Cairo University , Faculty of Engineering **Spring 2021**

Computer Engineering department

Advanced Logic design

Advanced Logic design Project Report

Name	ID	Email	Section	BN
Abdelrahman mohamed salem hassan	9202794	abdelrahman.ibrahim01@eng-st.cu.edu.eg	1	38
Ahmed Fawzy Mohamed Ibrahim	9202153	ahmed.ibrahim011@eng-st.cu.edu.eg	1	7
Ahmed Hany Farouk Tawfik	9202213	ahmed.tawfik011@eng-st.cu.edu.eg	1	10
Youssef Saeed Ibrahim Rabie	9203767	youssef.ibrahim01@eng-st.cu.edu.eg	2	39

Work distribution

Abdelrahman mohamed salem hassan	➔	Master module
Ahmed Fawzy Mohamed Ibrahim	➔	Slave module
Ahmed Hany Farouk Tawfik	➔	Master testbench module
Youssef Saeed Ibrahim Rabie	➔	Slave testbench module

Master Module

As previously said , the master should tell the slave about the data that the master have and not to store the data of the master in the slave , so we will discuss the code of the master module as follows.

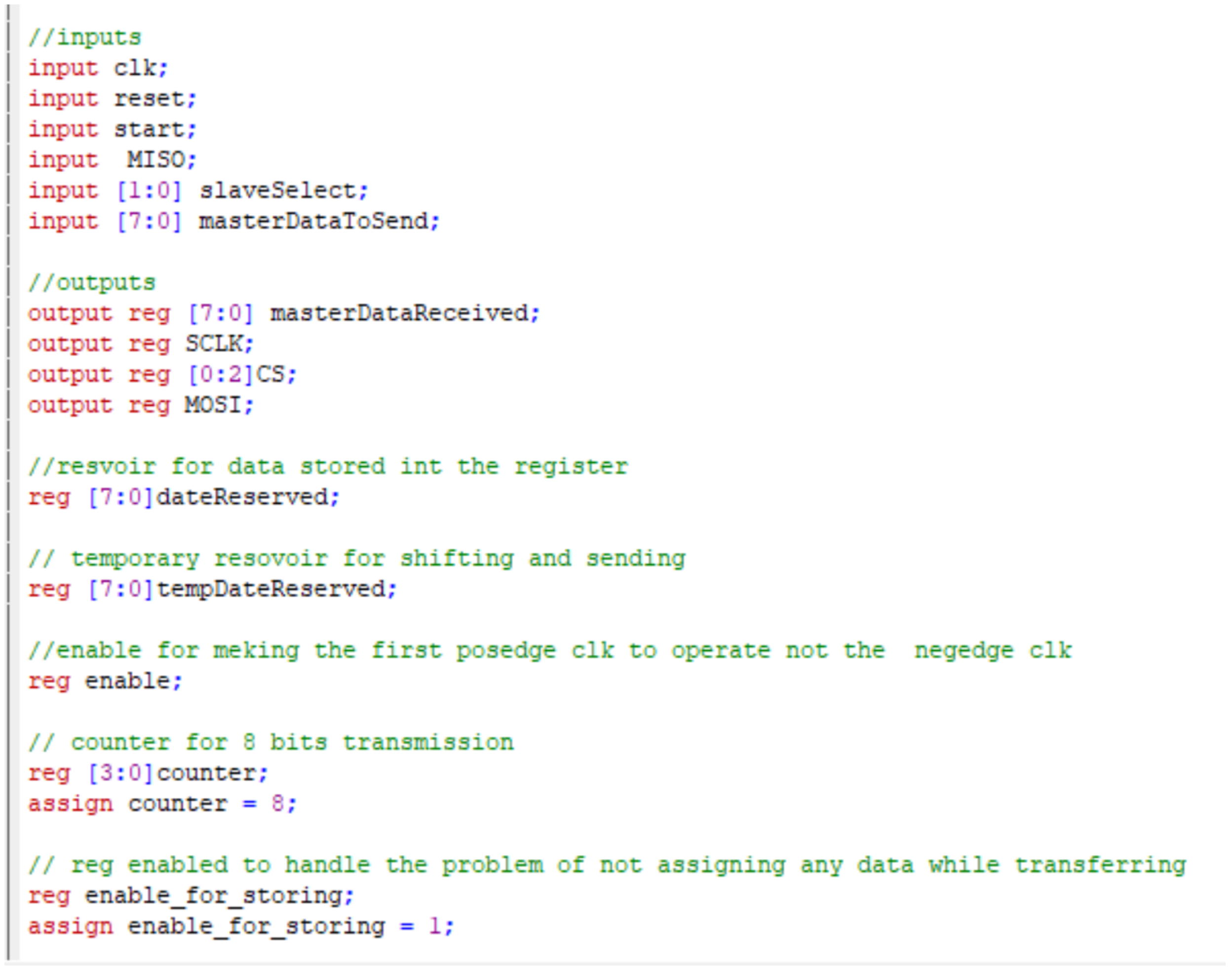
Here we are defining the inputs & outputs for the module , also we are having 4 extra registers, **first one** is **dateReserved** which is needed for storing the value of the data to be stored in the master and it's not changed unless we initialized the module or reset it .

Second one : is **tempDateReserved** as since we need the data to be stored in the module and not be changed , we also need a temporary reg for sending the data from **dateReserved** to the slave and receiving in it by shifting so this reg is always changing.

Third one : is **enable** and we will discuss it below but for now , it's mainly needed to make the module only to start transfer when receiving the first positive edge not the first edge , as if it receives the first edge that maybe negative then the module will store data that not sent yet.

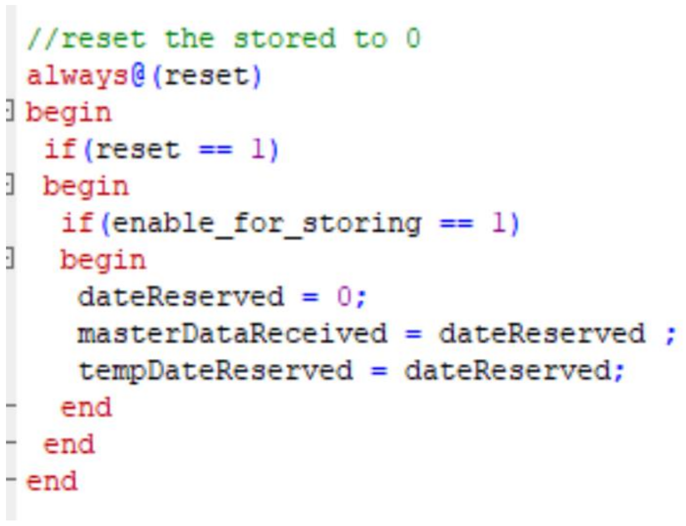
Fourth one : is **counter** that is needed to only make the master to send 8 bits and receive 8 bits only.

Fifth one : is **enable_for_storing** that is enable reg to prevent the initialization of data preserved in master so that the data doesn't corrupt wile transferring.



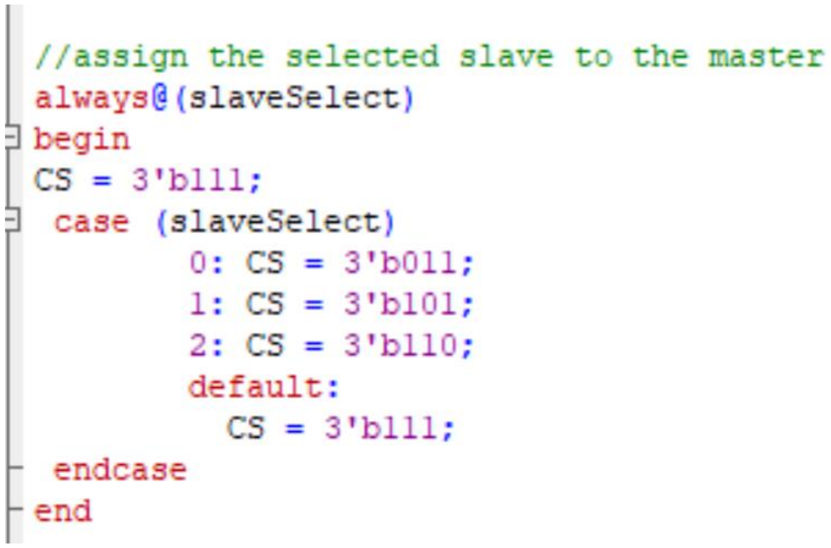
```
//inputs
input clk;
input reset;
input start;
input  MISO;
input [1:0] slaveSelect;
input [7:0] masterDataToSend;

//outputs
output reg [7:0] masterDataReceived;
output reg SCLK;
output reg [0:2]CS;
output reg MOSI;

//resvoir for data stored int the register
reg [7:0]dateReserved;

// temporary resovoir for shifting and sending
reg [7:0]tempDateReserved;

//enable for meking the first posedge clk to operate not the  negedge clk
reg enable;

// counter for 8 bits transmission
reg [3:0]counter;
assign counter = 8;

// reg enabled to handle the problem of not assigning any data while transferring
reg enable_for_storing;
assign enable_for_storing = 1;
```

here the first always block needed for resetting the data stored in the master to make it equal zeros , it will reset the data only at reset = 1 , but it only reset the data if it’s not in the state of transferring the data.



