Cairo University, Faculty of Engineering
Computer Engineering Department
Advanced Logic Design
CMP1030

SPI

(serial Peripheral Interface)

CMP1030 project 2021 _ Team #3

Member Name	ID	Sec	BN
أحمد صبري عبد الراضي أحمد	9202119	1	4
سماء حازم محمد عبد اللطيف	9202660	1	32
محمود رضا سيد عبدالعزيز	9203399	2	20
ممدوح احمد محمد عطية	9203543	2	26

Table of contents

The design process of each module & simulation results

Master Module

•Parameters

a) Input :

i) Clk (1 bit).

ii) Reset (1 bit).

iii) Start (1 bit).

iv) MISO (1 bit).

v) slaveSelect (2 bits).

vi) masterDataToSend (8 bits).

b) Output :

i) SCLK (1 bit).

ii) CS (3 bits).

iii) MOSI (1 bit).

iv) masterDataReceived (8 bits).

•Master Design

- **How does Master work ?**

- Master module is built using “**Counter Ciruit**”, which acts as:

1) Index to Data Sent & Data Received to determine which bit will be sent or received.

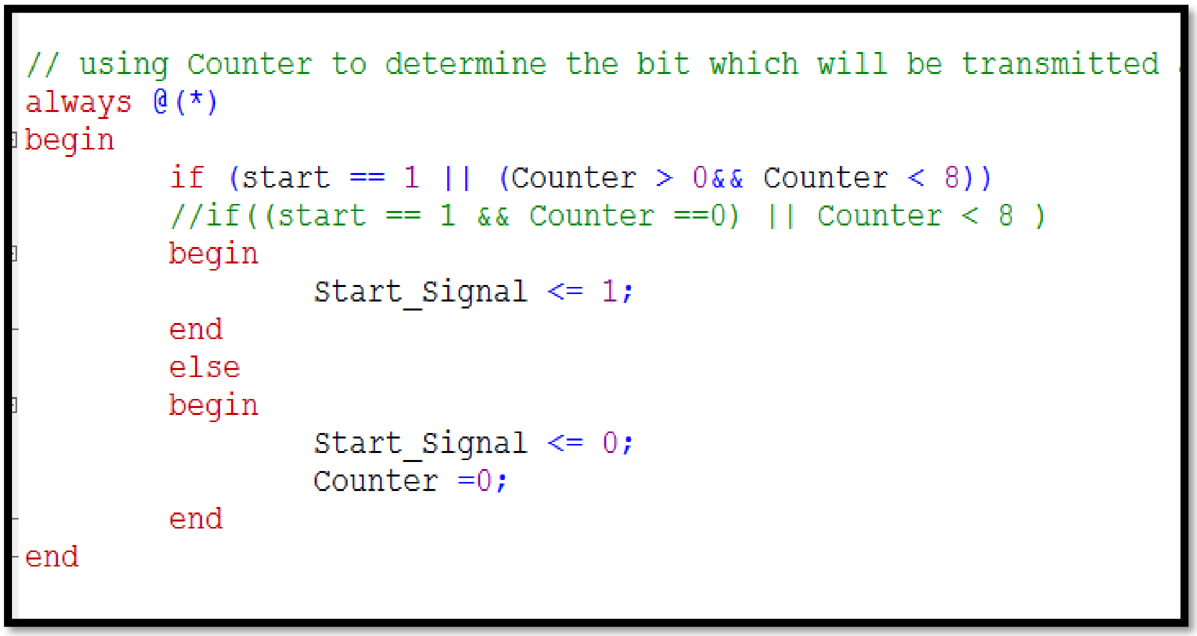
2) Stopping condition to Data Transmission process, After **8 SCLK Cycles**, Counter will be **8** then will terminate Transmission of Data.

How does Master generate outputs?

- Master module generates 4 outputs using Counter and Inputs according to the following Logic:

  1) **Start_Signal** (internal reg)

     Using **Start** signal (input), we generate Start_Signal that **Store** signal & **Begin** Data Transmission until all Data are shifted or sampled then **Stop** Transmission.



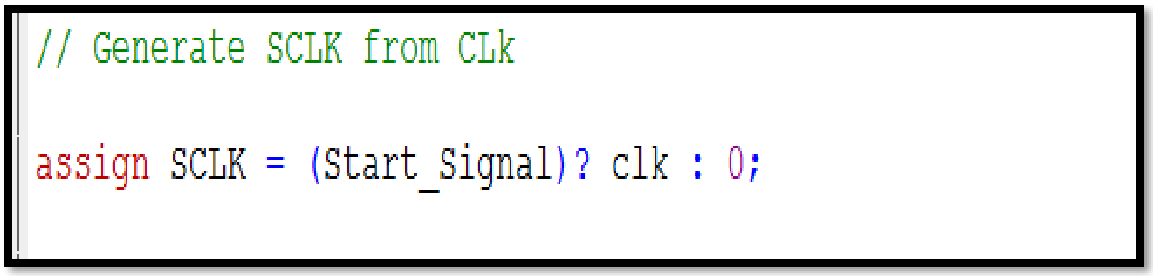
```
// using Counter to determine the bit which will be transmitted
always @(*)
begin
        if (start == 1 || (Counter > 0&& Counter < 8))
        //if((start == 1 && Counter ==0) || Counter < 8 )
        begin
                Start_Signal <= 1;
        end
        else
        begin
                Start_Signal <= 0;
                Counter =0;
        end
end
```

2) **SCLK**

Using internal signal Start_Signal , We Generate **SCLK**,

When Start_Signal is High “**1**”, **SCLK** synchronized with **CLK**

When Start_Signal is Low “**0**”, **SCLK** is off “**0**”

