    0
# 00101010  10000000    1
# 00010101  01000000    1
# 00010101  01000000    0
# 00001010  10100000    0
# 00001010  10100000    1
# 00000101  01010000    1
# 00000101  01010000    0
# 00000010  10101000    0
# 00000010  10101000    1
# 00000001  01010100    1
# 00000000  10101010    x
# Mode = 1
# Initial     Final     MISO
# 00000000  10101010    x
# 00000000  00000000    x
# 00000000  00000000    0
# 10101010  00000000    0
# 10101010  00000000    1
# 01010101  00000000    1
# 01010101  00000000    0
# 00101010  10000000    0
# 00101010  10000000    1
# 00010101  01000000    1
# 00010101  01000000    0
# 00001010  10100000    0
```

Figure 5: slave test-bench 1



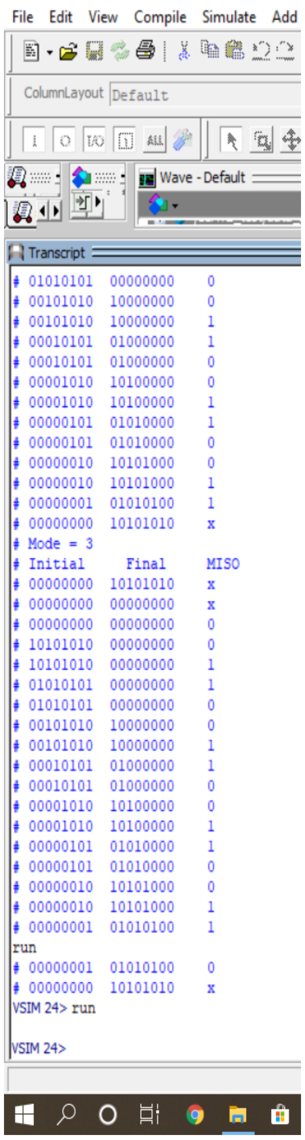
```
# 00000000  10101010    x
# Mode = 1
# Initial     Final     MISO
# 00000000  10101010    x
# 00000000  00000000    x
# 00000000  00000000    0
# 10101010  00000000    0
# 10101010  00000000    1
# 01010101  00000000    1
# 01010101  00000000    0
# 00101010  10000000    0
# 00101010  10000000    1
# 00010101  01000000    1
# 00010101  01000000    0
# 00001010  10100000    0
# 00001010  10100000    1
# 00000101  01010000    1
# 00000101  01010000    0
# 00000010  10101000    0
# 00000010  10101000    1
# 00000001  01010100    1
# 00000001  01010100    0
# 00000000  10101010    x
run
