Faculty of Engineering

Computer Engineering Department

Logic Design

Serial Peripheral Interface

Presented by

Ahmed Khaled Mahmod
Sec:1 BN:5

Hazem Mahmod Abdo
Sec:1 BN:26

Michael Aziz Fahem
Sec:2 BN:8

Mohamed Saad Fathy
Sec:2 BN:14

2020

Work distribution Section

Verilog Code

Ahmed Khaled Mahmod: Slave

Hazem Mahmod Abdo: Master

Michael Aziz Fahem: Integration and Test Bench {not totally}

Mohamed Saad Fathy: Integration and Test Bench {not totally}

Report

Ahmed Khaled Mahmod: Test Cases Section

Hazem Mahmod Abdo: Simulation Results

Michael Aziz Fahem: Comparison Between Communication Protocols

Mohamed Saad Fathy: Design Process Section

Design Process Section

Master

In master we started with input and output registers and wires (MISO, MOSI, CLK ,CS1, CS2, CS3) then we made a memory register to store ,send and receive data from slaves then we made command register to be able to send specific commands to slaves to perform like read and write then we created 4-bit register called done which performs as 4-bit counter to differ between whether sending the command (8 cycles) or performing the command (8 cycles) then we started to initialize the registers with the data then we made an internal clock(the clock is generated with the master) for the clock we used always block to generate the synchronous signal for master and slaves then we made an always block to perform the communication, firstly we check if there is any active slave or not if so we identify whether we are sending command or performing and in performing cycles we identify which command is sent to perform the specific task of that command and we increment the counter done and if there is no active slave we leave everything as it is and finally we assigned the MOSI data wire to the command most significant bit wire in case of sending commands or memory most significant bit wire in case of performing the command.

Slave

In Slaves we started with input and output registers and wires (SDO, SOI, CLK, CS) then we made a memory register to store ,send and receive data from the master then we made command register to be able to receive specific commands from the master to perform specific tasks like read and write then we created 4-bit register called done which performs as 4-bit counter to differ between whether sending the command (8 cycles) or performing the command (8 cycles) then we started to initialize the registers with the data then we made an always block to perform the communication, firstly we if there is signal from the master if so we check if we are receiving or sending a command if performing a command the we check the type of the command then performing it then increment the counter done and if master isn't activating that slave we leave everything as it is and finally we assigned the SDO wire to memory most significant bit wire if the slave is communicating with master if not we assingend the wire to high impedance state to prevent corrupted data if the master is communicate with other slave.

Integration

We created a module called MasterSlave to integrate everything in and for test bench firstly we created 1 module for Master and 3 modules for Slaves and some wires for connections (clk, CS1, CS2, CS3, MOSI, MISO) then all of these wires are connected to the master then the wires (CS1, clk, MOSI, MISO) are connected to slave1 then the wires (CS2, clk, MOSI, MISO) are connected to slave2 then the wires (CS3, clk, MOSI, MISO) are connected to slave3.

Test Cases Section

1-Firstly the master selects slave1 to communicate with, then the write command is sent from master to slave1 within 8 clock cycles (8-bit of data) then the master sends the data to be written in slave1 within 8 clock cycles (8-bit of data).

2-Then the master ends communications with slave1 and selects slave2 to communicate with, then the read command is sent from master to slave2 within 8 clock cycles (8-bit of data) then slave2 sends the data to be written to the master within 8 clock cycles (8-bit of data).

3-Then the master ends communications with slave2 and selects slave3 to communicate with, then the replace command is sent from master to slave3 within 8 clock cycles (8-bit of data) then slave3 sends the data to be written to the master within 8 clock cycles (8-bit of data), at the same time the master also is sending data to be written in slave3 (Full Duplex).

Why These Specific Test Cases Are Chosen

In the first test we checked the ability to write the memory of slaves by the master(the ability of slaves to read the master memory correctly).

