Cairo University

First Year Computer
Engineering Department

Serial Peripheral Interface

Name	Sec	BN
Donia AbdElfatah	1	30
We'am Bassem hosni	2	36
Yousef Ahmed Anwar	2	38
Yousef Atef Tawfek	2	40

Abstract

Nowadays, Serial Communication has become very suitable with regard to cost and synchronization. Serial Peripheral Interface (SPI) is a serial communication protocol that operates in full duplex master-slave-based interface. In this report we describe the implementation and design process of Serial Peripheral Interface using VERILOG and show the test cases, the reason behind choosing them and their results, in addition to the simulation results. Through this Report, we compare between the SPI, I2C and UART, describing their interfaces, clocks, data frames, start and end conditions, etc.

Introduction

There are a lot of communication protocols for both short and long-distance communication purposes such as USB, ETHERNET, PCI-EXPRESS and SATA, which are used for long distance communication and I2C and SPI that are used for short distance communications. SPI is a serial interface protocol; it has high transmission speed and simple to use, compared to other protocols. The interface was developed by **Motorola**[1] in the mid-1980s and has become a **de facto standard[2]. [1)]**

SPI is basically a relatively simple synchronous serial interface for connecting external devices that has low speed using minimal number of wires. It can be used to communicate with another microcontroller with an SPI interface or with a serial peripheral device.

The SPI is a synchronous serial interface in which 8-bit data can be shifted-in and/or out one bit at a time, i.e., a synchronous clock shifts serial data into and out of the microcontrollers in blocks of 8 bits. Every SPI system consists of one master and one or more slaves, where a master begins the communication by enabling the SS line. After a slave device is selected, the master starts clocking out the data through the MOSI line to the selected slave device. The master sends and receives one bit for every clock cycle. 8-bits can be exchanged after eight clock cycles. The master finishes communication by disabling the SS line. SPI is used frequently in handheld and other mobile platform systems.

[1] was an American multinational telecommunications company founded on September 25, 1928
[2] is a custom or convention that has achieved a dominant position by public acceptance or market forces

The design process of SPI

The SPI design is composed of mainly three modules:

- Master
- Slave
- Integration (controller)

And, of course, a test bench for each module. The master and slave should transmit data according to specific mode and the integration function is to control the selected slave and connect it with the master so the data transition happens only between them as shown in [Figure 2].

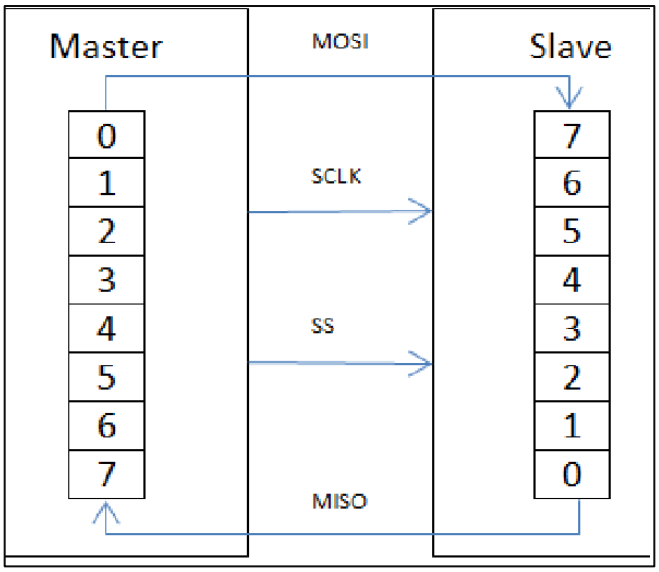

Figure 1- The process of data transition between master and slave

1. The design process of the master:

The master module takes five inputs:

a. [0:1] mode: to choose one mode of four modes to operate on
b. reset: to reset to the idle states
c. clock: that is taken from outside (the user)
d. [0:7] tx_data: 8 bit data that will be transmitted to slave
e. miso: the bit of data that master read at one clock cycle as data is read bit by bit

And has five outputs:

a. [0:7]rx_data: 8 bit data that is read from slave
b. mosi: bit that master send at this clock cycle
c. slave_clk: clock that master send to slave to operate with
d. tx_ready_flag: just two flags to indicate that the process of sending data is finished
e. rx_ready_flag: just two flags to indicate that the process of reading data is finished

First, we set the values of both clock phase and clock polarity depending on the selected mode so the clock phase is equal to one only in modes 1, 2, and clock polarity is equal to one only in modes 2 and three according to the table shown in [Figure 3].

SPI Mode	CPOL	CPHA	Clock Polarity in Idle State	Clock Phase Used to Sample and/or Shift the Data
0	0	0	Logic low	Data sampled on rising edge and shifted out on the falling edge
1	0	1	Logic low	Data sampled on the falling edge and shifted out on the rising edge
2	1	1	Logic high	Data sampled on the falling edge and shifted out on the rising edge
3	1	0	Logic high	Data sampled on the rising edge and shifted out on the falling edge

Figure 2- SPI modes table

Clock Setup

In reset state, we setup the clock of the slave depending on the clock polarity (Active low or Active high), store the data that will be read by the slave in register inside the master, setting the value of the trailing edge and leading edge to 1 and 0 respectively.

In any other case, we complement the clock given to the slave, the trailing edge and the falling edge.

Master-in Slave-out setup

In reset state, the counter of transition is set to zero, in any other case there are two conditions

a. The master has received the bit from the slave, the transition ready flag is not active which means that the master did not read all the 8 bits yet, and the trailing or falling edge is active according to the selected mode, if so, the master takes the input bit, store it in LSB, shift the total 8 bit data to the left and increases the counter of reading data.

b. The counter of reading data is equal to 1000(8) which mean the master has finished reading all eight bits so we set the flag of reading data to 1.

Master-out Slave-in setup

In reset state, the counter of transition is set to zero, in any other case there are two conditions

a. The transition ready flag is not active which mean that the master did not send all the 8 bit yet, and the trailing or falling edge is active according to the selected mode, if so, the master send the MSB by storing it in the output variable and increases the counter of sending data.
b. The counter of sending data is equal to 1000(8) which mean the master has finished sending all eight bits, so we set the ready flag of sending data to 1.

2. The design process of the master:

The design process of the slave is the same as the master except for having one extra input; the slave select (SC), which is responsible for enabling the selected slave, SC is active low so if a specific slave is selected it must be set to zero, also, before the beginning of reading or sending any data, we must check if it's enabled or not, so if it's enabled the transition of data will start normally, otherwise, nothing will be sent or read.

3. The design process of the Integration(controller):

The integration module takes six inputs:

a. [0:1] mode: for both master and slave
b. Reset: for both master & slave
c. Clk: for master only
d. [0:1] C_S: slave chip select
e. [0:7]M_tx_data: data to be transmitted by master
f. [0:7]S_tx_data: data to be transmitted by slave

And has two outputs:

a. [0:7]M_rx_data: data received by master
b. [0:7]S_rx_data: data received by Slave

In integration module, we made three instances of slave module and one instance of master module, and depending on the C_S we connect the miso of selected slave to the master and mosi of the master to the selected slave and connect the output full 8 bit data of the master and the selected slave with the output of the integration module and thus, the SPI module will work properly and the integration module will be the same as [Figure 4].