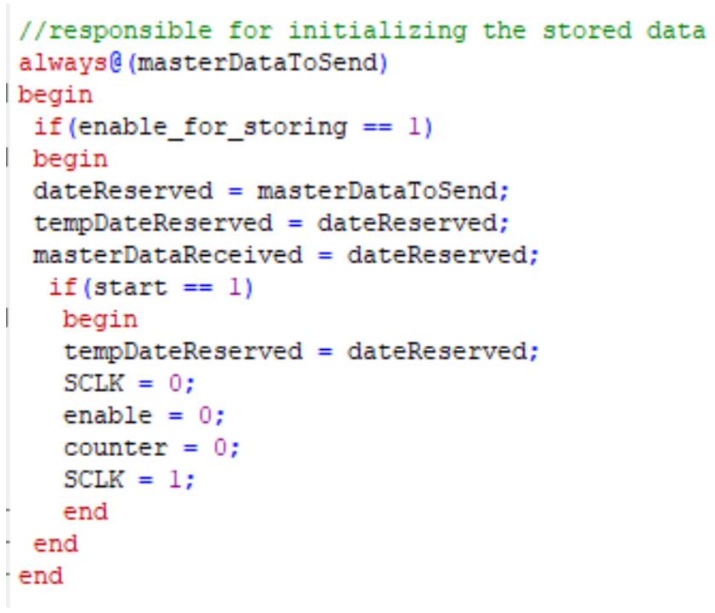
```
//reset the stored to 0
always@(reset)
begin
 if(reset == 1)
 begin
  if(enable_for_storing == 1)
  begin
   dateReserved = 0;
   masterDataReceived = dateReserved ;
   tempDateReserved = dateReserved;
  end
 end
end
```

Here the second always block , it will first set the default of CS to 111 so to make the master not to communicate with the slaves as a default value , only when the **slaveSelect** changes , at that time the CS will changes according to **slaveSelect** so to make the required slave work as the slave the works at Falling edge of CS , so we set the CS of the slave we will work with to 0.



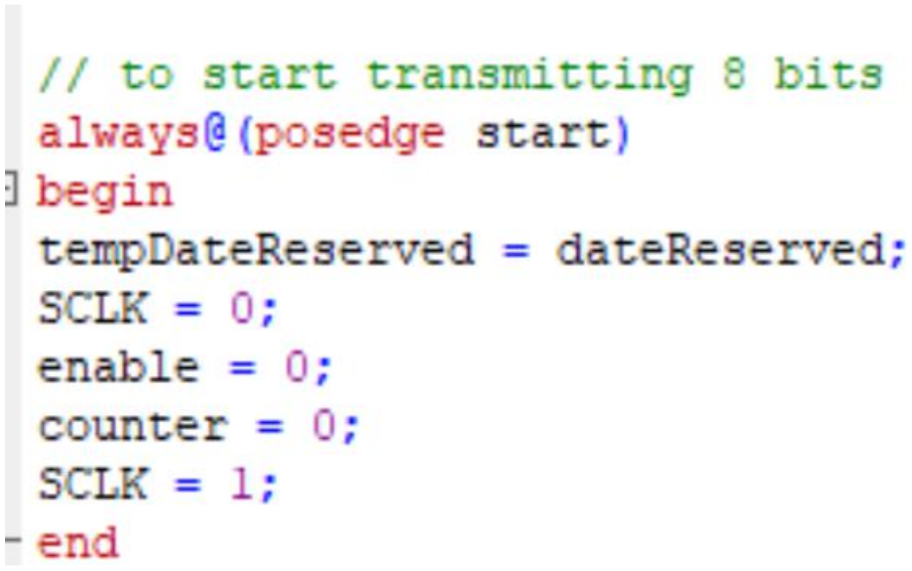
```
//assign the selected slave to the master
always@(slaveSelect)
begin
CS = 3'b111;
 case (slaveSelect)
        0: CS = 3'b011;
        1: CS = 3'b101;
        2: CS = 3'b110;
        default:
          CS = 3'b111;
 endcase
end
```

Here , in the 3rd always block that is mainly for initializing the data stored in the master also the the temporary reg will be assigned to the same value to use it in transferring , but it will only do that in case that he master isn't transferring anything to prevent data corruption . also if the master wasn't transferring anything then a new was initialized to the master then the master will start a new transfer operation if only the start was one to tell the slave that his data changed.



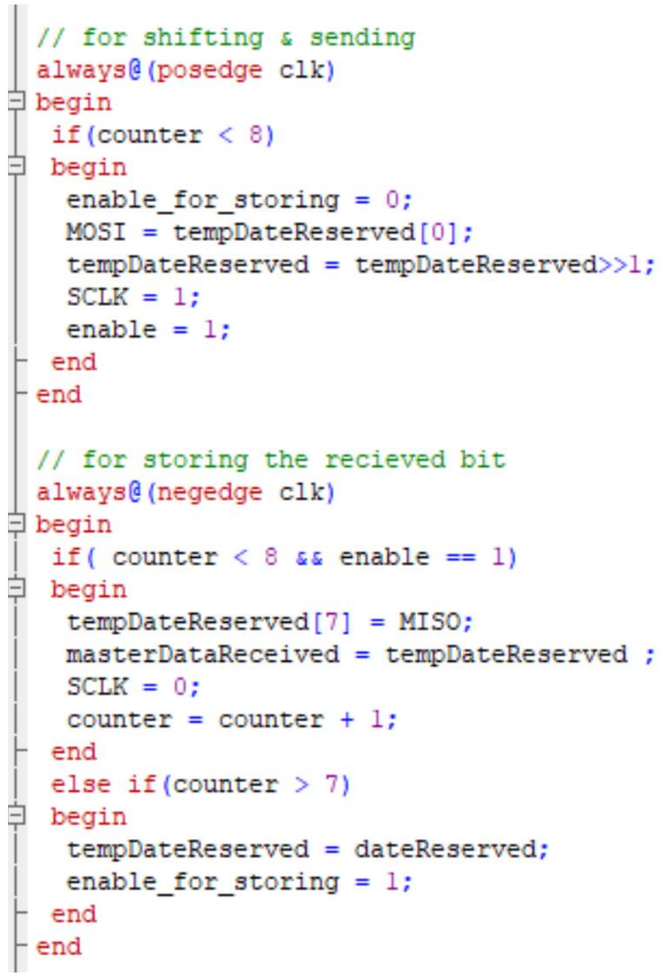
```
//responsible for initializing the stored data
always@(masterDataToSend)
begin
 if(enable_for_storing == 1)
 begin
 dateReserved = masterDataToSend;
 tempDateReserved = dateReserved;
 masterDataReceived = dateReserved;
  if(start == 1)
   begin
   tempDateReserved = dateReserved;
   SCLK = 0;
   enable = 0;
   counter = 0;
   SCLK = 1;
   end
 end
end
```

Here is the 6th always block for the posedge start , as when the start becomes 1 , we need to transfer 8 bits , so we set the counter to be 0 , so the module interact with the **clk** , then we set the enable to be 0 , the enable functionality is discussed below , as it's set to be 1 at the first positive edge of **clk** to avoid assigning before receiving , also we assign the temporary reg to the stored data so that we can send & receive the data from and to slave , also we enhance the slave to start it's posedge **SCLK** block , in other words the save will send its data first but it will not differ whether one will send his data first.



```
// to start transmitting 8 bits
always@(posedge start)
begin
tempDateReserved = dateReserved;
SCLK = 0;
enable = 0;
counter = 0;
SCLK = 1;
end
```