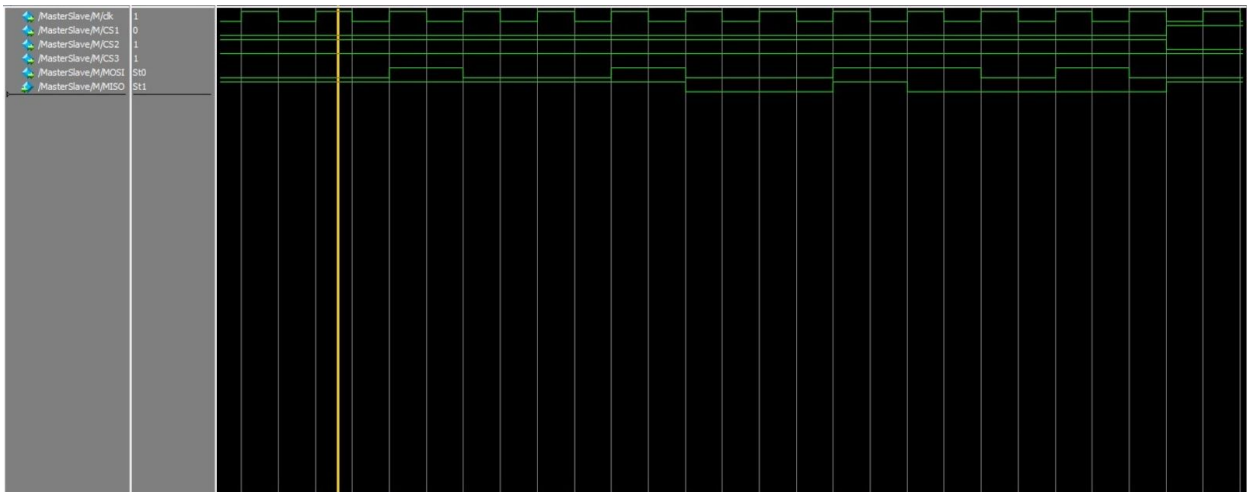
In the second test we checked the ability to write the memory of the master by slaves (the ability of the master to read slaves' memory correctly).

In the third test we checked the ability to write two memories at the same time the master is sending data to a slave at the same time the slave is sending data to the master (Full Duplex).

And during all of these three tests we checked the ability to communicate with specific slave and the ability of slaves to receive command correctly from the master.

Simulation Results

The results came out as expected, the master sends the data correctly through the MOSI line and receives the data correctly through MISO line and slaves sends the data correctly through the MISO and receive the data correctly through the MOSI line and slaves are correctly chosen to communicate by the master and there is no interference between the data that comes from slaves (no corrupted data).

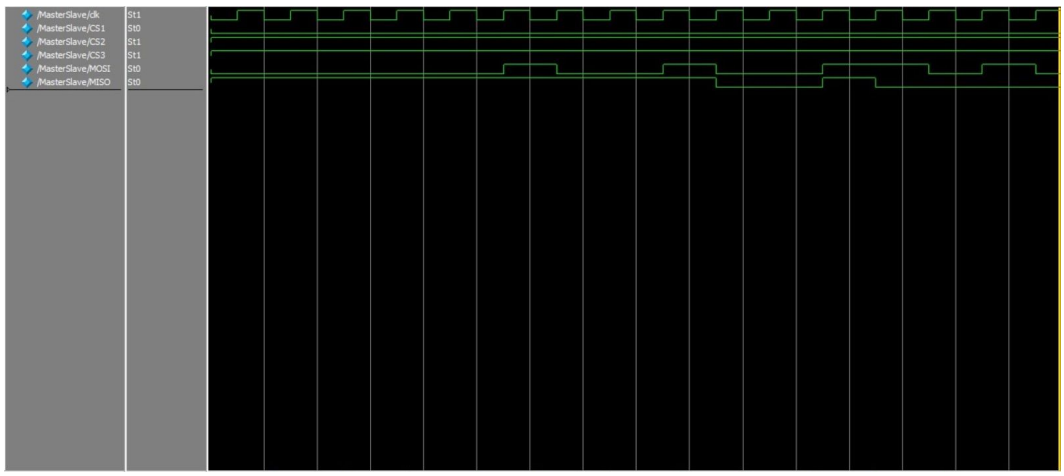


And it is important to note that a self checking test bench is implemented to automatically test the functionality of the master and slave modules and that self checking test bench shows that all tests are passed.

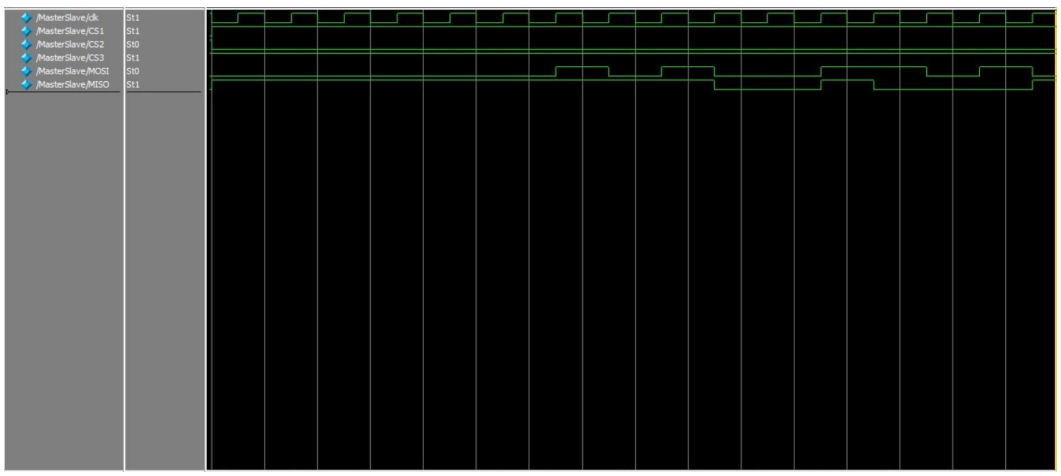
```
add wave -position end  sim:/MasterSlave/M/clk
add wave -position end  sim:/MasterSlave/M/CS1
add wave -position end  sim:/MasterSlave/M/CS2
add wave -position end  sim:/MasterSlave/M/CS3
add wave -position end  sim:/MasterSlave/M/MOSI
add wave -position end  sim:/MasterSlave/M/MISO
VSIM 11> run
# Pass maser test1
# Pass slaves test1
VSIM 12> run -continue
run -continue
VSIM 13> run
# Pass maser test2
# Pass slaves test2
VSIM 14> run
# Pass maser test3
# Pass slaves test3
```