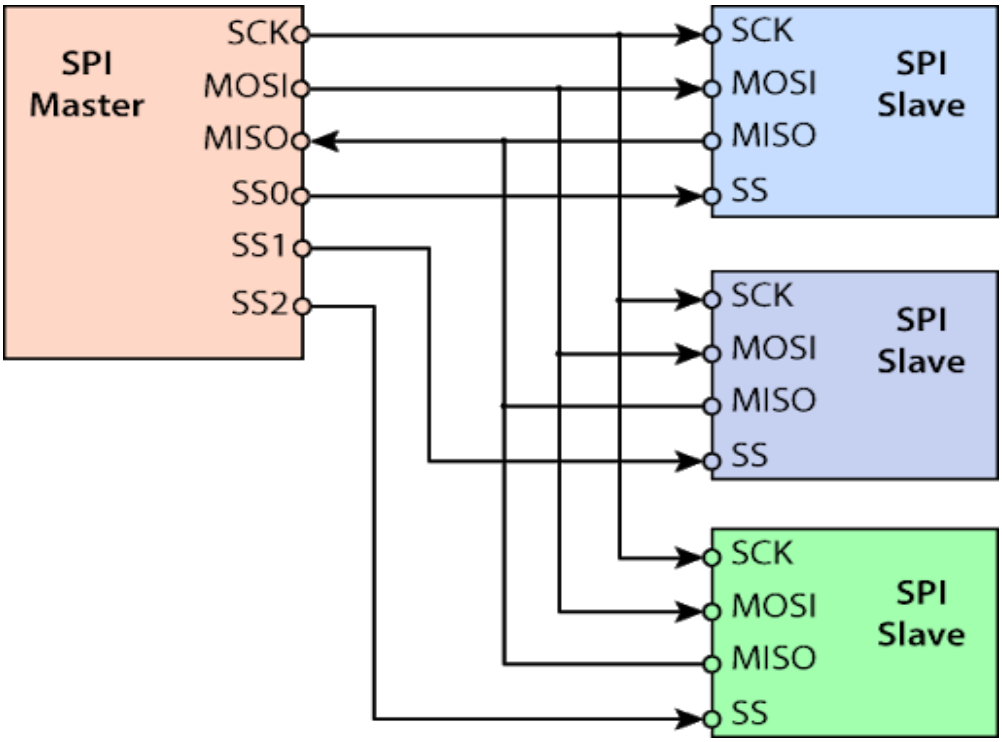

Figure 3-SPI integration

Test cases and Simulation:

1-Master:
we worked for four test cases to test master in all modes as we test how (MISO) and (MOSI) worked in each mode.

Test cases process in master:
- we monitor the initial data in master and the data after shifting with clk.
- we display the Expected data out from master and check if the data out equal our expected data if it is correct we display that is passed else we display it is failed.

Test case 1#
In this test case we aim to test master in mode #0.
Operation: send initial byte to master then data bit by bit to miso then checking if master work successfully in mode #0.

```
###############TestCase(1)---->mode #0##############
# clk   MISO MOSI      initial_data     out_shift
# 0     x    x         10100101         10100101
# 1     x    x         10100101         10100101
# 0     x    x         10100101         10100101
# 1     1    1         10100101         10100101
# 0     1    1         10100101         01001011
# 1     0    0         10100101         01001011
# 0     0    0         10100101         10010110
# 1     1    1         10100101         10010110
# 0     1    1         10100101         00101101
# 1     1    0         10100101         00101101
# 0     1    0         10100101         01011011
# 1     1    0         10100101         01011011
# 0     1    0         10100101         10110111
# 1     0    1         10100101         10110111
# 0     0    1         10100101         01101110
# 1     1    0         10100101         01101110
# 0     1    0         10100101         11011101
# 1     0    1         10100101         11011101
# 0     0    1         10100101         10111010
#
# initial data in master=10100101
# Expected dataout = 10111010
# Exact dataout    = 10111010
# Test case #1 is passed successfuly
#
```

Figure 1 Master command test case 1

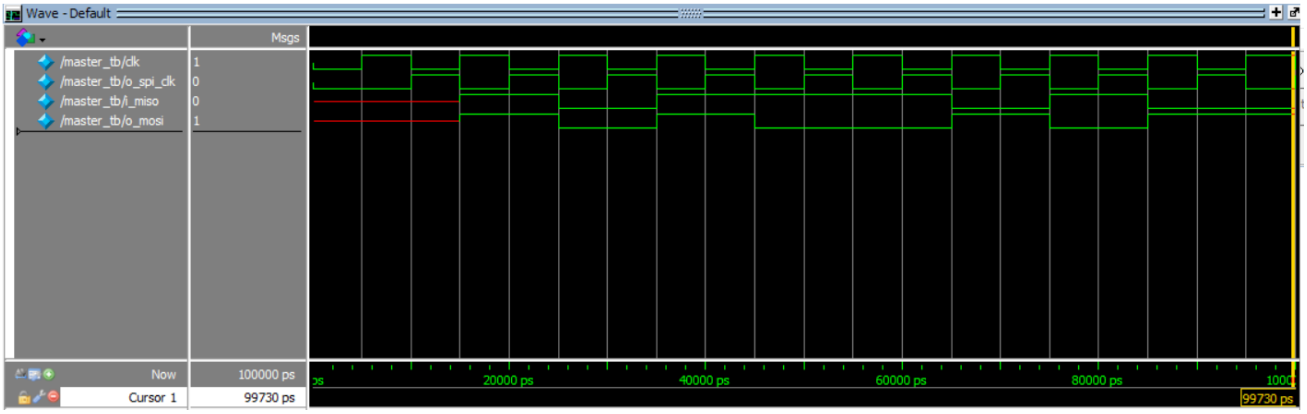

Figure 2 Master waveform in mode 0

Test case #2

In this test case we aim to test master in mode #1.

Operation: send initial byte to master then data bit by bit to miso then checking if master work successfully in mode #1.

```
###############TestCase(2)---->mode #1###############
#
# clk   MISO MOSI      initial_data     out_shift
# 1     0    1         10100101        10100101
# 0     0    1         10100101        10100101
# 1     0    1         10100101        10100101
# 0     1    1         10100101        01001011
# 1     1    0         10100101        01001011
# 0     0    0         10100101        10010110
# 1     0    1         10100101        10010110
# 0     1    1         10100101        00101101
# 1     1    0         10100101        00101101
# 0     1    0         10100101        01011011
# 1     1    0         10100101        01011011
# 0     1    0         10100101        10110111
# 1     1    1         10100101        10110111
# 0     0    1         10100101        01101110
# 1     0    0         10100101        01101110
# 0     1    0         10100101        11011101
# 1     1    1         10100101        11011101
# 0     0    1         10100101        10111010
# 1     0    1         10100101        10111010
#
# initial data in master=10100101
# Expected dataout = 10111010
# Exact dataout    = 10111010
# Test case #2 is passed successfuly
```