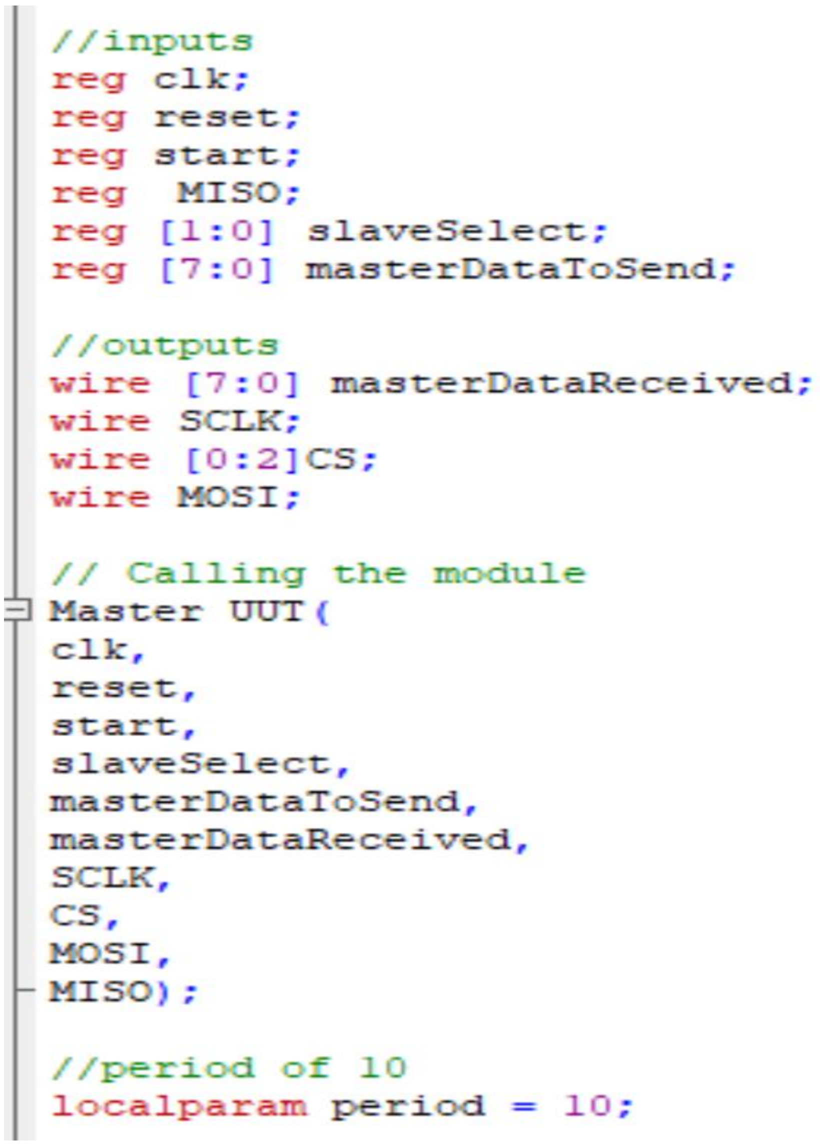
In the 4th & 5th always blocks , these two blocks are responsible for sending to slave & receiving from it as we will see , the initial value of the counter will be x , and it will not be zero except only at the 6th always block for **start** , so here we are receiving continuously positive edges of **clk** , so when we know when to start transmitting , the answer is when the **counter** is < 8 and when the **counter** value to be initialized , the answer is : in the always block of the **start** , so when the counter becomes zero we send the data from the master and shift the temporary reg that is responsible for sending & receiving then we set **SCLK** to be 1 to enforce the positive edge of it so that the slave sends us 1 bit from its data as **SCLK & clk** must be Synchronized and also we set **enable** to be one so that we can receive the next falling edge as we must receive the 1st positive rising edge not the first negative edge as we need to send the value before starting it and that's why the if statement for posedge clk differs a little bit from the negedge of the clk as it must come second after the posedge , in the 5th always block , what we do is to store the value sent from the slave to be stored in the temporary reg and the enforce the negedge for slave to do the same , and after we finished sending and storing , we increment the counter as indication of success of 1 bit transfer then after finishing transmission of transferring 8 bits then the counter become 8 (> 7) so we reset the temporary reg to be the value of data reserved in the master and set **enable_for_storing** to be 1 so to make the master able to reset his data or even changes it . also in the positive edge clk we make **enable_for_storing** to be 0 to prevent changing the data stored in the master while transmission to prevent data corruption.



```
// for shifting & sending
always@(posedge clk)
begin
 if(counter < 8)
 begin
  enable_for_storing = 0;
  MOSI = tempDateReserved[0];
  tempDateReserved = tempDateReserved>>1;
  SCLK = 1;
  enable = 1;
 end
end

// for storing the recieved bit
always@(negedge clk)
begin
 if( counter < 8 && enable == 1)
 begin
  tempDateReserved[7] = MISO;
  masterDataReceived = tempDateReserved ;
  SCLK = 0;
  counter = counter + 1;
 end
 else if(counter > 7)
 begin
  tempDateReserved = dateReserved;
  enable_for_storing = 1;
 end
end
```

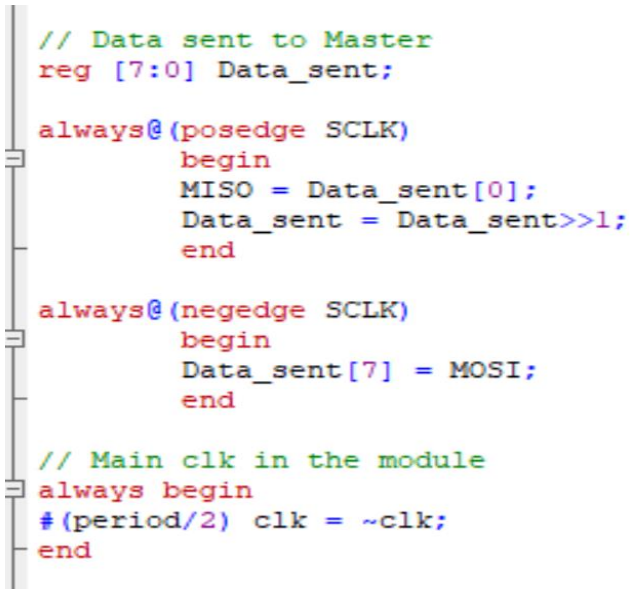
Master TestBench Module

The Master test bench works as follows. First, declaring registers which are the input of the module (clk, reset, start, MISO, slave select, Master Data to send) and declare the outputs as wires which are (master, SCLK, CS, MOSI) to detect the output. Then it calls the master by name UUT giving it the data we declared in the same order it was declared in the module. A local parameter of name period was declared, and its value is 10 picoseconds to facilitate dealing with delay.



```
//inputs
reg clk;
reg reset;
reg start;
reg  MISO;
reg [1:0] slaveSelect;
reg [7:0] masterDataToSend;

//outputs
wire [7:0] masterDataReceived;
wire SCLK;
wire [0:2]CS;
wire MOSI;

// Calling the module
Master UUT(
clk,
reset,
start,
slaveSelect,
masterDataToSend,
masterDataReceived,
SCLK,
CS,
MOSI,
MISO);

//period of 10
localparam period = 10;
```

There is always block where the CLK changes every (period /2). There is two always blocks one for the positive edge of the SCLK and another for the negative edge of the SCLK. The SCLK is a clock which will be equal to CLK when start signal is given and its responsible for shifting of data in the positive edge and sampling (saving) in the negative edge.

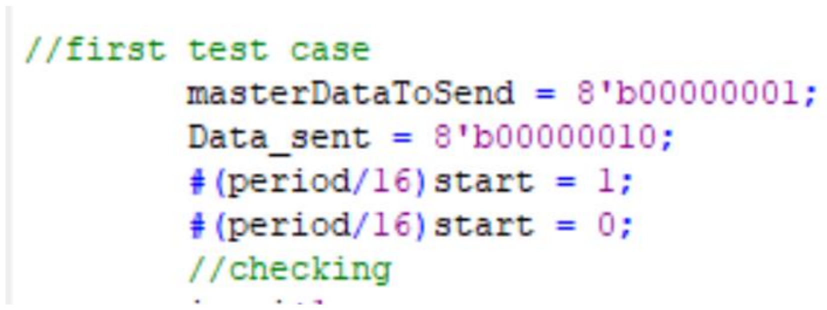


```
// Data sent to Master
reg [7:0] Data_sent;

always@(posedge SCLK)
        begin
        MISO = Data_sent[0];
        Data_sent = Data_sent>>1;
        end

always@(negedge SCLK)
        begin
        Data_sent[7] = MOSI;
        end

// Main clk in the module
always begin
#(period/2) clk = ~clk;
end
```

For testing each case as the reg **Data_sent** is used for sending & receiving from the master and then manually test the data sent to the master with the expected and same for the data sent from the master , also giving a delay of (period*9) in each test case to give the module a chance of sending & receiving the data.