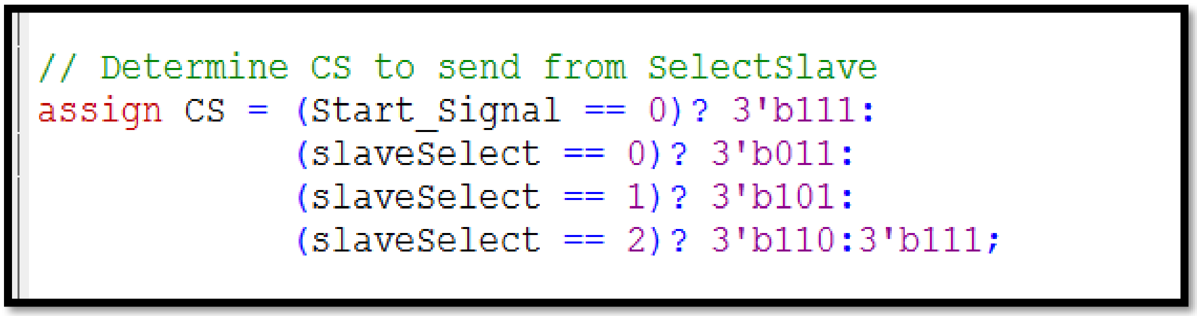


```
// Generate SCLK from CLk

assign SCLK = (Start_Signal)? clk : 0;
```

3) **CS**

Using **selectSlave & Start_Signal** input , we can determine

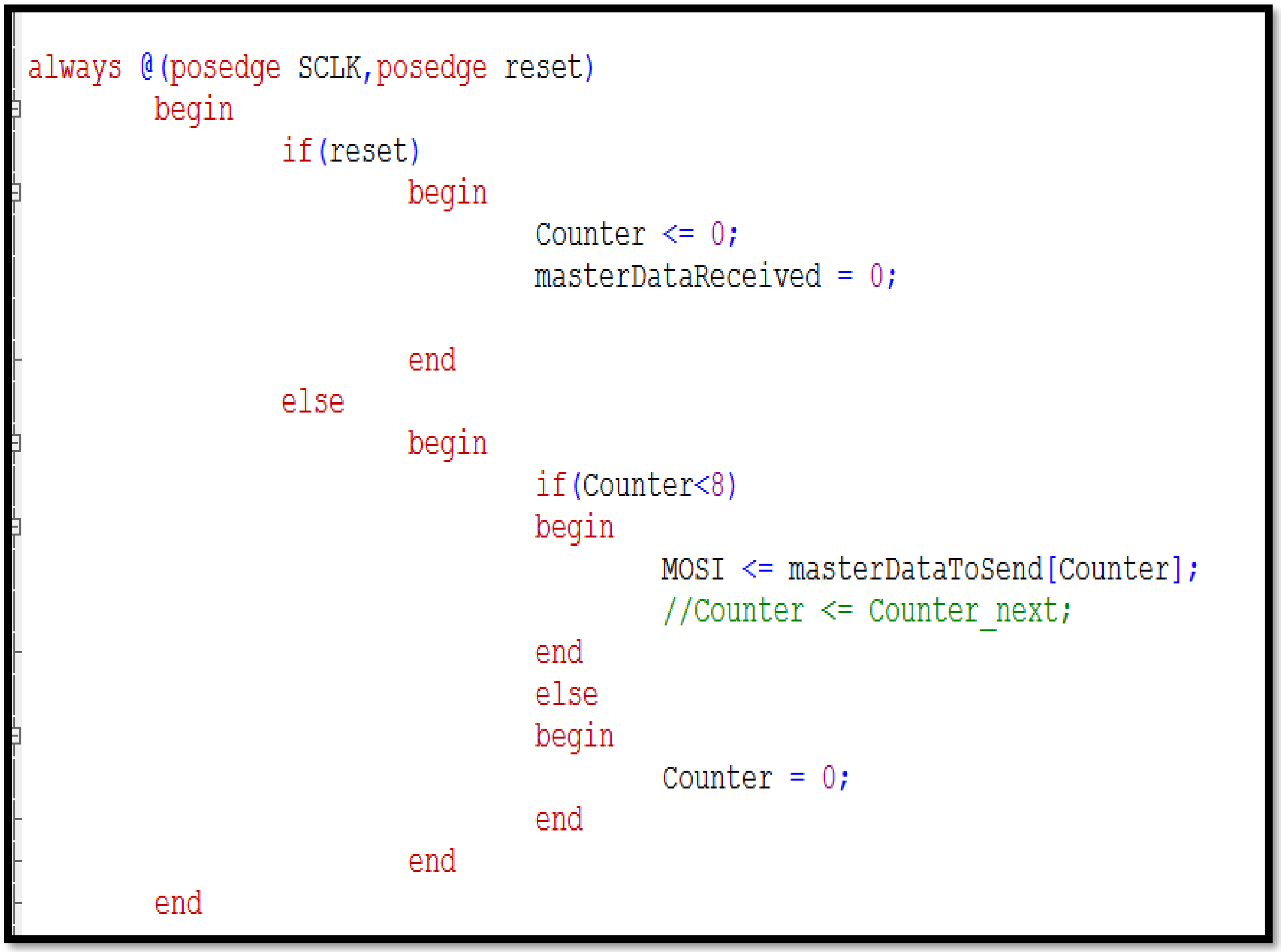
which slave will master **Send** or **Receive** from.



```
// Determine CS to send from SelectSlave
assign CS = (Start_Signal == 0)? 3'b111:
            (slaveSelect == 0)? 3'b011:
            (slaveSelect == 1)? 3'b101:
            (slaveSelect == 2)? 3'b110:3'b111;
```