Figure 3 Master command test case 2

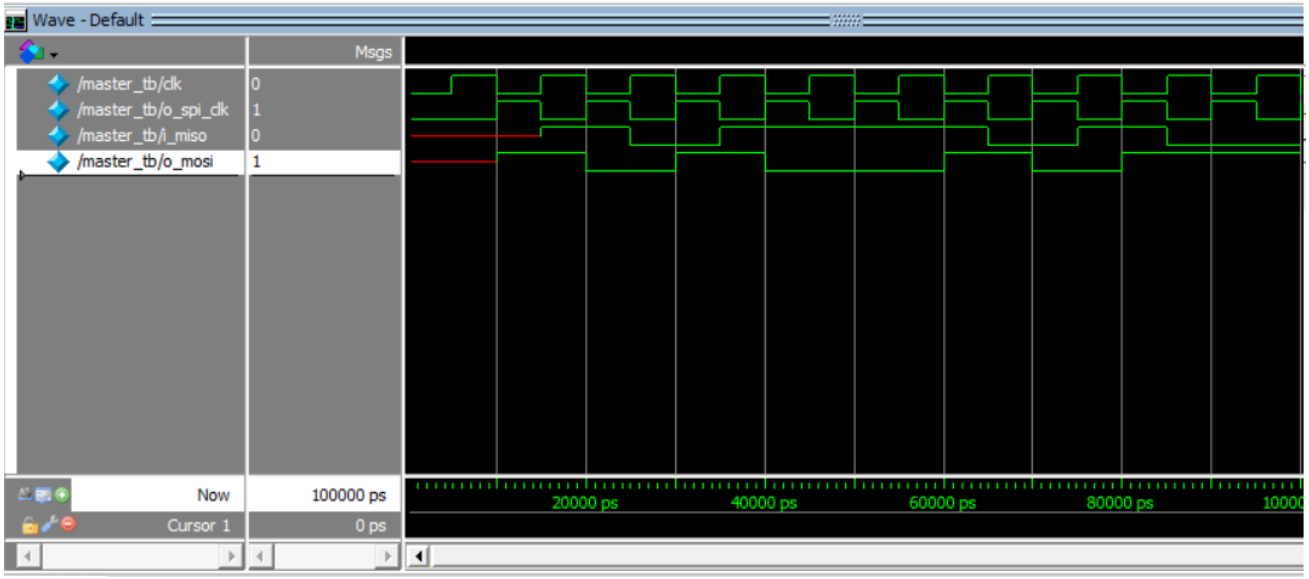

Figure 4 Master Waveform in mode 1

Test case #3

in this case we aim to test master in mode #2

Operation: send initial byte to master then data bit by bit to miso then checking if master work successfully in mode #2.

```
###############TestCase(3)---->mode #2##############
#
# clk   MISO MOSI      initial_data     out_shift
# 0     0    1         10100101         10100101
# 1     0    1         10100101         10100101
# 0     0    1         10100101         10100101
# 1     1    1         10100101         01001011
# 0     1    0         10100101         01001011
# 1     0    0         10100101         10010110
# 0     0    1         10100101         10010110
# 1     1    1         10100101         00101101
# 0     1    0         10100101         00101101
# 1     1    0         10100101         01011011
# 0     1    0         10100101         01011011
# 1     1    0         10100101         10110111
# 0     1    1         10100101         10110111
# 1     0    1         10100101         01101110
# 0     0    0         10100101         01101110
# 1     1    0         10100101         11011101
# 0     1    1         10100101         11011101
# 1     0    1         10100101         10111010
# 0     0    1         10100101         10111010
#
# initial data in master=10100101
# Expected dataout = 10111010
# Exact dataout    = 10111010
```

Figure 5 Master command in test case 3

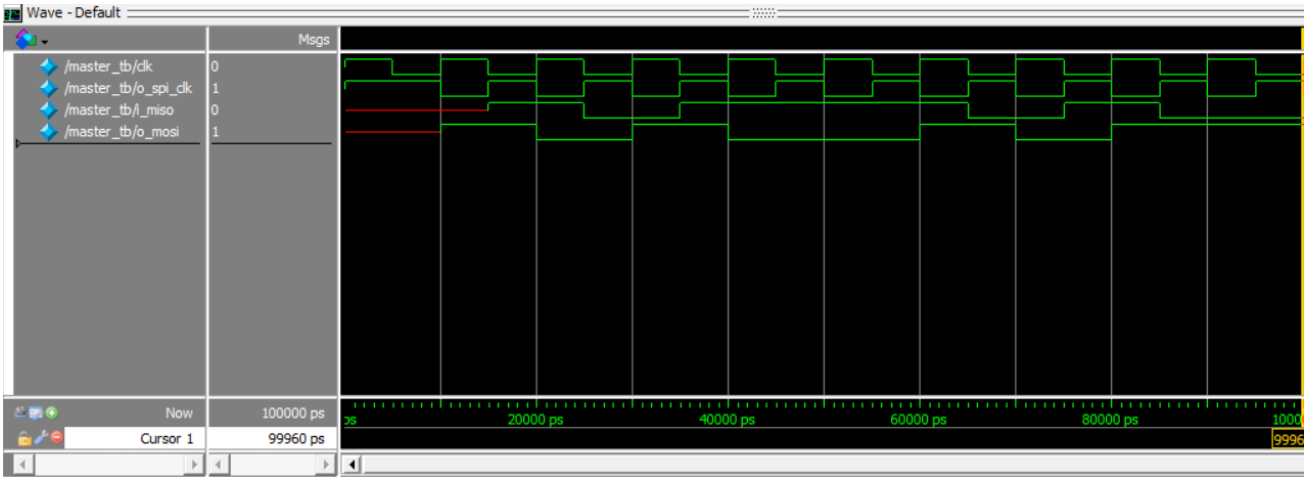

Figure 6 Master Waveform in mode 2

Test case #4

in this case we aim to test master in mode #3

Operation: send initial byte to master then data bit by bit to miso then checking if master work successfully in mode #3.

```
###############TestCase(4)---->mode #3###############
#
# clk   MISO MOSI      initial_data     out_shift
# 0     0    1         10100101         10100101
# 1     0    1         10100101         10100101
# 0     0    1         10100101         10100101
# 1     1    1         10100101         10100101
# 0     1    1         10100101         01001011
# 1     0    0         10100101         01001011
# 0     0    0         10100101         10010110
# 1     1    1         10100101         10010110
# 0     1    1         10100101         00101101
# 1     1    0         10100101         00101101
# 0     1    0         10100101         01011011
# 1     1    0         10100101         01011011
# 0     1    0         10100101         10110111
# 1     0    1         10100101         10110111
# 0     0    1         10100101         01101110
# 1     1    0         10100101         01101110
# 0     1    0         10100101         11011101
# 1     0    1         10100101         11011101
# 0     0    1         10100101         10111010
#
# initial data in master=10100101
# Expected dataout = 10111010
# Exact dataout    = 10111010
# Test case #4 is passed successfuly
```

Figure 7 Master command for test case 4

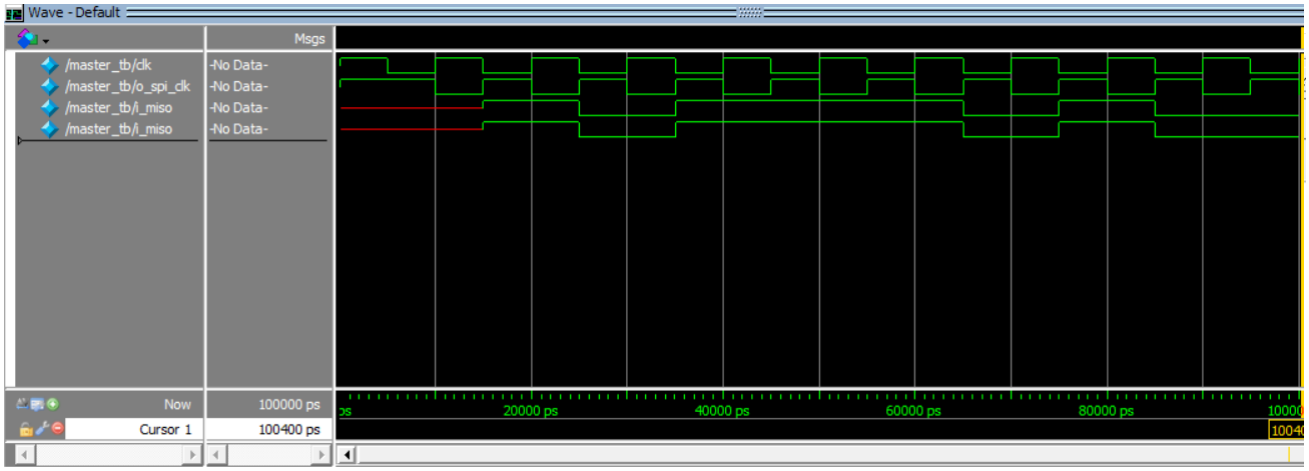

Figure 8 Master Waveform in mode 3

2-Slave:

we worked for #4 test cases to test all possible communication for slave for all modes