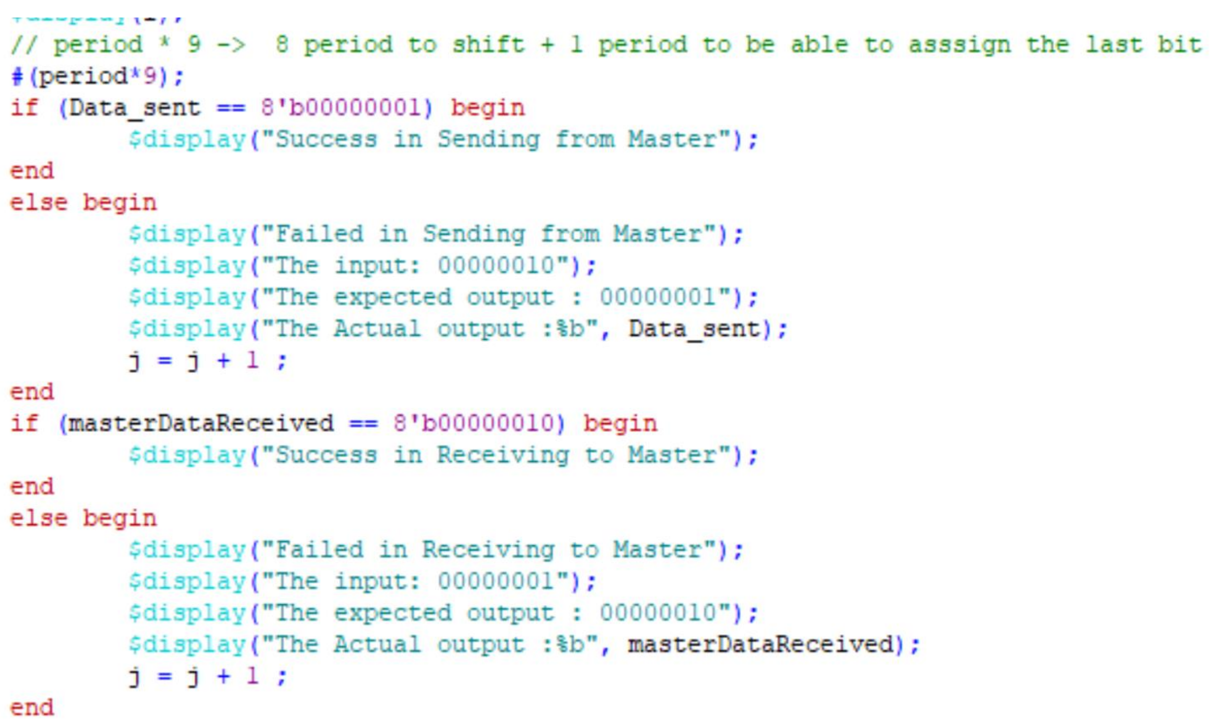
Also before each sending , we assign the data to be sent to sent to the master.

And giving a delay to the **start** to give it a chance of starting the transfer operation.

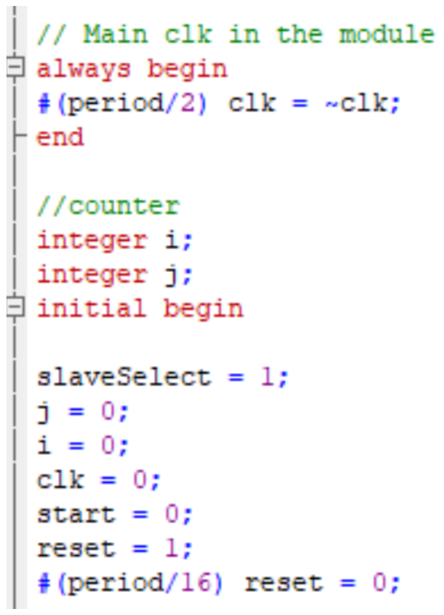


```
//first test case
        masterDataToSend = 8'b00000001;
        Data_sent = 8'b00000010;
        #(period/16)start = 1;
        #(period/16)start = 0;
        //checking
```

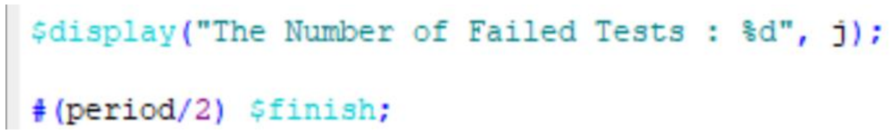
The initial block starts with initializing the slave select and the start then resting. After that A 10 test cases is chosen to try the Master module. In the case of success it prints the order of the test case then success in sending and success in receiving. But in the case of failing it print the expected output, input, the actual data and it prints the number of test cases that failed in the test bench.



```
// period * 9 ->  8 period to shift + 1 period to be able to asssign the last bit
#(period*9);
if (Data_sent == 8'b00000001) begin
        $display("Success in Sending from Master");
end
else begin
        $display("Failed in Sending from Master");
        $display("The input: 00000010");
        $display("The expected output : 00000001");
        $display("The Actual output :%b", Data_sent);
        j = j + 1 ;
end
if (masterDataReceived == 8'b00000010) begin
        $display("Success in Receiving to Master");
end
else begin
        $display("Failed in Receiving to Master");
        $display("The input: 00000001");
        $display("The expected output : 00000010");
        $display("The Actual output :%b", masterDataReceived);
        j = j + 1 ;
end
```

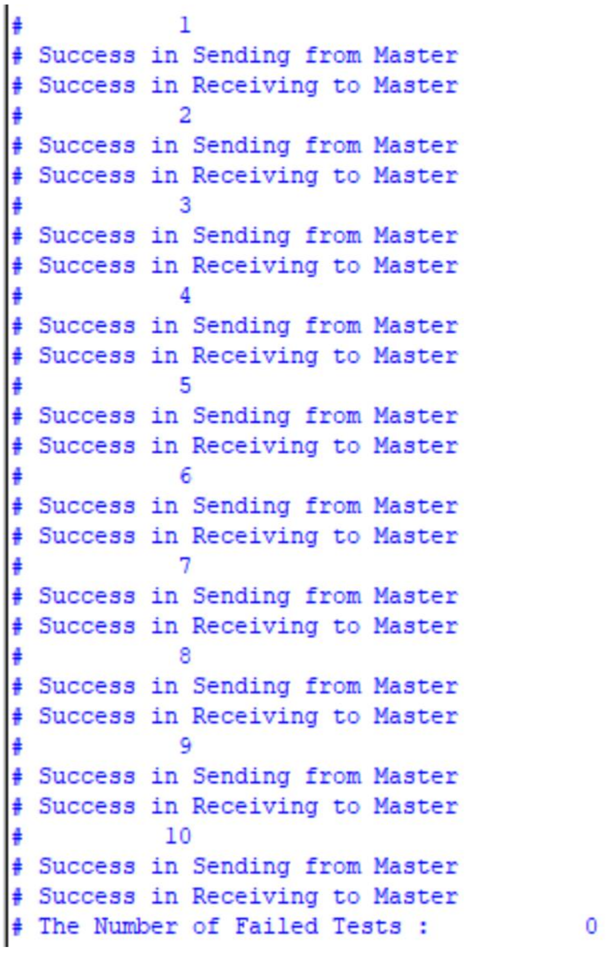


```
// Main clk in the module
always begin
#(period/2) clk = ~clk;
end

//counter
integer i;
integer j;
initial begin

slaveSelect = 1;
j = 0;
i = 0;
clk = 0;
start = 0;
reset = 1;
#(period/16) reset = 0;
```

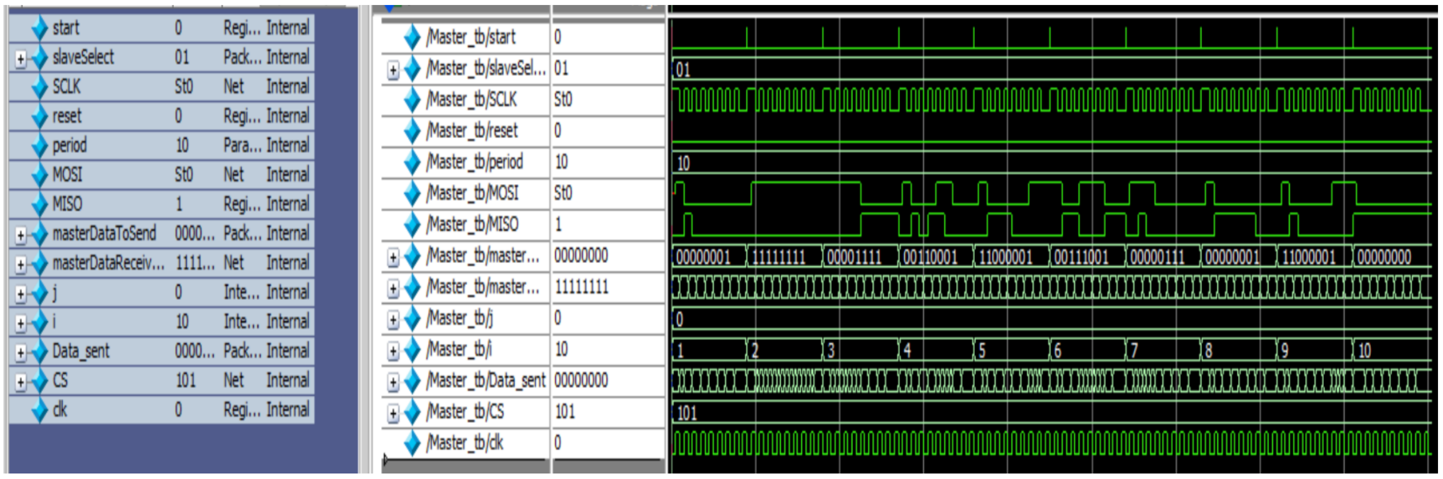


```
$display("The Number of Failed Tests : %d", j);

#(period/2) $finish;
```