4) MOSI

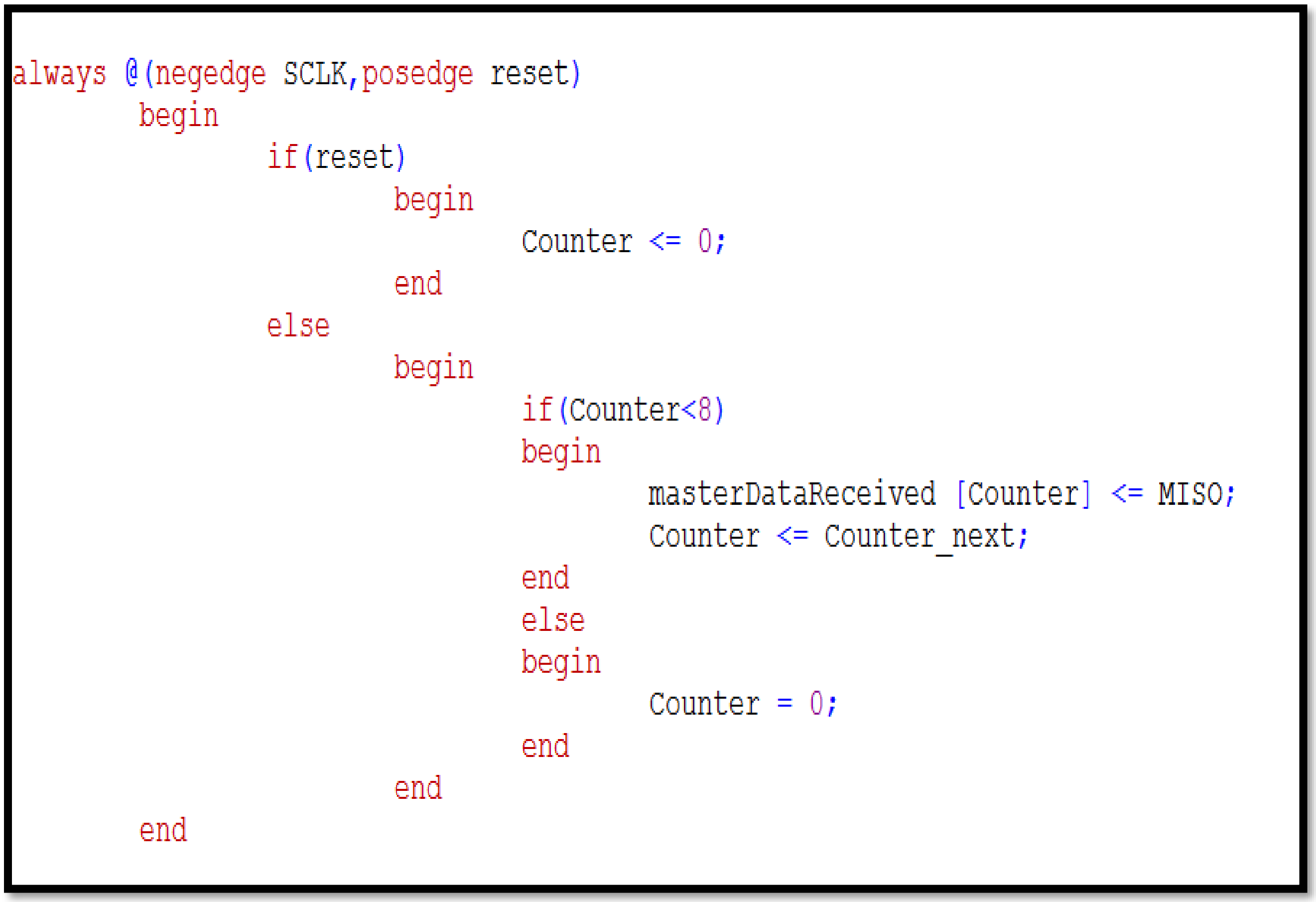
Using **SCLK** which is the cLock that outputs synchronized with, On **Posedge**↑,we shift out **MasterDataToSend** to **MOSI** bit by bit using Counter.



```
always @(posedge SCLK,posedge reset)
        begin
                if(reset)
                        begin
                                Counter <= 0;
                                masterDataReceived = 0;

                        end
                else
                        begin
                                if(Counter<8)
                                begin
                                        MOSI <= masterDataToSend[Counter];
                                        //Counter <= Counter_next;
                                end
                                else
                                begin
                                        Counter = 0;
                                end
                        end
        end
```

5) MasterDataReceived

On **Negedge**↓ ,We receive Data send by slave throught **MISO** & store it in **MasterDataReceived** bit by bit using counter**.**



```
always @(negedge SCLK,posedge reset)
        begin
                if(reset)
                        begin
                                Counter <= 0;
                        end
                else
                        begin
                                if(Counter<8)
                                begin
                                        masterDataReceived [Counter] <= MISO;
                                        Counter <= Counter_next;
                                end
                                else
                                begin
                                        Counter = 0;
                                end
                        end
        end
```

Master Test Banch

The data used to check the master:

Input data:

- *Clock(clk)*
- *Start*
- *Reset*
- *Miso*
- *Salve select of width(2)*
- *Master data to salve of width(8)*

Output data :

- *Sclk*
- *Mosi*
- *Cs of width(3)*
- *Master data received of width(8)*

Storage elements:

- *MOSI of width (8)*
- *MISO of width (8)*
- *testcase_masterData (array of 8 elements every element of width 8)*
- *testcase_salveData (array of 8 elements every element of width 8)*

the processes of the test:

1. *the clock will be changed every 10 units of time by using always block*
2. *using the initial block to initialize some data like (clock,start,reset, testcase_masterData, testcase_salveData ,SalveSelect,)*

the logic:

Using for loop to initialize the data like (Master data to salve and Miso) and the processes of initializing the first variable by using a temporary parameter (testcase_masterData).

After initializing the data, we fill the testing data, we fill the parameter (MOSI) from the output (Mosi) in each period and after eight iterations, the variable (MOSI) will be compeletly filled.