Test case #1

in this case we aim to test Slave in mode #0

```
################mode #0###############
#
# time    clk  MOSI MISO  datain     dataout
# 0         0    x    x     01010101   01010101
# 5         1    x    x     01010101   01010101
# 10        0    x    x     01010101   01010101
# 15        1    0    0     01010101   01010101
# 20        0    0    0     01010101   10101010
# 25        1    0    1     01010101   10101010
# 30        0    0    1     01010101   01010100
# 35        1    0    0     01010101   01010100
# 40        0    0    0     01010101   10101000
# 45        1    0    1     01010101   10101000
# 50        0    0    1     01010101   01010000
# 55        1    0    0     01010101   01010000
# 60        0    0    0     01010101   10100000
# 65        1    0    1     01010101   10100000
# 70        0    0    1     01010101   01000000
# 75        1    0    0     01010101   01000000
# 80        0    0    0     01010101   10000000
# 85        1    0    1     01010101   10000000
# 90        0    0    1     01010101   00000000
# Expected dataout = 00000000
# Exact dataout    = 00000000
#
```

Figure 1 Slave command for test case 0

Figure 2 Slave Waveform in mode 0

Test case #2

in this case we aim to test Slave in mode #1

```
###############mode #1##############
#
# time    clk  MOSI MISO  datain     dataout
# 95       1    0    1     11111111   11111111
# 100      0    0    1     11111111   11111111
# 105      1    0    1     11111111   11111111
# 110      0    0    1     11111111   11111110
# 115      1    0    1     11111111   11111110
# 120      0    1    1     11111111   11111101
# 125      1    1    1     11111111   11111101
# 130      0    1    1     11111111   11111011
# 135      1    1    1     11111111   11111011
# 140      0    0    1     11111111   11110110
# 145      1    0    1     11111111   11110110
# 150      0    1    1     11111111   11101101
# 155      1    1    1     11111111   11101101
# 160      0    1    1     11111111   11011011
# 165      1    1    1     11111111   11011011
# 170      0    0    1     11111111   10110110
# 175      1    0    1     11111111   10110110
# 180      0    0    1     11111111   01101100
# 185      1    0    1     11111111   01101100
# Expected dataout = 01101100
# Exact dataout    = 01101100
#
```

Figure 3 Slave command for test case 2

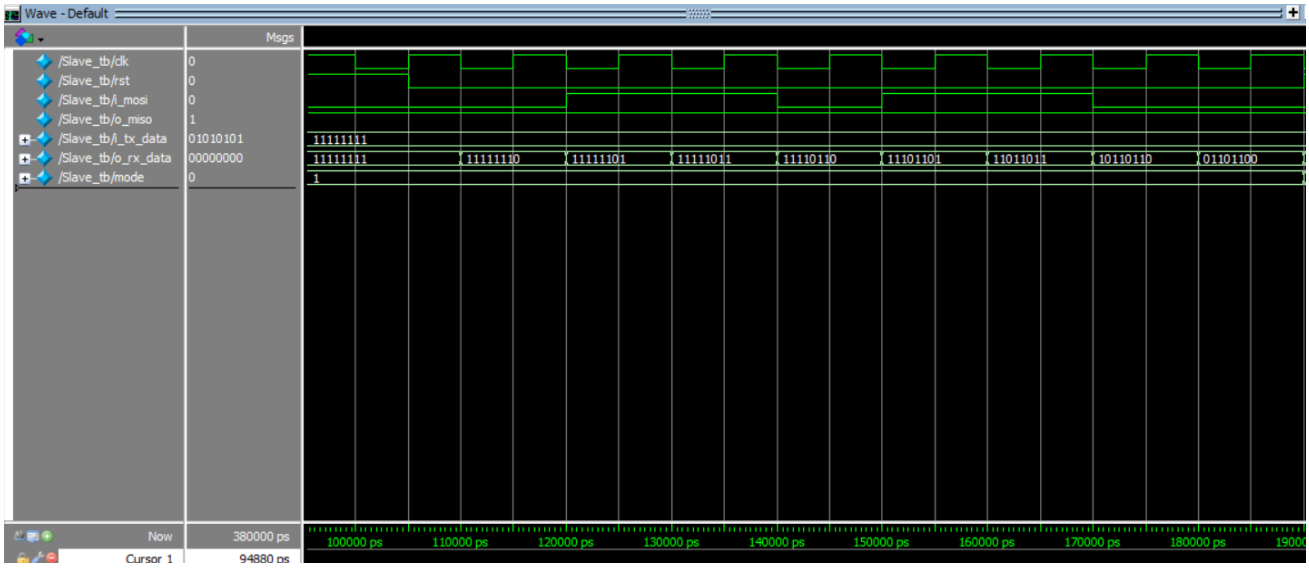

Figure 4 Slave Waveform in mode 1

Test case #3

in this case we aim to test Slave in mode #2

```
###############mode #2###############
#
# time    clk  MOSI MISO  datain     dataout
# 190      0    0    1     11111111   11111111
# 195      1    0    1     11111111   11111111
# 200      0    0    1     11111111   11111111
# 205      1    1    1     11111111   11111111
# 210      0    1    1     11111111   11111111
# 215      1    0    1     11111111   11111110
# 220      0    0    1     11111111   11111110
# 225      1    1    1     11111111   11111101
# 230      0    1    1     11111111   11111101
# 235      1    1    1     11111111   11111011
# 240      0    1    1     11111111   11111011
# 245      1    0    1     11111111   11110110
# 250      0    0    1     11111111   11110110
# 255      1    0    1     11111111   11101100
# 260      0    0    1     11111111   11101100
# 265      1    1    1     11111111   11011001
# 270      0    1    1     11111111   11011001
# 275      1    0    1     11111111   10110010
# 280      0    0    1     11111111   10110010
# Expected dataout = 10110010
# Exact dataout    = 10110010
#
```