The following screenshots shows the communication in the three tests in action throw timing diagrams

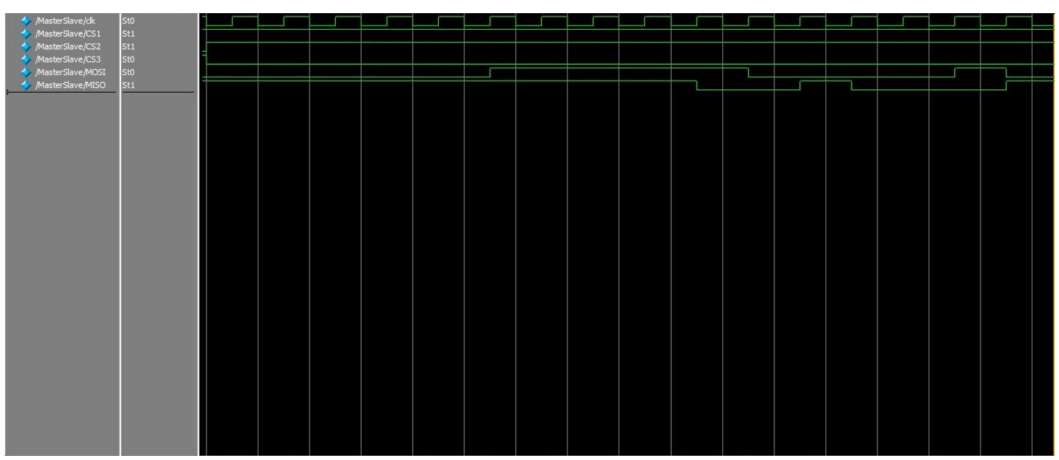
Test 1



Test 2



Test 3



Comparison Between Communication Protocols

Specifications communication:

SPI	I2C	UART
Serial Peripheral Interface	Inter-Integrated Circuit	Universal Asynchronous Receiver/Transmitter
4 wires at least	2 wires	2 wires
Synchronous	Synchronous	Asynchronous
Needs a new wire for each new slave	No needs for a new wire for each new slave	Only two devices are connected.
High Data rate, up to 20 Mbps	Medium data rate, up to 3.4 Mbps	Low data rate, up to 460kbps
Full Duplex	Half Duplex	Full Duplex
One Master	One or more Masters	Only two devices are connected.
Relatively medium hardware complexity	Relatively high hardware complexity	Relatively low hardware complexity

How these protocols work

SPI:

Was explained during the report.

I2C:

The master sends a start signal to every connected slave by switching the SDA line from a high (1) to a low(0) and then switches also the SCL line from high(1) to low(0). Then it sends a 7 or 10 bit address of the slave to all slaves and a (read/write) bit.

The slaves will then compare the received address with their own. If the address matches, the specific slave that has that address returns an ACK bit which switches the SDA line low(0) for one bit. If the address does not match any address, the slaves leave the SDA line high

The master will then send or receive the data frame. After each data frame has been transferred, the receiving device returns another ACK bit to the sender to acknowledge successful transmission.

Then finally the master sends a stop signal to the slave by switching SCL high(1) before switching SDA high(1) to stop the data transmission.

UART:

Once The transmitting and receiving devices are connected, data flows from TX to RX of the receiving device, and as Uart is an asynchronous serial transmission. So, the two devices have different clock speeds, because of that the transmitting device adds start and stop bits that are being transferred, to help the receiving device to know when to start and stop reading data. When the receiving device detects a start bit, it will read the bits at the **baud rate** (device data transmission speed), and it is equal to 115,200 by default. Both devices must operate at about the same baud rate.