In the end of each iteration we check the data by using if condition and if the data is correct, it displays "successful" and if the data is incorrect,it displays "failed" and the ckeck will be between the (testcase_salveData&&MasterDataReceived) and(MOSI&&MasterDataToSalve).

Slave Module

•Parameters

a) Input :

i) SClk (1 bit).

ii) reset (1 bit).

iii) CS (1 bit).

iv)MOSI (1 bit).

v) slaveDataToSend (8 bits)

b) Output :

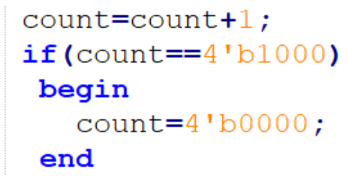
i)MISO (1 bit).

ii)slaveDataReceived (8 bits).

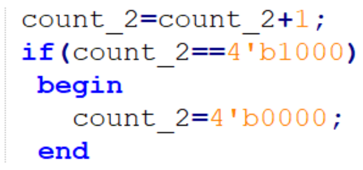
•Slave Design

1) **Count:**
   -The algorithm of setting the data in slaveDataReceived from MOSI is the same as setting the data in masterDataReceived from MISO , it is done by replacing bit by bit using data "Count" which increments with each cycle.

-It is important to add this part of code to set Count to zero again if we finish the 8 cycles, which we determined before to perform the receiving and sending operations.



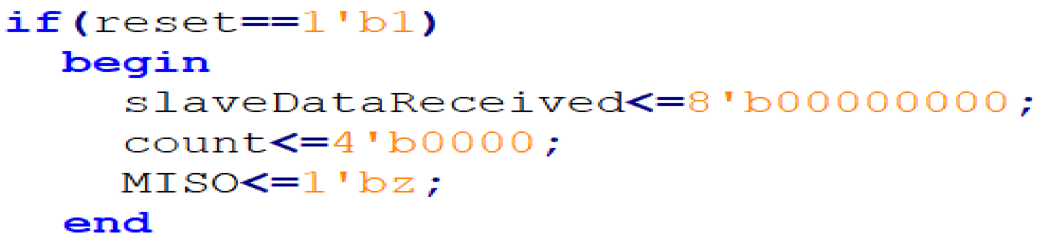
```
count=count+1;
if(count==4'b1000)
 begin
   count=4'b0000;
 end
```



```
count_2=count_2+1;
if(count_2==4'b1000)
 begin
   count_2=4'b0000;
 end
```

2) Reset:

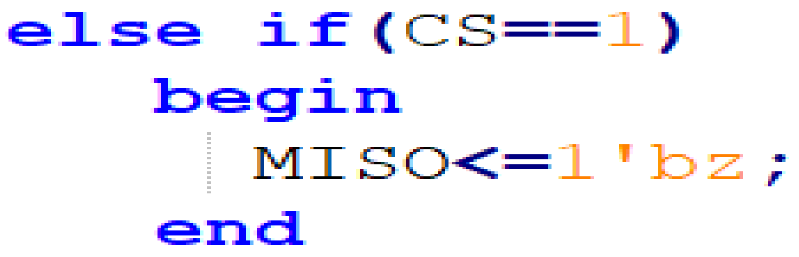
if reset=1, we reset the data of slave , Count , or slaveDataReceived and set MISO with "z"



```
if(reset==1'b1)
  begin
    slaveDataReceived<=8'b00000000;
    count<=4'b0000;
    MISO<=1'bz;
  end
```

3) CS:

1-if CS=1,no receiving or sending is done through the slave , we just set the output MISO of this slave to "z" to prevent this slave to overwrite in MISO , it is important if there are many slaves connected with the master , so we allow the master to receive the data from the slave which it targets and sets this salve's CS with 0.



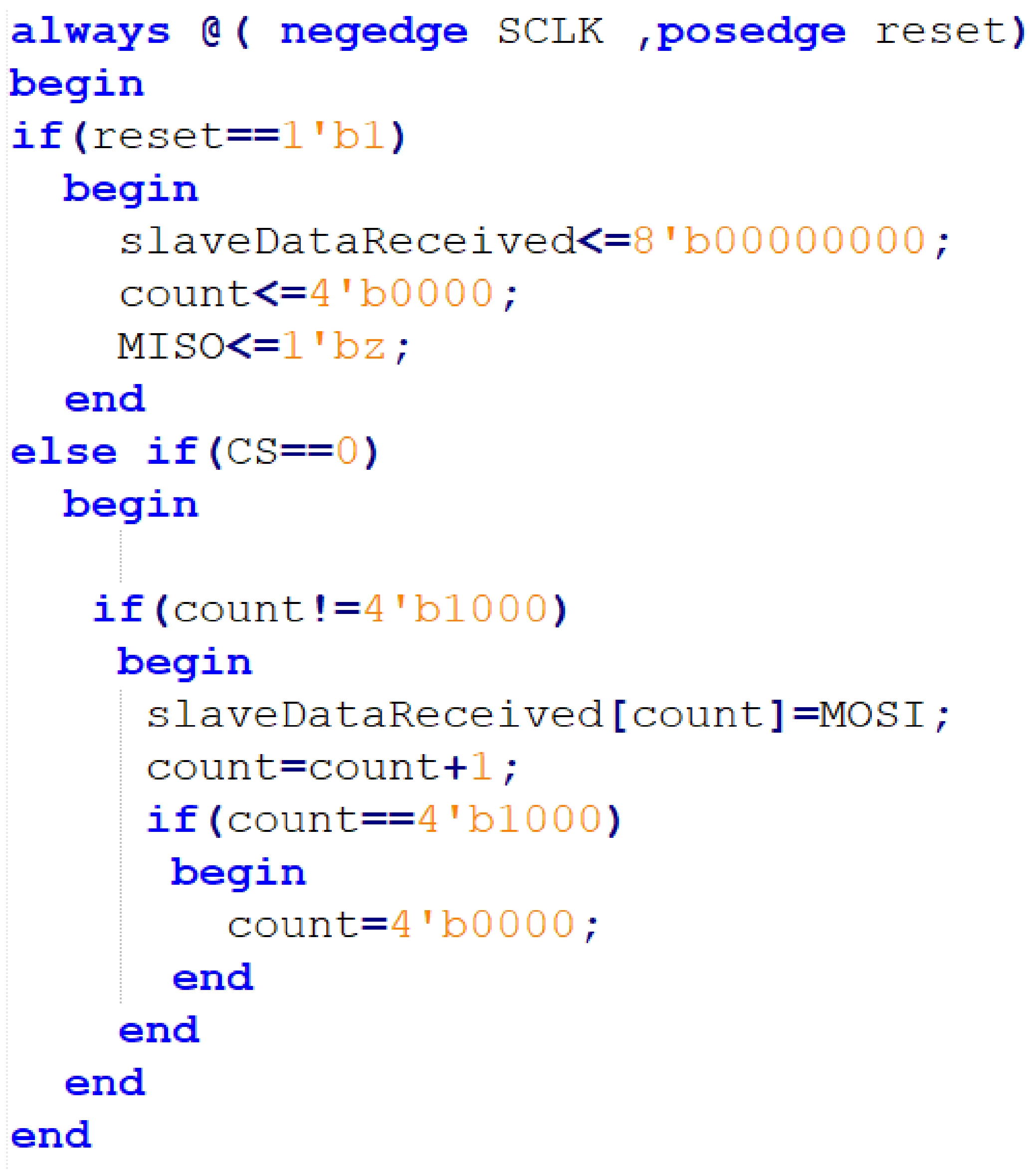
```
else if(CS==1)
    begin
      MISO<=1'bz;
    end
```