Figure 5 Slave command for test case 3

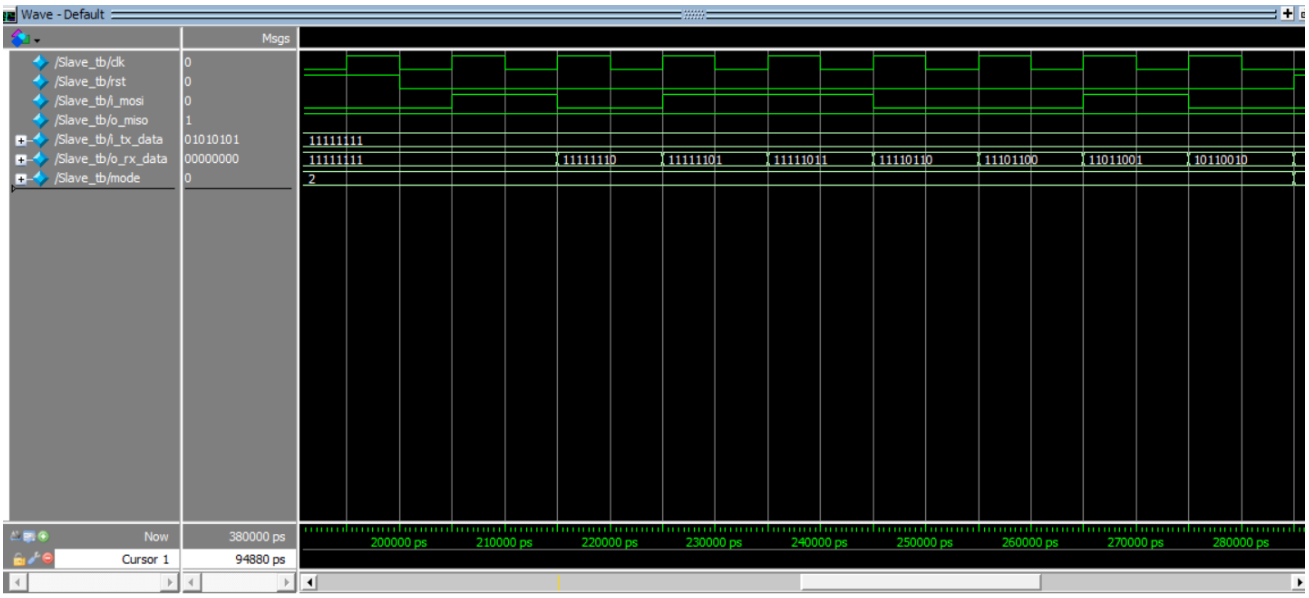

Figure 6 Slave Waveform in mode 2

Test case #4

in this case we aim to test Slave in mode #3

```
#
###############mode #3##############
#
# time    clk  MOSI MISO  datain     dataout
# 285      0    0    1     11111111   11111111
# 290      1    0    1     11111111   11111111
# 295      0    0    1     11111111   11111111
# 300      1    0    1     11111111   11111111
# 305      0    0    1     11111111   11111110
# 310      1    1    1     11111111   11111110
# 315      0    1    1     11111111   11111101
# 320      1    1    1     11111111   11111101
# 325      0    1    1     11111111   11111011
# 330      1    0    1     11111111   11111011
# 335      0    0    1     11111111   11110110
# 340      1    0    1     11111111   11110110
# 345      0    0    1     11111111   11101100
# 350      1    1    1     11111111   11101100
# 355      0    1    1     11111111   11011001
# 360      1    1    1     11111111   11011001
# 365      0    1    1     11111111   10110011
# 370      1    0    1     11111111   10110011
# 375      0    0    1     11111111   01100110
# Expected dataout = 01100110
# Exact dataout    = 01100110
```

Figure 7 Slave command for test case 4

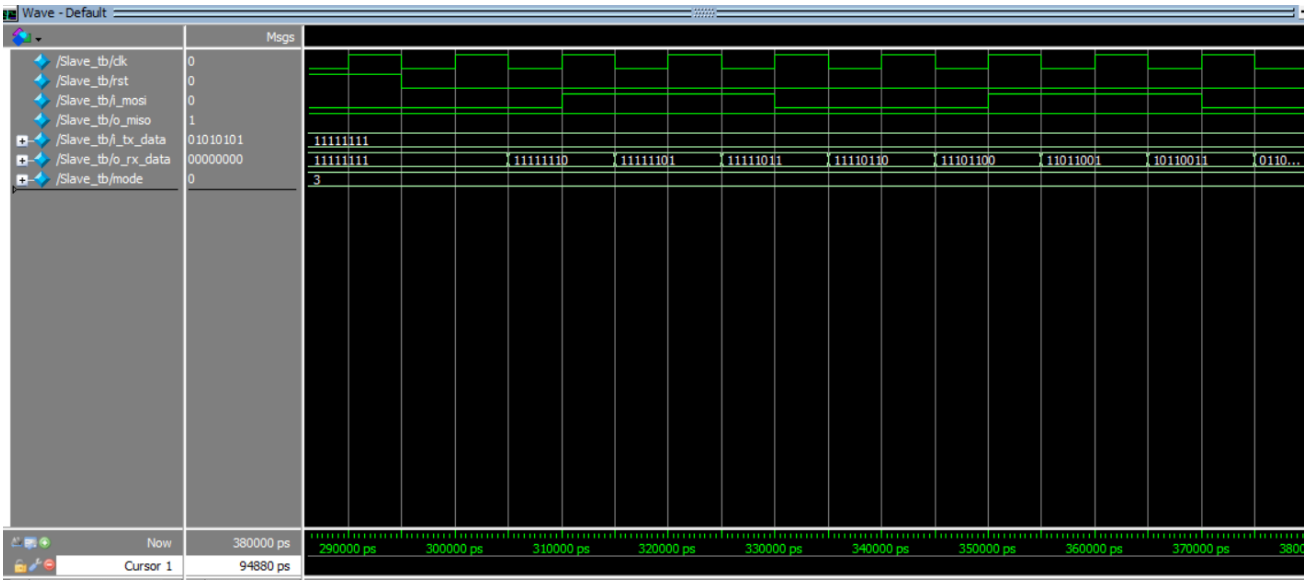

Figure 8 Slave Waveform in mode 3

3-SPI Integration

We aim in SPI Integration test bench to test all possible communication between master and slave in all modes.

- we work for #12 test case.

Test cases #1 #2 #3

- In all this test cases we work to test SPI Integration in mode #0.

-Test case #1 aims to test all possible integration between the Master and Slave #1.

-Test case #2 aims to test all possible integration between the Master and Slave #2.

-Test case #3 aims to test all possible integration between the Master and Slave #3.

```
#  ######################Test Case (1)---------Mode 0-----############################
#  Hey Master , Send this data 00100111 to Slave #1 and read the data from it and give it to me
#
#  Data in master                 = 00100111
#  Data in slave #1               = 00101010
#
#  Expected Data in master        = 00101010
#  Expected Data in slave #1      = 00100111
#
#  Output data from master        = 00101010
#  Output data from slave #1      = 00100111
#  ######################Test Case (2)---------Mode 0-----############################
#  Hey Master , Send this data 10101111 to Slave #2 and read the data from it and give it to me
#
#  Data in master                 = 10101111
#  Data in slave #2               = 00000000
#
#  Expected Data in master        = 00000000
#  Expected Data in slave #2      = 10101111
#
#  Output data from master        = 00000000
#  Output data from slave #2      = 10101111
#  ######################Test Case (3)---------Mode 0-----############################
#  Hey Master , Send this data 11001100 to Slave #3 and read the data from it and give it to me
#
#  Data in master                 = 11001100
#  Data in slave #3               = 00110011
#
#  Expected Data in master        = 00110011
#  Expected Data in slave #3      = 11001100
#
#  Output data from master        = 00110011
#  Output data from slave #3      = 11001100
```