This is the screen for the output & wave for the test cases written in the master testbench module.



```
#           1
# Success in Sending from Master
# Success in Receiving to Master
#           2
# Success in Sending from Master
# Success in Receiving to Master
#           3
# Success in Sending from Master
# Success in Receiving to Master
#           4
# Success in Sending from Master
# Success in Receiving to Master
#           5
# Success in Sending from Master
# Success in Receiving to Master
#           6
# Success in Sending from Master
# Success in Receiving to Master
#           7
# Success in Sending from Master
# Success in Receiving to Master
#           8
# Success in Sending from Master
# Success in Receiving to Master
#           9
# Success in Sending from Master
# Success in Receiving to Master
#          10
# Success in Sending from Master
# Success in Receiving to Master
# The Number of Failed Tests :           0
```



start	0	Regi...	Internal
slaveSelect	01	Pack...	Internal
SCLK	St0	Net	Internal
reset	0	Regi...	Internal
period	10	Para...	Internal
MOSI	St0	Net	Internal
MISO	1	Regi...	Internal
masterDataToSend	0000...	Pack...	Internal
masterDataReceiv...	1111...	Net	Internal
j	0	Inte...	Internal
i	10	Inte...	Internal
Data_sent	0000...	Pack...	Internal
CS	101	Net	Internal
clk	0	Regi...	Internal

/Master_tb/start	0
/Master_tb/slaveSel...	01
/Master_tb/SCLK	St0
/Master_tb/reset	0
/Master_tb/period	10
/Master_tb/MOSI	St0
/Master_tb/MISO	1
/Master_tb/master...	00000000
/Master_tb/master...	11111111
/Master_tb/j	0
/Master_tb/i	10
/Master_tb/Data_sent	00000000
/Master_tb/CS	101
/Master_tb/clk	0

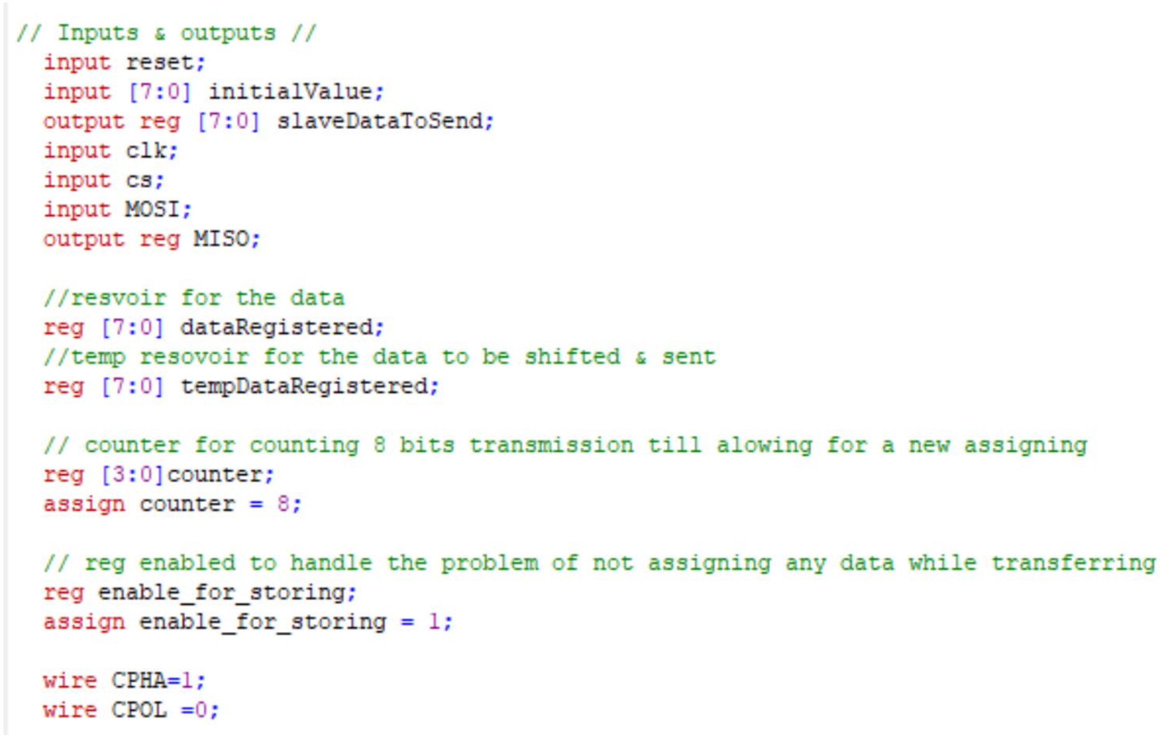
Slave Module

Mainly, Slave works only in mode1 (CPAHA=1 &CPOL=0) in which data is shifted at rising edge and sampled at falling edge.

First we have 2 reg , one for storing the data of slave and not to change except only when initializing the slave with a value and the other reg is temporary reg for sending and receiving the 8 bits from & to master.

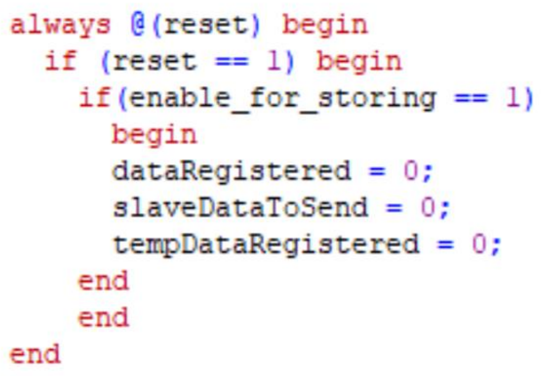
Second : enable_for_storing that is enable reg to prevent the initialization of data preserved in slave so that the data doesn't corrupt wile transferring.

Third : we have a counter to count to know that we transferred 8 bits to make sure that we completed transmission so that we can change data in the slave



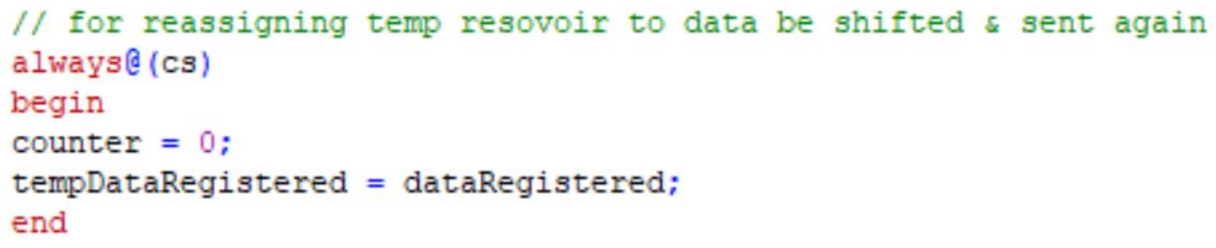
```
// Inputs & outputs //
  input reset;
  input [7:0] initialValue;
  output reg [7:0] slaveDataToSend;
  input clk;
  input cs;
  input MOSI;
  output reg MISO;

  //resvoir for the data
  reg [7:0] dataRegistered;
  //temp resovoir for the data to be shifted & sent
  reg [7:0] tempDataRegistered;

  // counter for counting 8 bits transmission till alowing for a new assigning
  reg [3:0]counter;
  assign counter = 8;

  // reg enabled to handle the problem of not assigning any data while transferring
  reg enable_for_storing;
  assign enable_for_storing = 1;

  wire CPHA=1;
  wire CPOL =0;
```

Initially at reset, if reset is equal to one the data of slave is cleared and the data registered in slave and temporary data registered are also cleared , it will reset the data only at reset = 1 , but it only reset the data if it's not in the state of transferring the data..