# Mode = 2
# Initial     Final     MISO
# 00000000  10101010    x
# 00000000  00000000    x
# 00000000  00000000    0
# 10101010  00000000    0
# 10101010  00000000    1
# 01010101  00000000    1
# 01010101  00000000    0
# 00101010  10000000    0
# 00101010  10000000    1
# 00010101  01000000    1
# 00010101  01000000    0
# 00001010  10100000    0
# 00001010  10100000    1
```

Figure 6: slave test-bench 2



```
# Mode = 2
# Initial     Final     MISO
# 00000000  10101010    x
# 00000000  00000000    x
# 00000000  00000000    0
# 10101010  00000000    0
# 10101010  00000000    1
# 01010101  00000000    1
# 01010101  00000000    0
# 00101010  10000000    0
# 00101010  10000000    1
# 00010101  01000000    1
# 00010101  01000000    0
# 00001010  10100000    0
# 00001010  10100000    1
# 00000101  01010000    1
# 00000101  01010000    0
# 00000010  10101000    0
# 00000010  10101000    1
# 00000001  01010100    1
# 00000000  10101010    x
# Mode = 3
# Initial     Final     MISO
# 00000000  10101010    x
# 00000000  00000000    x
# 00000000  00000000    0
# 10101010  00000000    0
# 10101010  00000000    1
# 01010101  00000000    1
# 01010101  00000000    0
# 00101010  10000000    0
# 00101010  10000000    1
# 00010101  01000000    1
# 00010101  01000000    0
# 00001010  10100000    0
# 00001010  10100000    1
# 00000101  01010000    1
# 00000101  01010000    0
# 00000010  10101000    0
```

Figure 7: slave test-bench 3



```
# 01010101  00000000    0
# 00101010  10000000    0
# 00101010  10000000    1
# 00010101  01000000    1
# 00010101  01000000    0
# 00001010  10100000    0
# 00001010  10100000    1
# 00000101  01010000    1
# 00000101  01010000    