Figure 1 SPI Integration test bench command in mode 0 test cases #1 #2 #3

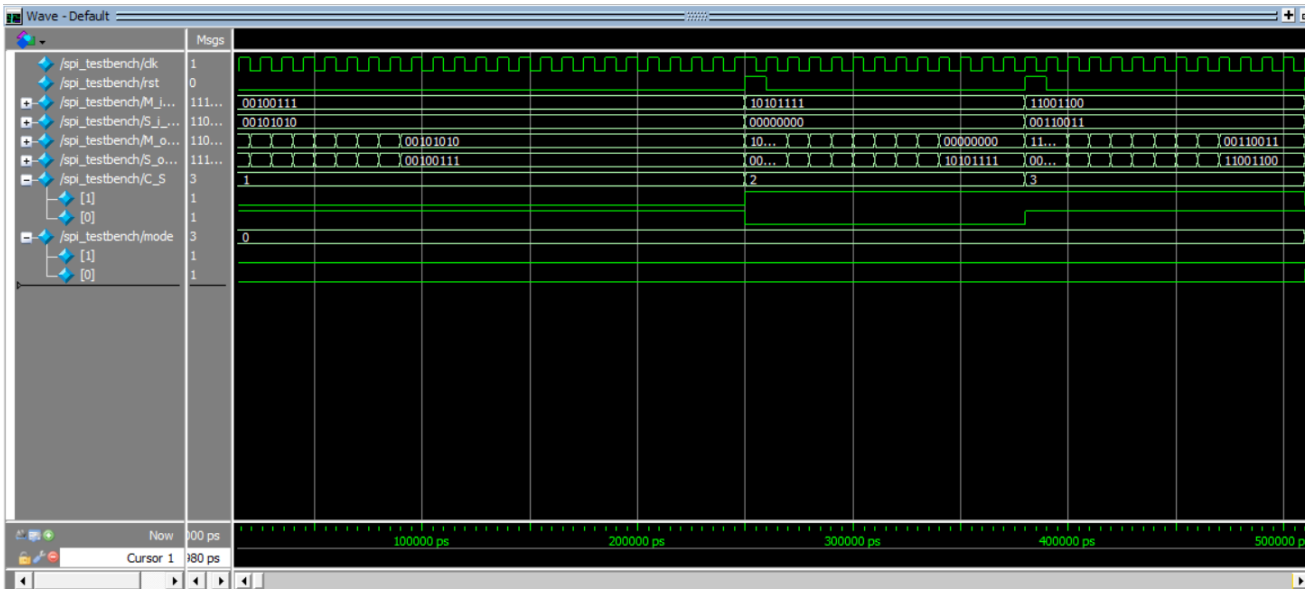

Figure 2 SPI Integration Waveform in mode 0 with Slave #1, Slave #2, Slave #3

Test cases #4 #5 #6

- In all this test cases we work to test SPI Integration in mode #1.

-Test case #4 aims to test all possible integration between the Master and Slave #1.

-Test case #5 aims to test all possible integration between the Master and Slave #2.

-Test case #6 aims to test all possible integration between the Master and Slave #3.

```
#  ######################Test Case (4)---------Mode 1-----############################
#  Hey Master , Send your data 00000011 to Slave #1 and read the data from it and give it to me
#
#  Data in master                 = 00000011
#  Data in slave #1               = 11100111
#
#  Expected Data in master        = 11100111
#  Expected Data in slave         = 00000011
#
#  Output data from master        = 11100111
#  Output data from slave #1      = 00000011
#  ######################Test Case (5)---------Mode 1-----############################
#  Hey Master , Send your data 00100100 to Slave #2 and read the data from it and give it to me
#
#  Data in master                 = 00100100
#  Data in slave #2               = 00000111
#
#  Expected Data in master        = 00000111
#  Expected Data in slave         = 00100100
#
#  Output data from master        = 00000111
#  Output data from slave #2      = 00100100
#  ######################Test Case (6)---------Mode 1-----############################
#  Hey Master , Send your data 00010011 to Slave #3 and read the data from it and give it to me
#
#  Data in master                 = 00010011
#  Data in slave #3               = 00000111
#
#  Expected Data in master        = 00000111
#  Expected Data in slave         = 00010011
#
#  Output data from master        = 00000111
#  Output data from slave #3      = 00010011
```

Figure 3 SPI Integration test bench command in mode 1 test cases #1 #2 #3

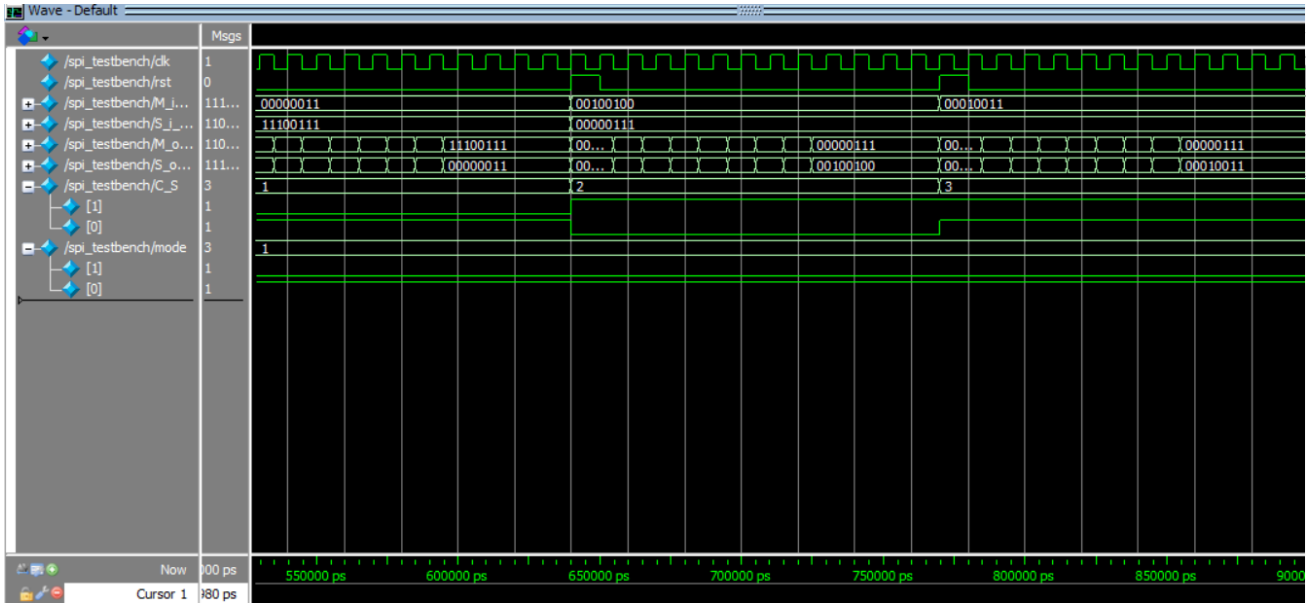

Figure 4 SPI Integration Waveform in mode 1 with Slave #1, Slave #2, Slave #3

Test cases #7 #8 #9

- In all this test cases we work to test SPI Integration in mode #2.

-Test case #7 aims to test all possible integration between the Master and Slave #1.

-Test case #8 aims to test all possible integration between the Master and Slave #2.