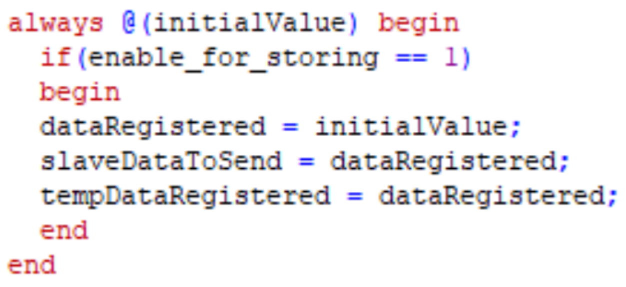
```
always @(reset) begin
  if (reset == 1) begin
    if(enable_for_storing == 1)
      begin
      dataRegistered = 0;
      slaveDataToSend = 0;
      tempDataRegistered = 0;
    end
    end
end
```

Also when changing the CS of the master to operate , there is always block for CS to assign the temporary reg to the value of the data of slave so as to make the slave ready for sending & receiving , also it resets the counter as indication of starting a new transmission process.



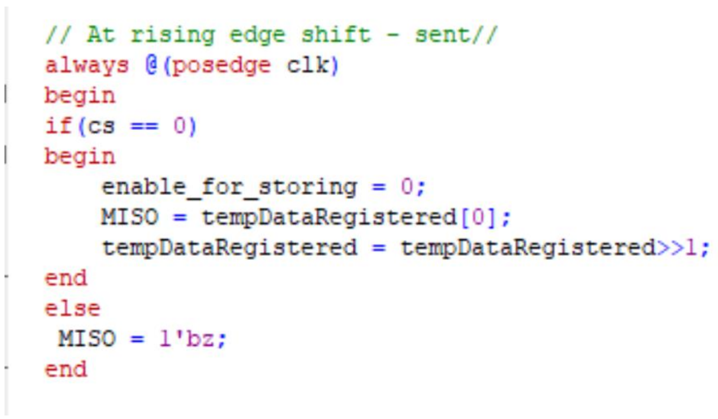
```
// for reassigning temp resovoir to data be shifted & sent again
always@(cs)
begin
counter = 0;
tempDataRegistered = dataRegistered;
end
```

Then, at initial value of slave the data registered will equal the input initial value of slave and it will be registered in temporary data registered too but only and only if the slave isn't at the process of transmitting data to prevent data corruption.



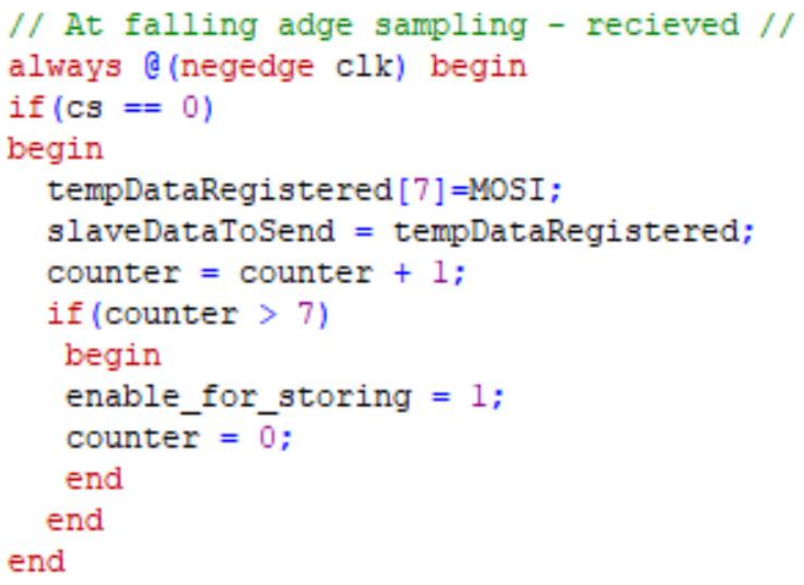
```
always @(initialValue) begin
  if(enable_for_storing == 1)
  begin
  dataRegistered = initialValue;
  slaveDataToSend = dataRegistered;
  tempDataRegistered = dataRegistered;
  end
end
```

At every positive edge clock shifting occurs if the chip selector of the salve is equal to zero and the "MISO" will equal the least significant bit of temporary data registered and the temporary data registered will be shifted right by one. If the chip selector of slave is one the data registered will equal z so as not the slave sends don't care (x) as if the slave not working and sent x then the x will override the value sent from the working slave so the master will receive x instead of 1 or 0 from the working slave and store the x instead of the bit sent , that's why we send z (high impledance) so to force the master to receive from the working slave and it also it makes the slave not to change his data till the completion of transmission process by making : enable_for_storing equal to 0 to prevent data corruption .



```
// At rising edge shift - sent//
always @(posedge clk)
begin
if(cs == 0)
begin
    enable_for_storing = 0;
    MISO = tempDataRegistered[0];
    tempDataRegistered = tempDataRegistered>>1;
end
else
 MISO = 1'bz;
end
```

At every negative edge clock sampling occurs if the enable of salve is equal to zero and the most significant bit of the temporary data registered will equal the “MOSI” and the data output from slave will be equal to the temporary data registered also it increment the counter by one as indication of 1 bit successful transmission then it checks if counter has become 8 (> 7) as indication of successful of 8 bits transfer and if it was true then it will reset the counter to be ready for the next 8 bits transformation and it makes the slave can change his as the slave completed transfer of 8 bits .



```
// At falling adge sampling - recieved //
always @(negedge clk) begin
if(cs == 0)
begin
  tempDataRegistered[7]=MOSI;
  slaveDataToSend = tempDataRegistered;
  counter = counter + 1;
  if(counter > 7)
   begin
   enable_for_storing = 1;
   counter = 0;
   end
  end
end
```

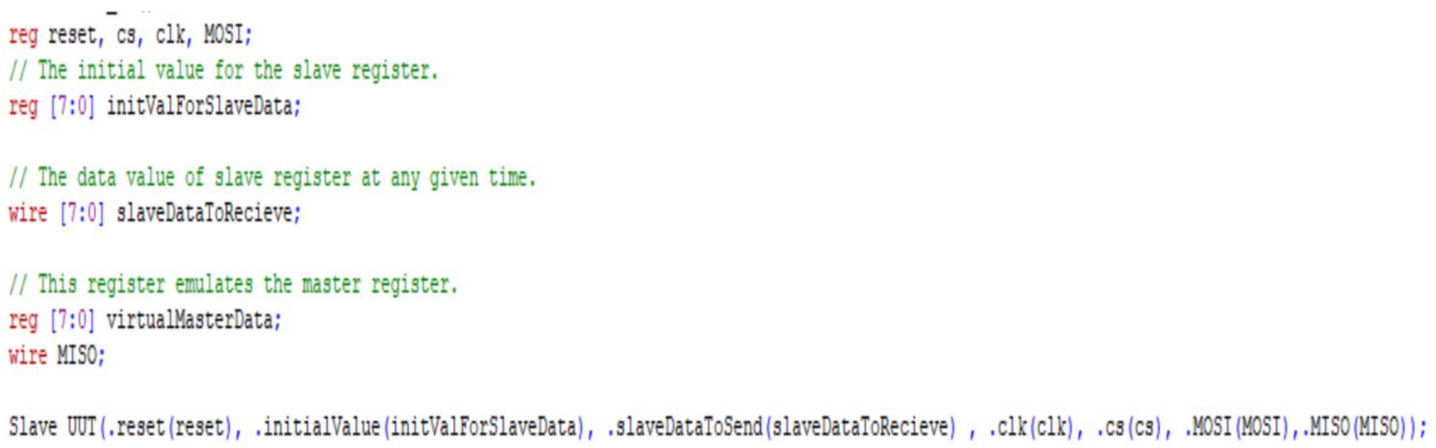
Slave TestBench Module

Slave test bench was designed with the following:

A slave module

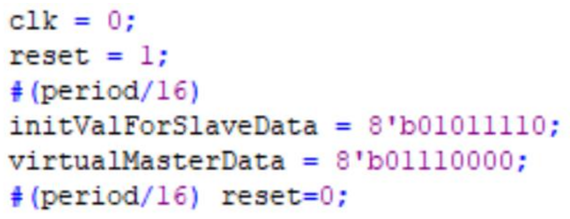
A register emulating the master data register.

Two counters for failed sending/receiving test cases.



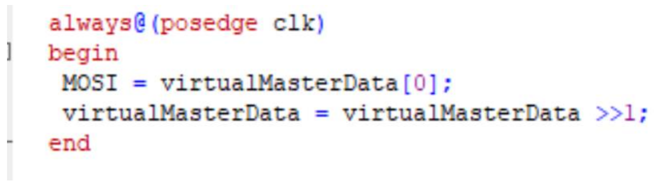
```
reg reset, cs, clk, MOSI;
// The initial value for the slave register.
reg [7:0] initValForSlaveData;

// The data value of slave register at any given time.
wire [7:0] slaveDataToRecieve;

// This register emulates the master register.
reg [7:0] virtualMasterData;
wire MISO;

Slave UUT(.reset(reset), .initialValue(initValForSlaveData), .slaveDataToSend(slaveDataToRecieve) , .clk(clk), .cs(cs), .MOSI(MOSI),.MISO(MISO));
```

10 test cases were written into it, between each one a reset happened then the setting of the slave data and the master data.