0
# 00000010  10101000    0
# 00000010  10101000    1
# 00000001  01010100    1
# 00000000  10101010    x
# Mode = 3
# Initial     Final     MISO
# 00000000  10101010    x
# 00000000  00000000    x
# 00000000  00000000    0
# 10101010  00000000    0
# 10101010  00000000    1
# 01010101  00000000    1
# 01010101  00000000    0
# 00101010  10000000    0
# 00101010  10000000    1
# 00010101  01000000    1
# 00010101  01000000    0
# 00001010  10100000    0
# 00001010  10100000    1
# 00000101  01010000    1
# 00000101  01010000    0
# 00000010  10101000    0
# 00000010  10101000    1
# 00000001  01010100    1
run
# 00000001  01010100    0
# 00000000  10101010    x
VSIM 24> run

VSIM 24>
```

Figure 8: slave test-bench 4

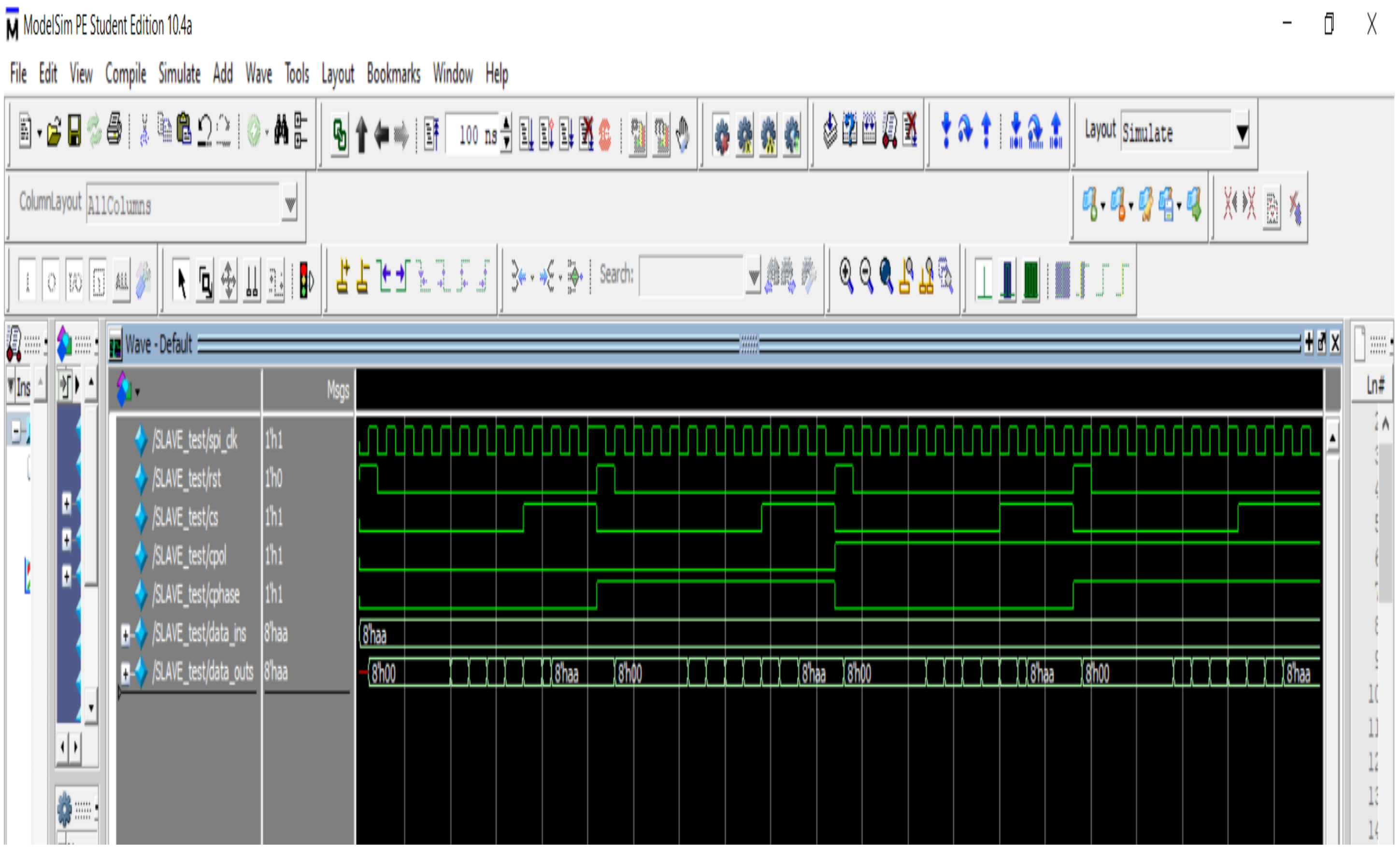
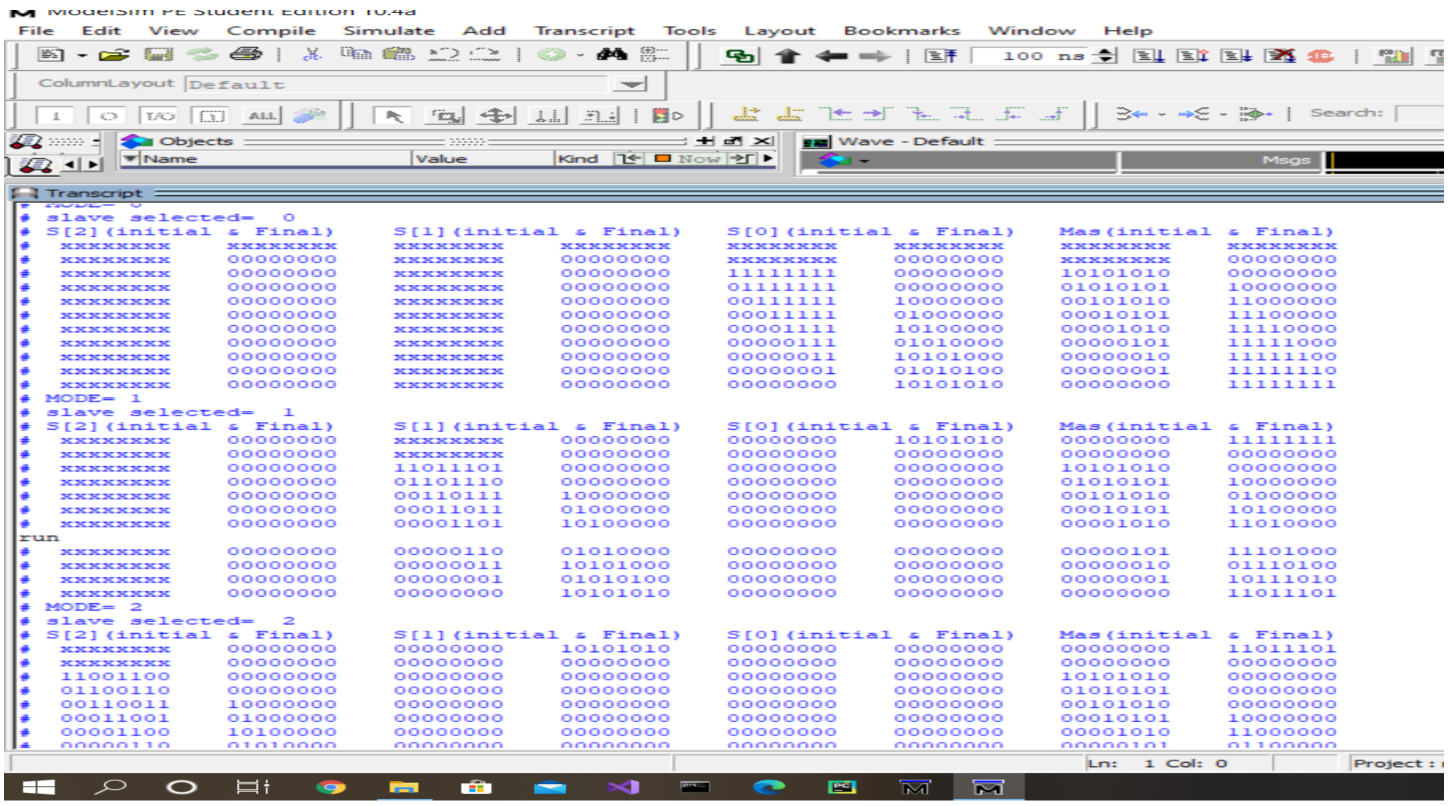


Figure9: slave test-bench wave form

As in the master test bench, the initial data value is given to the slave from the controller and displayed in the left column. The data of the master is shifted to the right column bit by bit through the mosi. Here we also display the mode decided by the c-pol and c-phase. Since there is no master also in the slave test bench so we transferred the data from the slave to itself by assigning the miso into the mosi “assign mosi=miso”.here we also tried the 4 modes by changing c-phase and c-pol as displayed in figures 5,6,7,8. The miso is also displayed it is the least significant bit of the slave that will be transmitted to the master.