-Test case #9 aims to test all possible integration between the Master and Slave #3.

```
#  ########################Test Case (7)---------Mode 2-----##############################
#  Hey Master , Send your data 00000100 to Slave #1 and read the data from it and give it to me
#
#  Data in master                  = 00000100
#  Data in slave #1                = 01100100
#
#  Expected Data in master         = 01100100
#  Expected Data in slave #1       = 00000100
#
#  Output data from master         = 01100100
#  Output data from slave #1       = 00000100
#  ########################Test Case (8)---------Mode 2-----##############################
#  Hey Master , Send your data 00100010 to Slave #2 and read the data from it and give it to me
#
#  Data in master                  = 00100010
#  Data in slave #2                = 01100100
#
#  Expected Data in master         = 01100100
#  Expected Data in slave #2       = 00100010
#
#  Output data from master         = 01100100
#  Output data from slave #2       = 00100010
#  ########################Test Case (9)---------Mode 2-----##############################
#  Hey Master , Send your data 01100100 to Slave #3 and read the data from it and give it to me
#
#  Data in master                  = 01100100
#  Data in slave #3                = 00000000
#
#  Expected Data in master         = 00000000
#  Expected Data in slave #3       = 11111111
#
#  Output data from master         = 00000000
#  Output data from slave #3       = 11111111
```

Figure 5 SPI Integration test bench command in mode 2 test cases #1 #2 #3

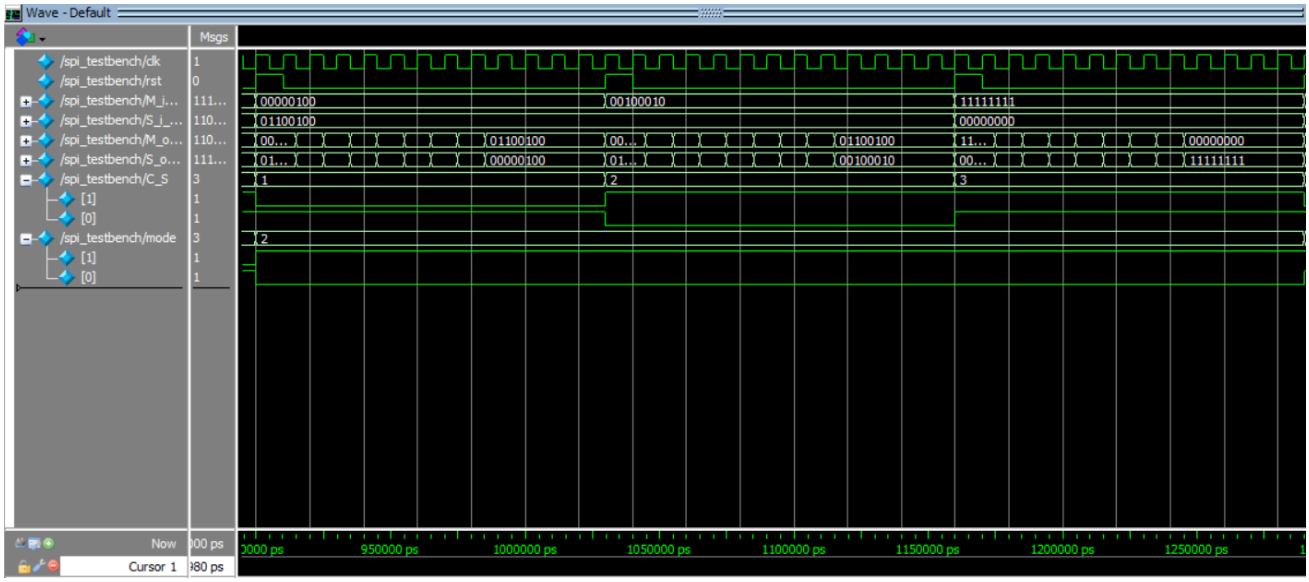

Figure 6 SPI Integration Waveform in mode 2 with Slave #1, Slave #2, Slave #3

Test cases #10 #11 #12

- In all this test cases we work to test SPI Integration in mode #0.

-Test case #10 aims to test all possible integration between the Master and Slave #1.

-Test case #11 aims to test all possible integration between the Master and Slave #2.

-Test case #12 aims to test all possible integration between the Master and Slave #3.

```
#  ########################Test Case (10)---------Mode 3-----##############################
#  Hey Master , Send your data 00000000 to Slave #1 and read the data from it and give it to me
#
#  Data in master                  = 00000000
#  Data in slave #1                = 11001000
#
#  Expected Data in master         = 11001000
#  Expected Data in slave #1       = 10001111
#
#  Output data from master         = 11001000
#  Output data from slave #1       = 10001111
#  ########################Test Case (11)---------Mode 3-----##############################
#  Hey Master , Send your data 11001000 to Slave #2 and read the data from it and give it to me
#
#  Data in master                  = 11001000
#  Data in slave #2                = 11001000
#
#  Expected Data in master         = 11001000
#  Expected Data in slave #2       = 10001111
#
#  Output data from master         = 11001000
#  Output data from slave #2       = 10001111
#  ########################Test Case (12)---------Mode 3-----##############################
#  Hey Master , Send your data 11001000 to Slave #3 and read the data from it and give it to me
#
#  Data in master                  = 11001000
#  Data in slave #3                = 11001000
#
#  Expected Data in master         = 11001000
#  Expected Data in slave #3       = 11111000
#
#  Output data from master         = 11001000
#  Output data from slave #3       = 11111000
#  Finished with  12  Successful,  0 failed test cases out  of 12 test cases
```

Figure 7 SPI Integration test bench command in mode 3 test cases #1 #2 #3

-as we shown in the end of .fig 17 we calculate the number of successful communication and the number of failed communication.

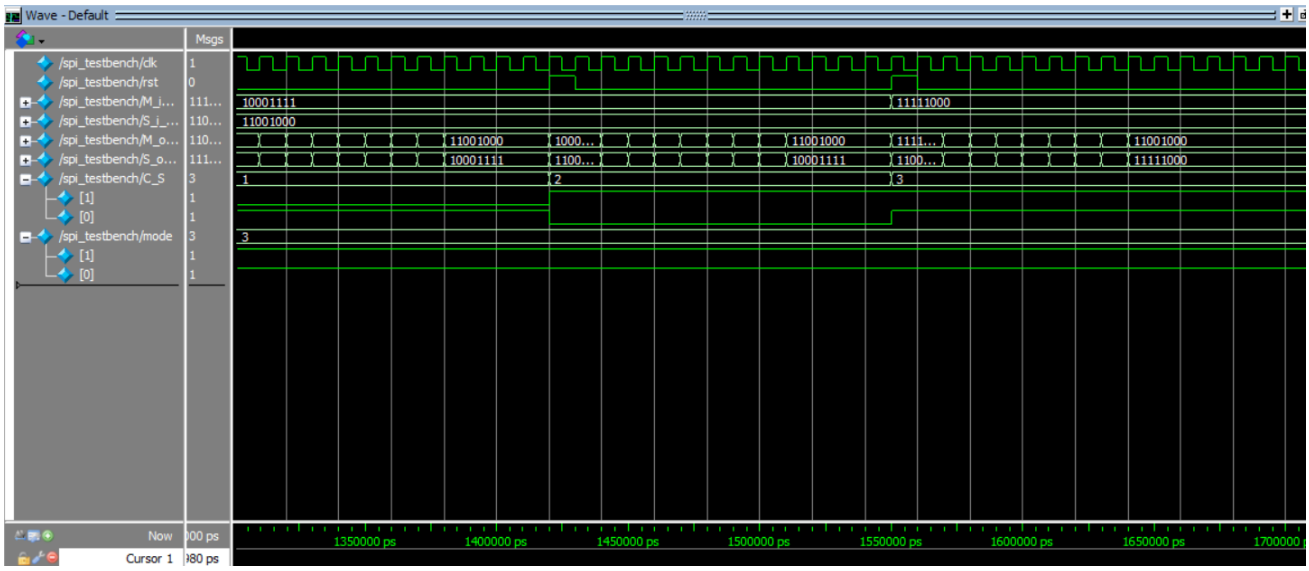

Figure 8 SPI Integration Waveform in mode 3 with Slave #1, Slave #2, Slave #3

Serial peripheral interface (SPI):

Is a synchronous serial communication interface which means the interface is controlled by a clock, SPI devices communicate in full duplex mode using a master and a slave, in other words you the slave and the master can send and receive data in the same time.