2-If CS=0, then both operations work

4) SCLK:

•If CS=0 and for each cycle:

1-at the **negative edge** of SCLK , we set the data coming from the master (MOSI) to the bit (which has the turn in this cycle) of the slaveDataReceived.



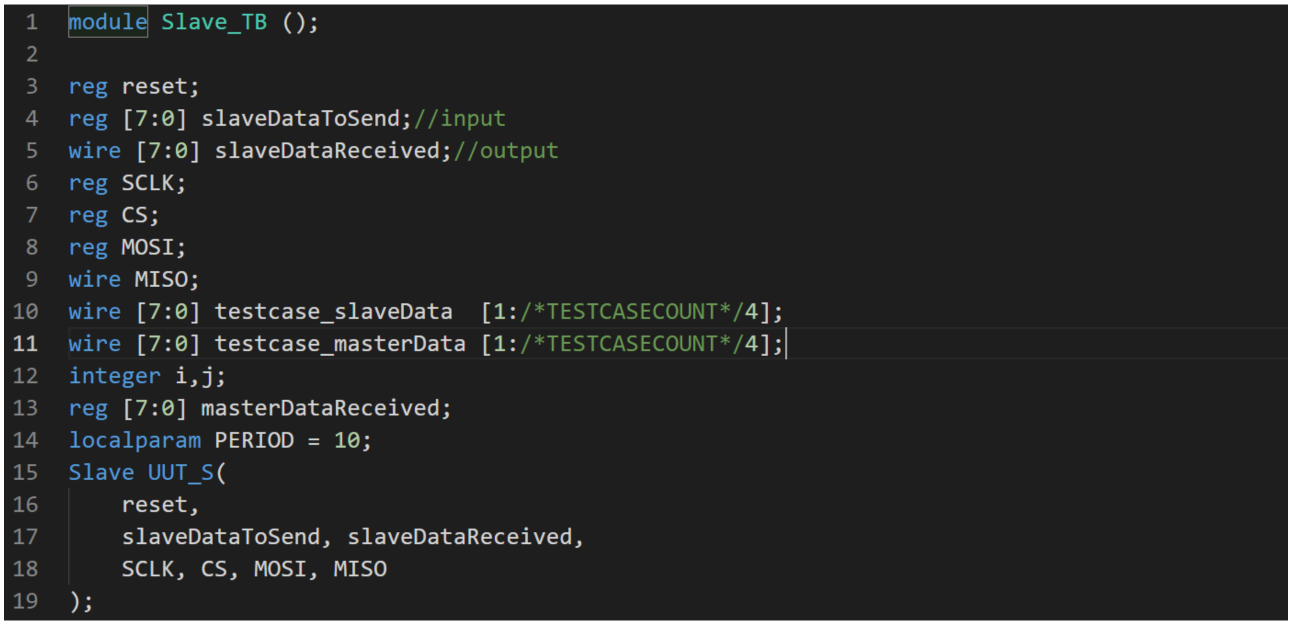
```
always @( negedge SCLK ,posedge reset)
begin
if(reset==1'b1)
  begin
    slaveDataReceived<=8'b00000000;
    count<=4'b0000;
    MISO<=1'bz;
  end
else if(CS==0)
  begin

     if(count!=4'b1000)
      begin
       slaveDataReceived[count]=MOSI;
       count=count+1;
       if(count==4'b1000)
        begin
          count=4'b0000;
        end
      end
  end
end
```

2-at the **positive edge** of SCLK, we set the bit (which also has the turn in this cycle) of the slaveDataToSend to the MISO.

```
always @( posedge SCLK )
begin
 if(CS==0 && reset==1'b0)
  begin

    if(count_2!=4'b1000)
     begin
      MISO=slaveDataToSend[count_2];
      count_2=count_2+1;
      if(count_2==4'b1000)
       begin
         count_2=4'b0000;
       end
     end
  end
else if(CS==1)
    begin
      MISO<=1'bz;
    end

end
```