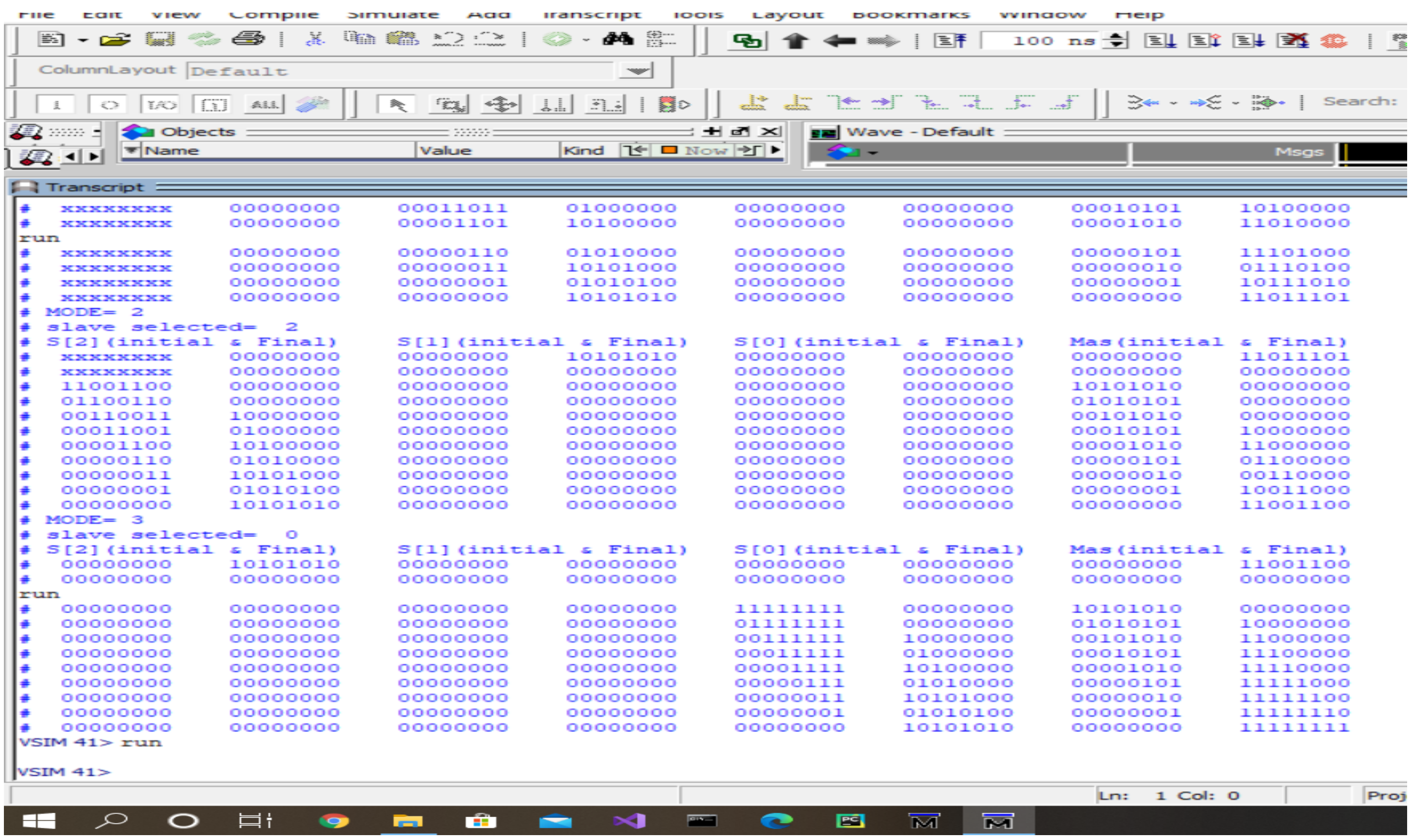
In the wave form, here we display first the spi_clk witch is the same clock as the aster and is inserted to the slave from the master. Then the rst control that resets the slave’s data to its initial state.then the cs control that controls if this slave is selected and it’s low active. Then the c-pol and c-phase they determine the mode and when the data is sampled and transmitted. The data_ins is the initial value of the data inside the slave assigned by the controller. And finally data_outs which is the final data value inside the slave during and after the master data is transmitted to the slave through the mosi.