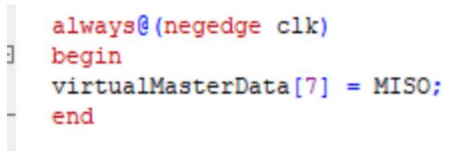
```
clk = 0;
reset = 1;
#(period/16)
initValForSlaveData = 8'b01011110;
virtualMasterData = 8'b01110000;
#(period/16) reset=0;
```

On the positive edge of the clock, we shifted the virtualMasterData register and took the LSB into MOSI.



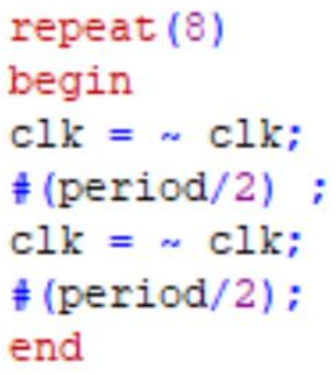
```
always@(posedge clk)
begin
 MOSI = virtualMasterData[0];
 virtualMasterData = virtualMasterData >>1;
end
```

On the negative edge, we set the MSB of virtualMasterData with the value of MISO.



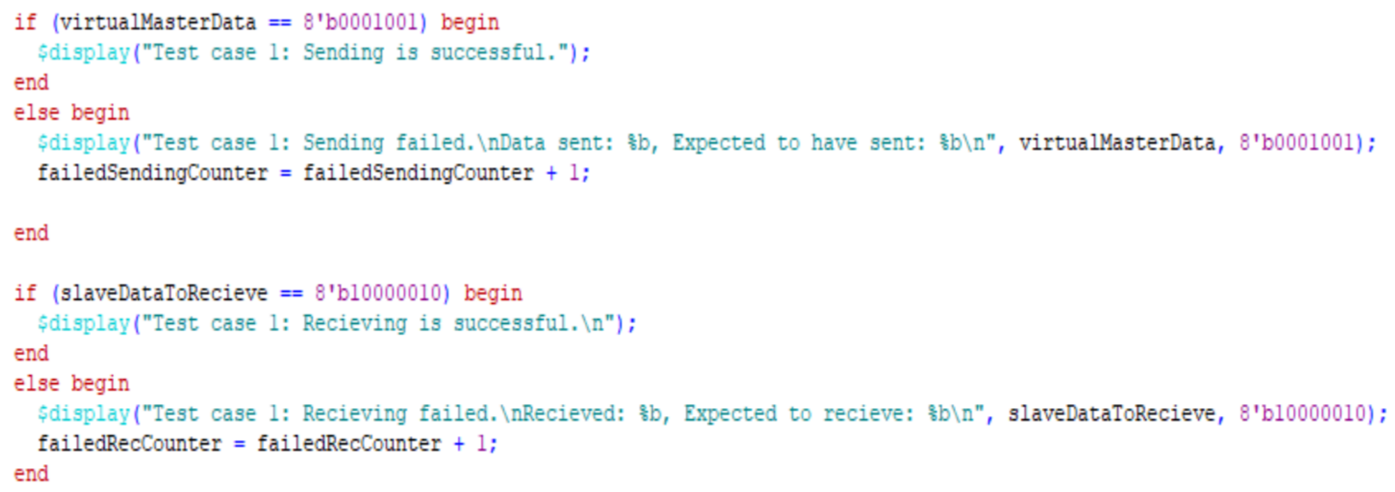
```
always@(negedge clk)
begin
virtualMasterData[7] = MISO;
end
```

In this block we are simulating as a master sending 8 clock pulses to the slave so that the slave could store and send the data and this block is done before checking in every test case.



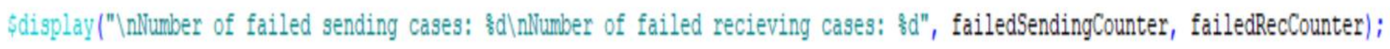
```
repeat(8)
begin
clk = ~ clk;
#(period/2) ;
clk = ~ clk;
#(period/2);
end
```

We checked for failure in sending and in receiving separately in each test cases, and printed what is expected and what was actually generated in case of failure.

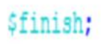


```
if (virtualMasterData == 8'b0001001) begin
  $display("Test case 1: Sending is successful.");
end
else begin
  $display("Test case 1: Sending failed.\nData sent: %b, Expected to have sent: %b\n", virtualMasterData, 8'b0001001);
  failedSendingCounter = failedSendingCounter + 1;

end

if (slaveDataToRecieve == 8'b10000010) begin
  $display("Test case 1: Recieving is successful.\n");
end
else begin
  $display("Test case 1: Recieving failed.\nRecieved: %b, Expected to recieve: %b\n", slaveDataToRecieve, 8'b10000010);
  failedRecCounter = failedRecCounter + 1;
end
```

At the end, the count of failed sending/receiving cases was printed.

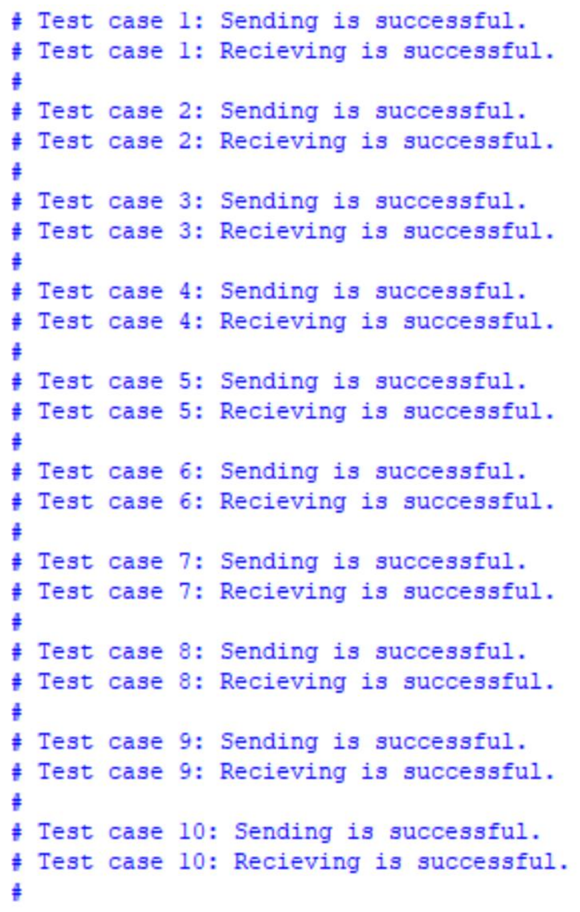


```
$display("\nNumber of failed sending cases: %d\nNumber of failed recieving cases: %d", failedSendingCounter, failedRecCounter);
```

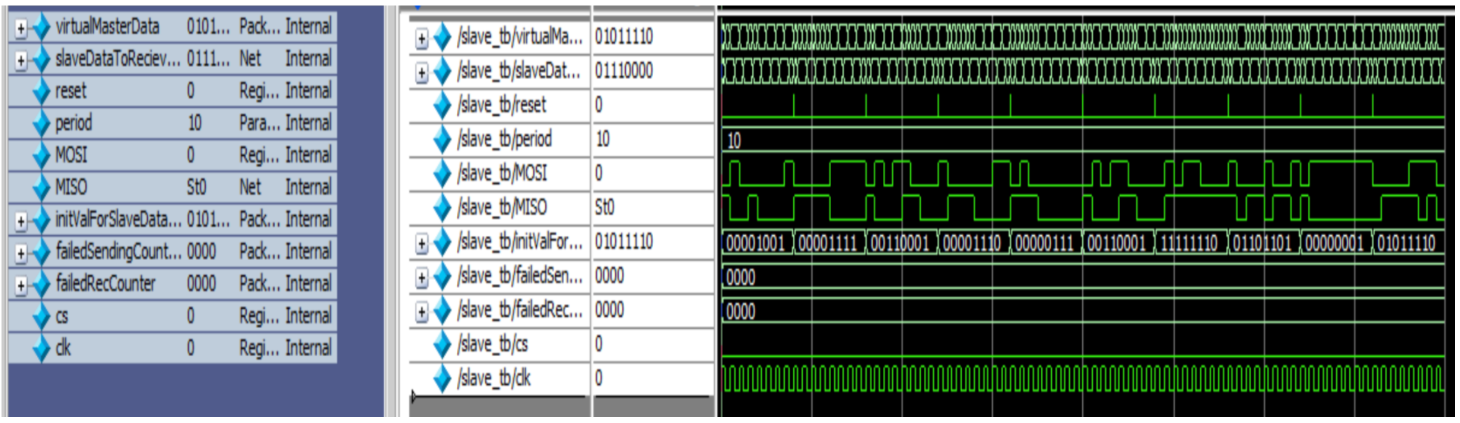


```
$finish;
```

This is the screen for the output & wave for the test cases written in the slave testbench module



```
# Test case 1: Sending is successful.
# Test case 1: Recieving is successful.
#
# Test case 2: Sending is successful.
# Test case 2: Recieving is successful.
#
# Test case 3: Sending is successful.
# Test case 3: Recieving is successful.
#
# Test case 4: Sending is successful.
# Test case 4: Recieving is successful.
#
# Test case 5: Sending is successful.
# Test case 5: Recieving is successful.
#
# Test case 6: Sending is successful.
# Test case 6: Recieving is successful.
#
# Test case 7: Sending is successful.
# Test case 7: Recieving is successful.
#
# Test case 8: Sending is successful.
# Test case 8: Recieving is successful.
#
# Test case 9: Sending is successful.
# Test case 9: Recieving is successful.
#
# Test case 10: Sending is successful.
# Test case 10: Recieving is successful.
#
```