Slave Test Banch

[I]Declaring variables ,used Regs & Wires



```
module Slave_TB ();

reg reset;
reg [7:0] slaveDataToSend;//input
wire [7:0] slaveDataReceived;//output
reg SCLK;
reg CS;
reg MOSI;
wire MISO;
wire [7:0] testcase_slaveData  [1:/*TESTCASECOUNT*/4];
wire [7:0] testcase_masterData [1:/*TESTCASECOUNT*/4];
integer i,j;
reg [7:0] masterDataReceived;
localparam PERIOD = 10;
Slave UUT_S(
    reset,
    slaveDataToSend, slaveDataReceived,
    SCLK, CS, MOSI, MISO
);
```

A- ***Regs***

1) Reset
2) SCLK: slave simulation Clock
3) CS: slave ship select
4) MOSI: Master Output slave input
5) SlaveDataToSend (8 bits): Data stored in Slave to be sent to master [TestCase]

6) MasterDataReceived (8 bits): Date data send from slave and received in master [TestCase]

B- ***<u>Wires:</u>***

1) MISO
2) Slave Data Received (8 bits) (Module output, Net data recieved)
3) testcase_slaveData (8 bits) ,array of 4 Testcases for slave data
4) testcase_masterData (8 bits) ,array of 4 Testcases for master data

C- ***<u>Variables</u>***

1) i -> integer to loop 4 times to test the 4 testcases

2) j -> for inner loop , which will be discussed later

3) UUT_S -> Slave module instance

4) LocalParameter PERIOD -> timeperiod of full SCLK cycle

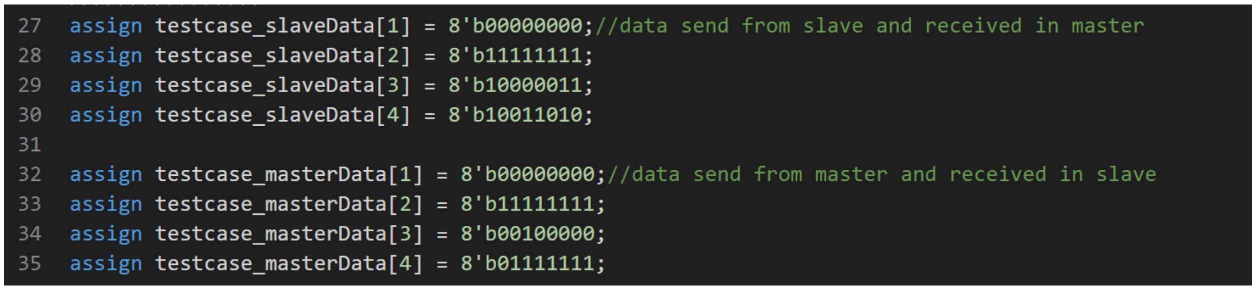
[II]Setting up SCLK

In this test bench only, we assume that SCLK is synchronous, unlike real deal in the Development module, so in testing only the slave module, we assume that only one slave is sending to and receiving from one master (SCLK = ~Master's CLK)

```
always #(PERIOD/2) SCLK = ~SCLK;
```

[III]Setting up Test cases

Here we have chosen 4 test cases for verifying successful transmission to and from slave module

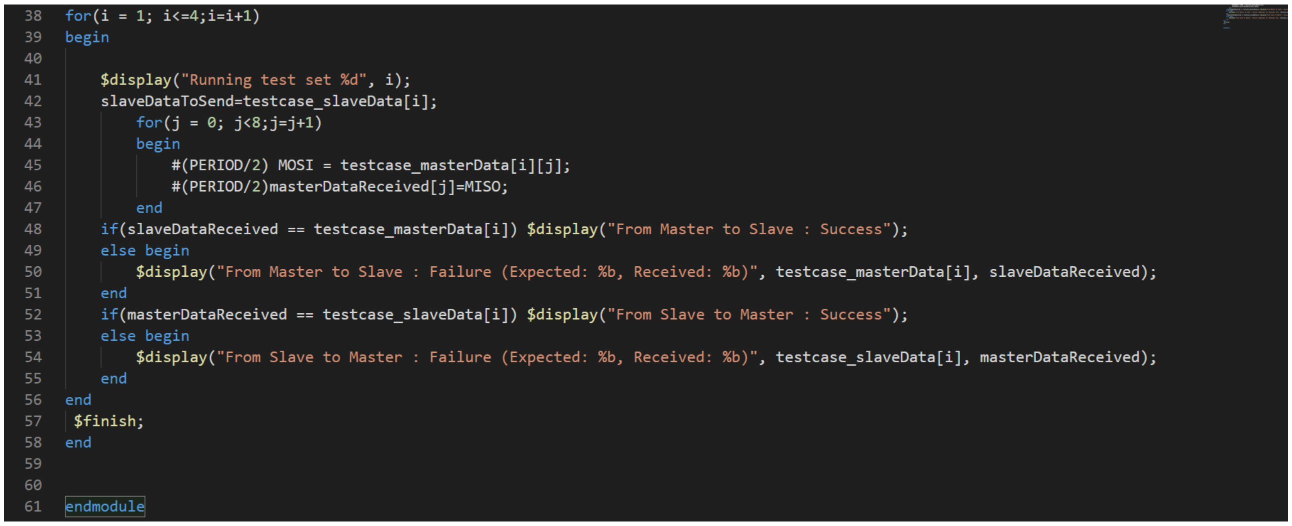


```
assign testcase_slaveData[1] = 8'b00000000;//data send from slave and received in master
assign testcase_slaveData[2] = 8'b11111111;
assign testcase_slaveData[3] = 8'b10000011;
assign testcase_slaveData[4] = 8'b10011010;

assign testcase_masterData[1] = 8'b00000000;//data send from master and received in slave
assign testcase_masterData[2] = 8'b11111111;
assign testcase_masterData[3] = 8'b00100000;
assign testcase_masterData[4] = 8'b01111111;
```