3. Integration Test-Bench:


ModelSim PE Student Edition 10.4a
File Edit View Compile Simulate Add Transcript Tools Layout Bookmarks Window Help
100 ns
ColumnLayout Default
Search:
Objects
Name Value Kind Now
Wave - Default
Msgs
Transcript
Ln: 1 Col: 0
Project :

```
# MODE= 0
# slave selected=  0
# S[2](initial & Final)     S[1](initial & Final)     S[0](initial & Final)     Mas(initial & Final)
#  xxxxxxxx   xxxxxxxx      xxxxxxxx   xxxxxxxx       xxxxxxxx   xxxxxxxx       xxxxxxxx   xxxxxxxx
#  xxxxxxxx   00000000      xxxxxxxx   00000000       xxxxxxxx   00000000       xxxxxxxx   00000000
#  xxxxxxxx   00000000      xxxxxxxx   00000000       11111111   00000000       10101010   00000000
#  xxxxxxxx   00000000      xxxxxxxx   00000000       01111111   00000000       01010101   10000000
#  xxxxxxxx   00000000      xxxxxxxx   00000000       00111111   10000000       00101010   11000000
#  xxxxxxxx   00000000      xxxxxxxx   00000000       00011111   01000000       00010101   11100000
#  xxxxxxxx   00000000      xxxxxxxx   00000000       00001111   10100000       00001010   11110000
#  xxxxxxxx   00000000      xxxxxxxx   00000000       00000111   01010000       00000101   11111000
#  xxxxxxxx   00000000      xxxxxxxx   00000000       00000011   10101000       00000010   11111100
#  xxxxxxxx   00000000      xxxxxxxx   00000000       00000001   01010100       00000001   11111110
#  xxxxxxxx   00000000      xxxxxxxx   00000000       00000000   10101010       00000000   11111111
# MODE= 1
# slave selected=  1
# S[2](initial & Final)     S[1](initial & Final)     S[0](initial & Final)     Mas(initial & Final)
#  xxxxxxxx   00000000      xxxxxxxx   00000000       00000000   10101010       00000000   11111111
#  xxxxxxxx   00000000      xxxxxxxx   00000000       00000000   00000000       00000000   00000000
#  xxxxxxxx   00000000      11011101   00000000       00000000   00000000       10101010   00000000
#  xxxxxxxx   00000000      01101110   00000000       00000000   00000000       01010101   10000000
#  xxxxxxxx   00000000      00110111   10000000       00000000   00000000       00101010   01000000
#  xxxxxxxx   00000000      00011011   01000000       00000000   00000000       00010101   10100000
#  xxxxxxxx   00000000      00001101   10100000       00000000   00000000       00001010   11010000
run
#  xxxxxxxx   00000000      00000110   01010000       00000000   00000000       00000101   11101000
#  xxxxxxxx   00000000      00000011   10101000       00000000   00000000       00000010   01110100
#  xxxxxxxx   00000000      00000001   01010100       00000000   00000000       00000001   10111010
#  xxxxxxxx   00000000      00000000   10101010       00000000   00000000       00000000   11011101
# MODE= 2
# slave selected=  2
# S[2](initial & Final)     S[1](initial & Final)     S[0](initial & Final)     Mas(initial & Final)
#  xxxxxxxx   