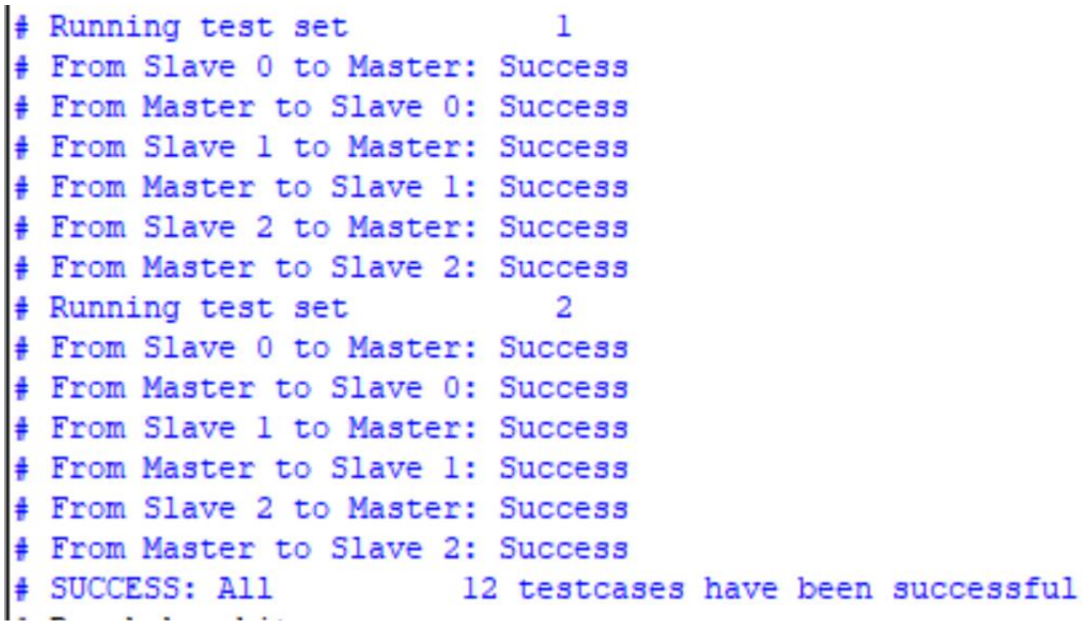


virtualMasterData	0101...	Pack...	Internal
slaveDataToReciev...	0111...	Net	Internal
reset	0	Regi...	Internal
period	10	Para...	Internal
MOSI	0	Regi...	Internal
MISO	St0	Net	Internal
initValForSlaveData...	0101...	Pack...	Internal
failedSendingCount...	0000	Pack...	Internal
failedRecCounter	0000	Pack...	Internal
cs	0	Regi...	Internal
clk	0	Regi...	Internal

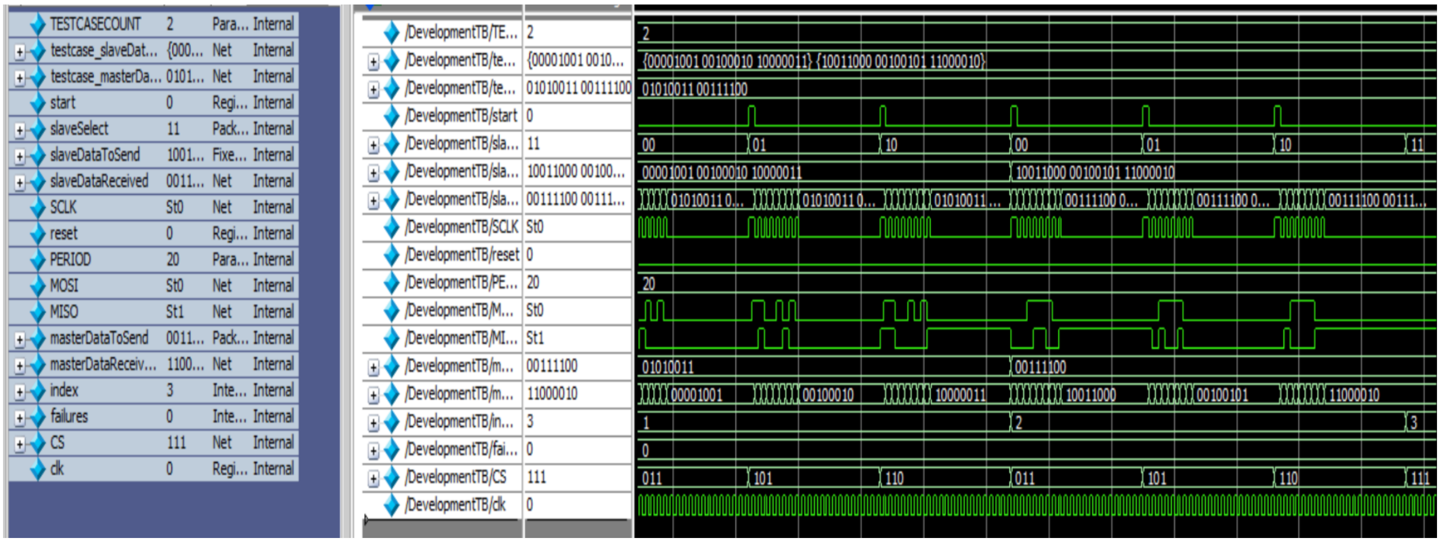
/slave_tb/virtualMa...	01011110
/slave_tb/slaveDat...	01110000
/slave_tb/reset	0
/slave_tb/period	10
/slave_tb/MOSI	0
/slave_tb/MISO	St0
/slave_tb/initValFor...	01011110
/slave_tb/failedSen...	0000
/slave_tb/failedRec...	0000
/slave_tb/cs	0
/slave_tb/clk	0

<u>Last but not least</u>

Here is output transcript & wave of the testbench that was sent to us(development TB) .



```
# Running test set           1
# From Slave 0 to Master: Success
# From Master to Slave 0: Success
# From Slave 1 to Master: Success
# From Master to Slave 1: Success
# From Slave 2 to Master: Success
# From Master to Slave 2: Success
# Running test set           2
# From Slave 0 to Master: Success
# From Master to Slave 0: Success
# From Slave 1 to Master: Success
# From Master to Slave 1: Success
# From Slave 2 to Master: Success
# From Master to Slave 2: Success
# SUCCESS: All          12 testcases have been successful
```



TESTCASECOUNT	2	Para...	Internal
testcase_slaveDat...	{000...	Net	Internal
testcase_masterDa...	0101...	Net	Internal
start	0	Regi...	Internal
slaveSelect	11	Pack...	Internal
slaveDataToSend	1001...	Fixe...	Internal
slaveDataReceived	0011...	Net	Internal
SCLK	St0	Net	Internal
reset	0	Regi...	Internal
PERIOD	20	Para...	Internal
MOSI	St0	Net	Internal
MISO	St1	Net	Internal
masterDataToSend	0011...	Pack...	Internal
masterDataReceiv...	1100...	Net	Internal
index	3	Inte...	Internal
failures	0	Inte...	Internal
CS	111	Net	Internal
clk	0	Regi...	Internal

/DevelopmentTB/TE...	2
/DevelopmentTB/te...	{00001001 0010...
/DevelopmentTB/te...	01010011 00111100
/DevelopmentTB/start	0
/DevelopmentTB/sla...	11
/DevelopmentTB/sla...	10011000 00100...
/DevelopmentTB/sla...	00111100 00111...
/DevelopmentTB/SCLK	St0
/DevelopmentTB/reset	0
/DevelopmentTB/PE...	20
/DevelopmentTB/M...	St0
/DevelopmentTB/MI...	St1
/DevelopmentTB/m...	00111100
/DevelopmentTB/m...	11000010
/DevelopmentTB/in...	3
/DevelopmentTB/fai...	0
/DevelopmentTB/CS	111
/DevelopmentTB/clk	0