Interface:

Usually it is used in 4-wire interface which include 4 signals:

- Clock (SPI CLK, SCLK)
- Chip select (CS)
- Master out, slave in (MOSI)
- Master in, slave out (MISO)

Clock: The master is responsible for generating the clock signal and is send to the slave through the SCKL signal; Data transmitted between the master and the slave is synchronized to the clock generated by the master.

CS: SPI have only one master but can have several slaves, the master activates the slave to communicate with using CS signal which is usually active low.

MOSI: is the data line which transmits data from master to slave.

MISO: is the data line which transmits data from slave to master.

Data transmission and modes:

the data is transmitted bit by bit, SPI protocol sample this bit and shift it on different edges to avoid any interfering between bits, for flexibility it provides different modes which can be selected using CPOL bit and CPHA bit ,the master can set the clock polarity during idle state (which is the state at which no data is being transmitted nor received) using CPOL bit and it can also selected the edge at which it samples data or shift it, hence we have 4 modes as the table shows.

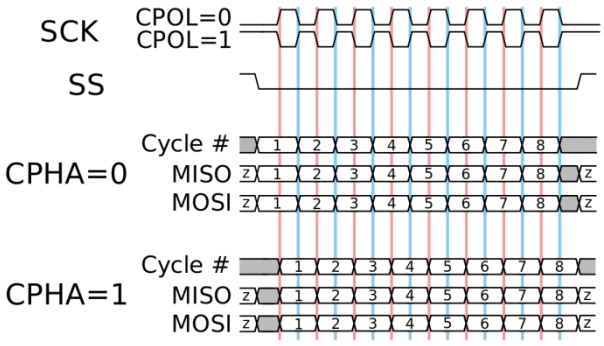

Figure 4

Inter-integrated circuit (I2C):

I2C is a synchronous serial communication half duplex protocol , data is transmitted bit by bit and transmission is controlled by a clock, using I2C we can either connect multiple slaves to a single master (like SPI) and you can have, multiple masters and single or multiple slaves.

Interface:

The protocol uses two wires to transmit and receive data

- Serial Data (SDA)
- Serial Clock (SCL)

SDA: is the data line through which the slave and the master transmit and receive data

SCL: is the clock line that carries the clock signal, the clock is controlled by the master only.

In i2c data is transferred in messages, which is broken up into frames of data, there is a frame for the address of the slave and one or more frames for data, the message should also contain start and stop conditions, read/write bit and ACK/NACK bit.

Start condition: indicates that there is a message, so it tells all slave to be attention, to do this the SDA line goes form high to low before SCL line goes from high to low

Stop condition: indicates the end of the message, to do this the SCL goes from low to high before the SDA goes from low to high.

Read/Write bit: a single bit and it is the least significant bit in the address frame, that indicates whether the master wishes to send data to the slave in this case will be (low voltage level) or receive data from the slave in this case (high voltage level).

ACK/NACK bit: after the address frame or the data frame the device which received this frame whether it is address frame or data frame should send an acknowledge bit which is pulling the SDA line to (low voltage level)

Address frame: a 7- or 10-bit sequence, in order to select the slave to communicate with the master send the slave's address in this frame to all the slaves, and the slave that carries this address shall send ACK bit.

Data frame: after sending the address frame and the Read/Write bit and the master received the ACK bit from the slave, the data is send from the master or the slave, the data frame consists of 8-bit long and the most significant bit first, after each 8-bit data frame the receiving should send ACK bit.

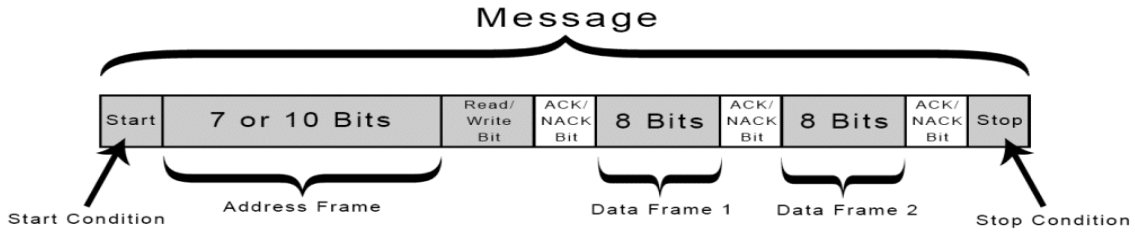

Figure 5

Steps of sending a message:

1. Master sends Start condition
2. Master sends Address frame + Read/write bit
3. Slave sends a ACK bit
4. The slave or the master send 8-bit data frame
5. The slave or the master send ACK bit
6. The master sends stop condition

Note steps 4 and 5 can be repeated several times.

Universal Asynchronous Receiver/Transmitter (UART):

UART is an asynchronous serial communication protocol it can be half or full duplex and it is the most common of this type, so there is no clock involved, it is a cheap communication device with a single transmitter / receiver.

UART interface:

This protocol uses two lines to transmit and receive data between only one master and one slave, Since it is an asynchronous protocol it replaces the clock with a start bit and an end bit, it also introduces a new concept which is the baud rate which is the

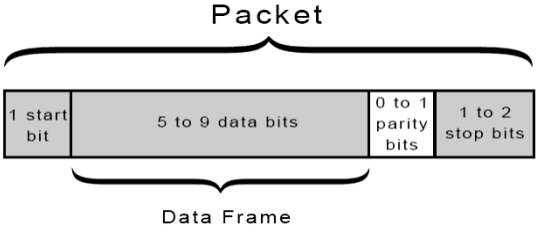

Figure 6

speed of data transfer measured in bits per second(bps), the transmitting and the receiving devices must operate at the same baud rate or the data will be corrupted, there is also another new concept which is the parity bits it is used to ensure the integrity of the transmitted data.

Start bit:

Data transmission line is normally held high when it is not transmitting data, when a device want to send data, it pulls the transmission line to low and that is the start bit, when the receiving device detect this bit it starts to read the data-frame which is usually 8-bit data frame.

Data frame:

The transmitted data which is usually 8-bit frame

Parity bit:

Parity bit usually is an indication for the number of 1 bits in the data frame, which show if number of 1 bits in the data frame even or odd number, and this can be helpful to deduct if the data frame is corrupted or not because the receiving device can count the number of 1 bits in the data frame and compare it to the parity bit.

Stop bit:

After the data frame and the parity bits sending device pulls the line again to high indicating the end of the transmission, and that is the stop bit.

Work distribution

Project:

-Master module && Master testbench module:

Yousef Atef Tawfik

-Slave module:

We ‘am Bassem Hosni

-Slave testbench module && SPI integration module:

Donia Abdelfattah

- SPI integration testbench module:

Yousef Ahmed Anwar

Report:

-The design process of the Master, Slave and Integration.

Donia Abdelfattah

- The test cases and simulation results

We ‘am Bassem Hosni

Yousef Ahmed Anwar

- A comparison between the SPI, I2C and UART.

Yousef Atef Tawfik

References

1) *Introduction to I²C and SPI protocols.* [Online] https://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/?/article/AA-00255/22/Introduction-to-SPI-and-IC-protocols.html.
2) [Online] https://www.epanorama.net/links/serialbus.html.
3) Design and Verification of Serial Peripheral Interface . [Online] https://www.ijedr.org/papers/IJEDR1303026.pdf.