00000000      00000000   10101010       00000000   00000000       00000000   11011101
#  xxxxxxxx   00000000      00000000   00000000       00000000   00000000       00000000   00000000
#  11001100   00000000      00000000   00000000       00000000   00000000       10101010   00000000
#  01100110   00000000      00000000   00000000       00000000   00000000       01010101   00000000
#  00110011   10000000      00000000   00000000       00000000   00000000       00101010   00000000
#  00011001   01000000      00000000   00000000       00000000   00000000       00010101   10000000
#  00001100   10100000      00000000   00000000       00000000   00000000       00001010   11000000
#  00000110   01010000      00000000   00000000       00000000   00000000       00000101   01100000
```

Figure 10: integration test-bench 1



```
#  xxxxxxxx   00000000      00011011   01000000       00000000   00000000       00010101   10100000
#  xxxxxxxx   00000000      00001101   10100000       00000000   00000000       00001010   11010000
run
#  xxxxxxxx   00000000      00000110   01010000       00000000   00000000       00000101   11101000
#  xxxxxxxx   00000000      00000011   10101000       00000000   00000000       00000010   01110100
#  xxxxxxxx   00000000      00000001   01010100       00000000   00000000       00000001   10111010
#  xxxxxxxx   00000000      00000000   10101010       00000000   00000000       00000000   11011101
# MODE= 2
# slave selected=  2
# S[2](initial & Final)     S[1](initial & Final)     S[0](initial & Final)     Mas(initial & Final)
#  xxxxxxxx   00000000      00000000   10101010       00000000   00000000       00000000   11011101
#  xxxxxxxx   00000000      00000000   00000000       00000000   00000000       00000000   00000000
#  11001100   00000000      00000000   00000000       00000000   00000000       10101010   00000000
#  01100110   00000000      00000000   00000000       00000000   00000000       01010101   00000000
#  00110011   10000000      00000000   00000000       00000000   00000000       00101010   00000000
#  00011001   01000000      00000000   00000000       00000000   00000000       00010101   10000000
#  00001100   10100000      00000000   00000000       00000000   00000000       00001010   11000000
#  00000110   01010000      00000000   00000000       00000000   00000000       00000101   01100000
#  00000011   10101000      00000000   00000000       00000000   00000000       00000010   00110000
#  00000001   01010100      00000000   00000000       00000000   00000000       00000001   10011000
#  00000000   10101010      00000000   00000000       00000000   00000000       00000000   11001100
# MODE= 3
# slave selected=  0
# S[2](initial & Final)     S[1](initial & Final)     S[0](initial & Final)     Mas(initial & Final)
#  00000000   10101010      00000000   00000000       00000000   00000000       00000000   11001100
#  00000000   00000000      00000000   00000000       00000000   00000000       00000000   00000000
run