[IV]Ports Initialization

```
initial begin
    SCLK = 1;
    reset = 1;
    #PERIOD reset = 0;
    i=0;j=0;
    CS = 0;
```

[V]Testing logic



```
for(i = 1; i<=4;i=i+1)
begin

    $display("Running test set %d", i);
    slaveDataToSend=testcase_slaveData[i];
        for(j = 0; j<8;j=j+1)
        begin
            #(PERIOD/2) MOSI = testcase_masterData[i][j];
            #(PERIOD/2)masterDataReceived[j]=MISO;
        end
    if(slaveDataReceived == testcase_masterData[i]) $display("From Master to Slave : Success");
    else begin
        $display("From Master to Slave : Failure (Expected: %b, Received: %b)", testcase_masterData[i], slaveDataReceived);
    end
    if(masterDataReceived == testcase_slaveData[i]) $display("From Slave to Master : Success");
    else begin
        $display("From Slave to Master : Failure (Expected: %b, Received: %b)", testcase_slaveData[i], masterDataReceived);
    end
end
 $finish;
end


endmodule
```

#1: loop on 4 test cases from i = 0 to 4

#2: in each loop:

- displays ID of test case

- executes inner loop of j from 0 to 7(chosen 8 loops for 8 bits, 0 to 7 for significance of ports)

->to send and receive from slave, in each inner loop:

•assigns at +ve edge the j^th bit to MOSI

•assigns at -ve edge the MISO bit to j^th bit of masterDataReceived

- After inner loop finish, the test data should be transmitted and received successfully

, being a self-checking testbench , it does 2 checks:

•#1 :

```
if
(slaveDataReceived == testcase_masterData[i])
```

>meaning: if Module Net output data received by slave is identical to current sent masterdata testcase

•#2 :

```
    if(masterDataReceived == testcase_slaveData[i])
```

>meaning: if Module Net output data received by master is identical to current sent slavedata testcase

- At the end, whenever a check is true: it displays

"From Master to Slave : Success" in #1

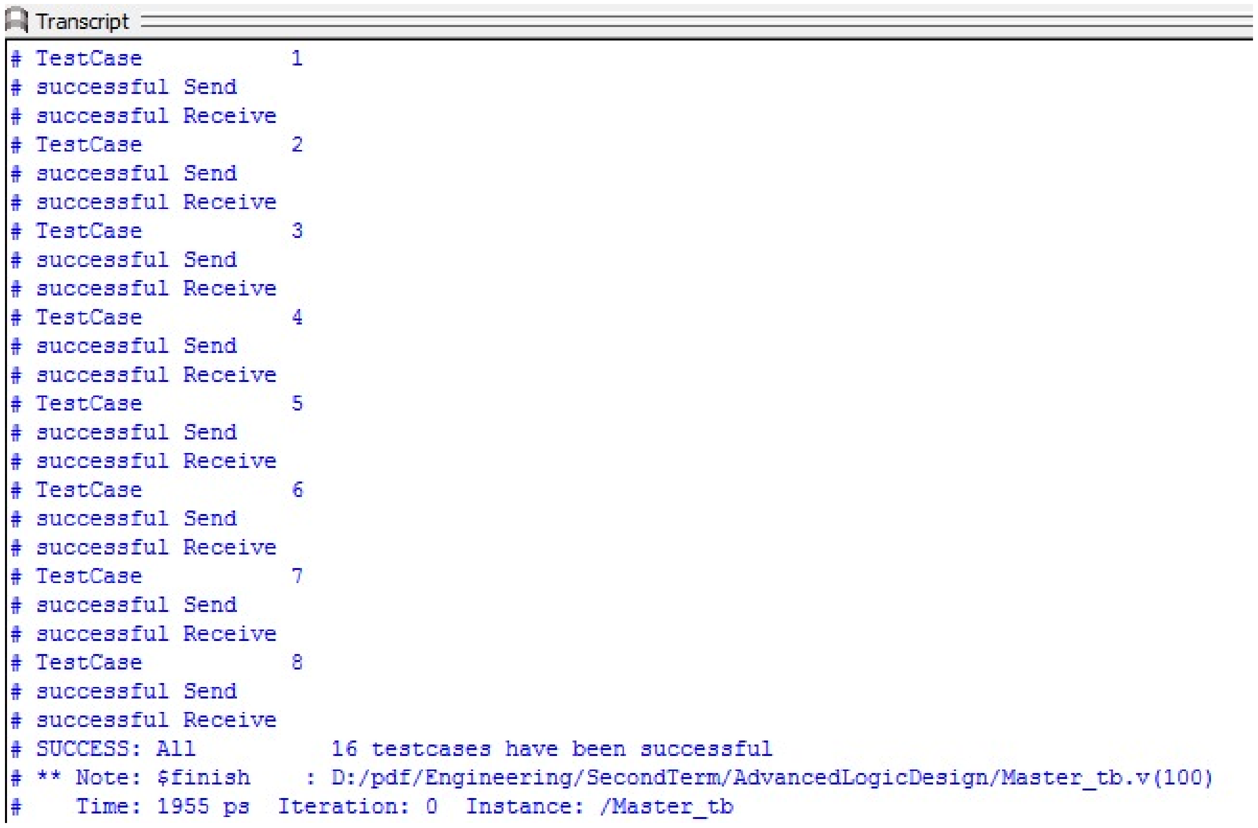
"From Slave to Master : Success" in #2

If false, it displays that it was a failure & displays both the expected output & the wrong one