#  00000000   00000000      00000000   00000000       11111111   00000000       10101010   00000000
#  00000000   00000000      00000000   00000000       01111111   00000000       01010101   10000000
#  00000000   00000000      00000000   00000000       00111111   10000000       00101010   11000000
#  00000000   00000000      00000000   00000000       00011111   01000000       00010101   11100000
#  00000000   00000000      00000000   00000000       00001111   10100000       00001010   11110000
#  00000000   00000000      00000000   00000000       00000111   01010000       00000101   11111000
#  00000000   00000000      00000000   00000000       00000011   10101000       00000010   11111100
#  00000000   00000000      00000000   00000000       00000001   01010100       00000001   11111110
#  00000000   00000000      00000000   00000000       00000000   10101010       00000000   11111111
VSIM 41> run

VSIM 41>
```

Figure 11: integration test-bench 2

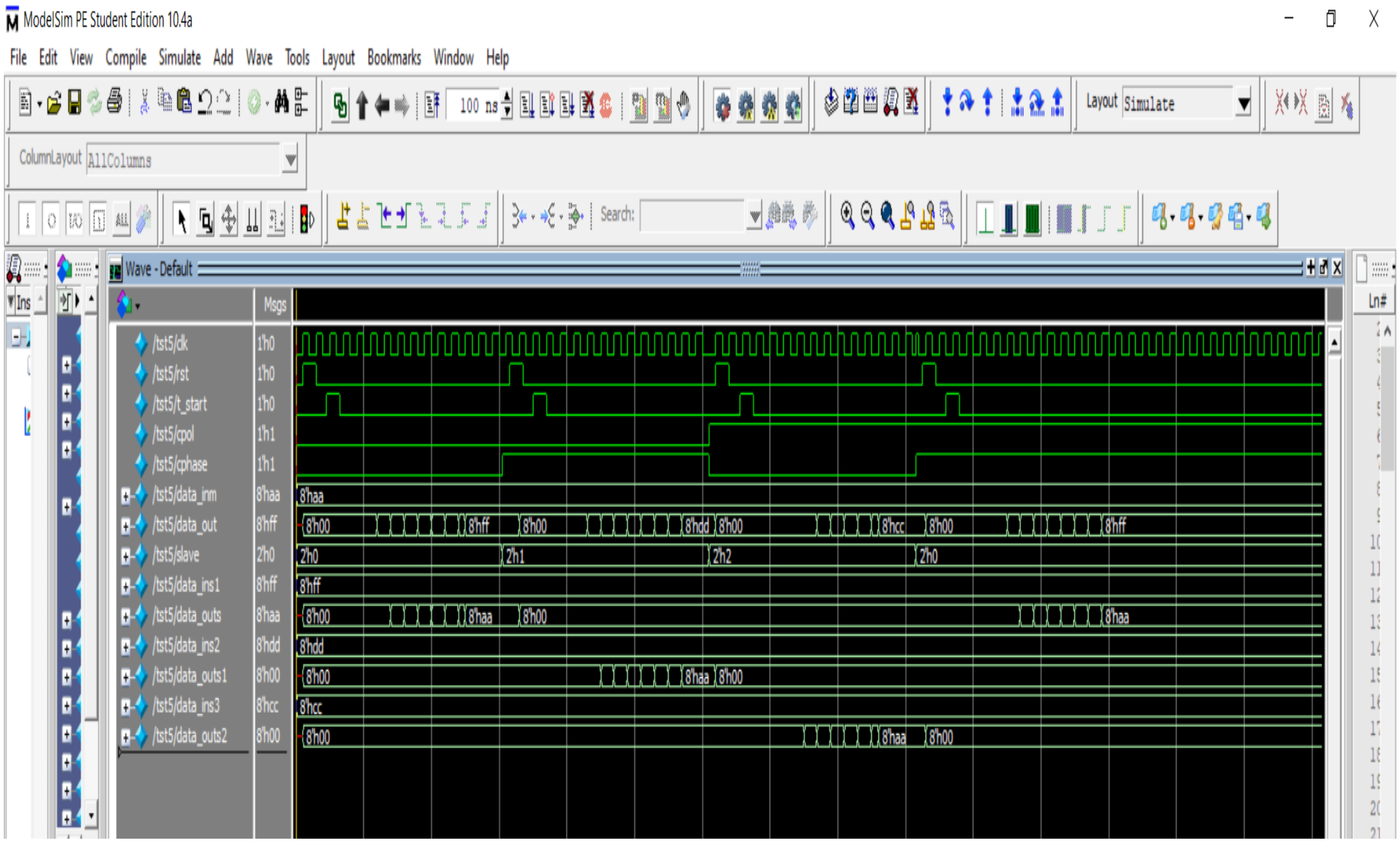


Figure 12: integration test-bench wave form

Here in the integration test bench the data of the master and the three slaves is displayed in the table in figures 10, 11. First we display the mode determined by the c-pol and c-phase. Then we display the selected slave that will send and receive data with the master. In the test bench we examined the four modes and the three slaves by changing the c-pol, c-phase and the slave address. The initial data in the master is transferred bit by bit to the final data of the selected slave, and the initial data of the slave is also transferred bit by bit to the master final data. Between each transfer process we reset the system.

In the wave representation, here we first display the common clock between the master and all the slaves. Then the rst control that returns the system to the reset state and resets its data to the initial state. Then the start control that marks the beginning of the transfer process as it moves the system to the load state. Then the c-pol and c-phase that determines the mode and the edge of the clock at which the data is sampled and transferred. Then the data-in of the master that is inserted to it from the controller and the data-out of the master that represents the data inside the master during and after the transfer process. Then the slave address that determines the selscted slave that will perform the transfer process with the master. Then the input data of the three slaves which is inserted from the controller each input data of every slave is followed by the output data of the same slave.

Work Load Distrebution :

Hussien Mustafa Hussien: Test-Benches

Ayman Mohamed Reda: Integration

Mina Emad Fakhry: Slave

Mahmoud El-Sayed Mahmoud Raoff: Master