Simuation Results

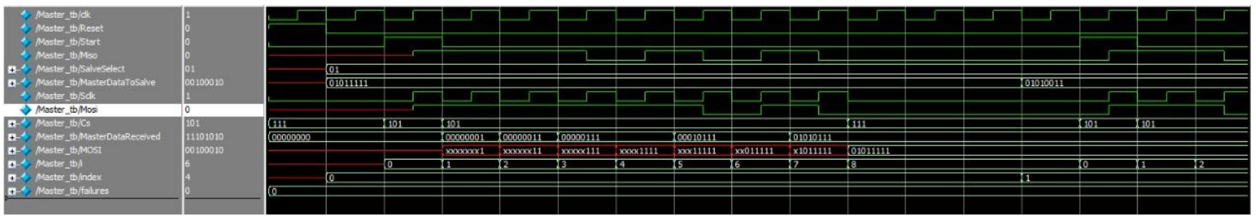
The output and the wave form

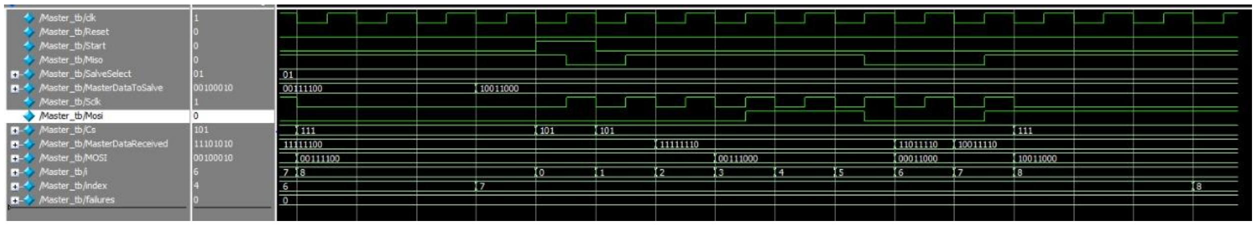
#1 Master TB



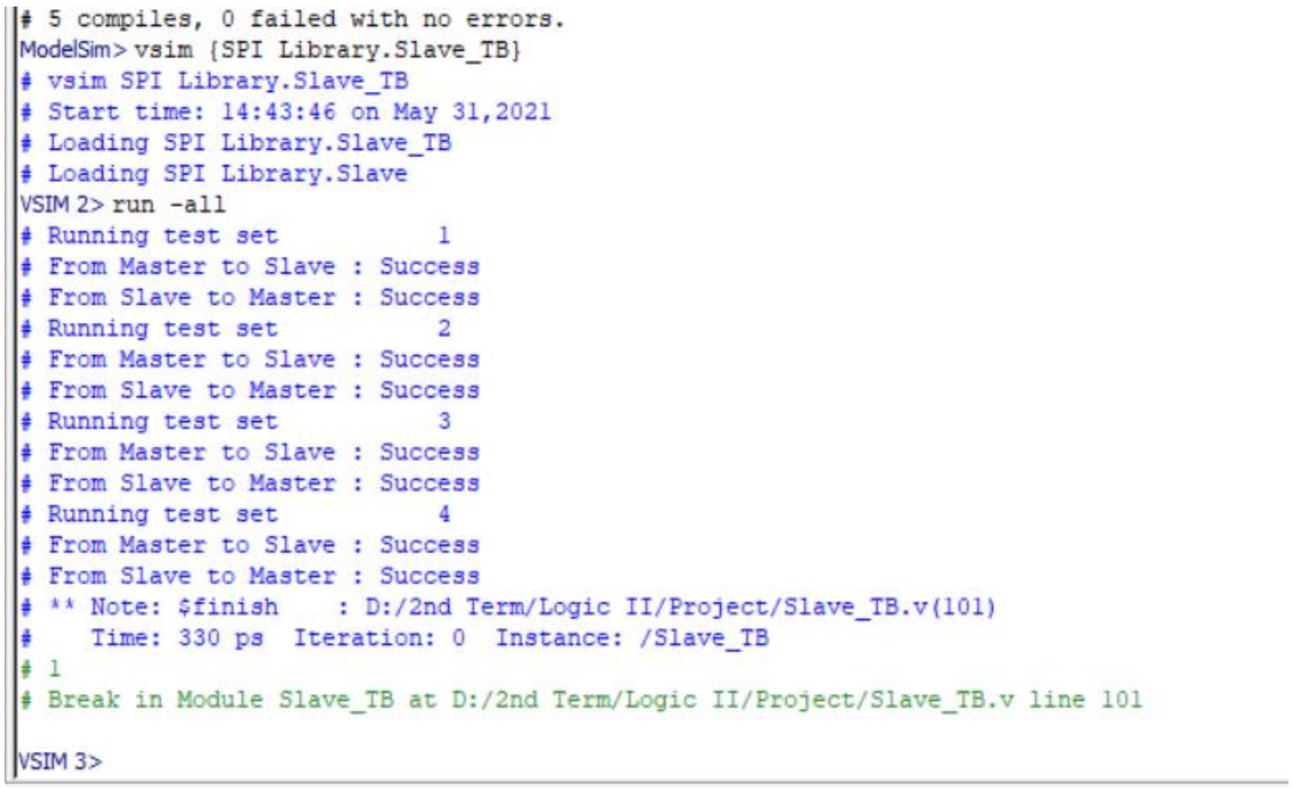
Transcript

```
# TestCase          1
# successful Send
# successful Receive
# TestCase          2
# successful Send
# successful Receive
# TestCase          3
# successful Send
# successful Receive
# TestCase          4
# successful Send
# successful Receive
# TestCase          5
# successful Send
# successful Receive
# TestCase          6
# successful Send
# successful Receive
# TestCase          7
# successful Send
# successful Receive
# TestCase          8
# successful Send
# successful Receive
# SUCCESS: All          16 testcases have been successful
# ** Note: $finish    : D:/pdf/Engineering/SecondTerm/AdvancedLogicDesign/Master_tb.v(100)
#    Time: 1955 ps  Iteration: 0  Instance: /Master_tb
```

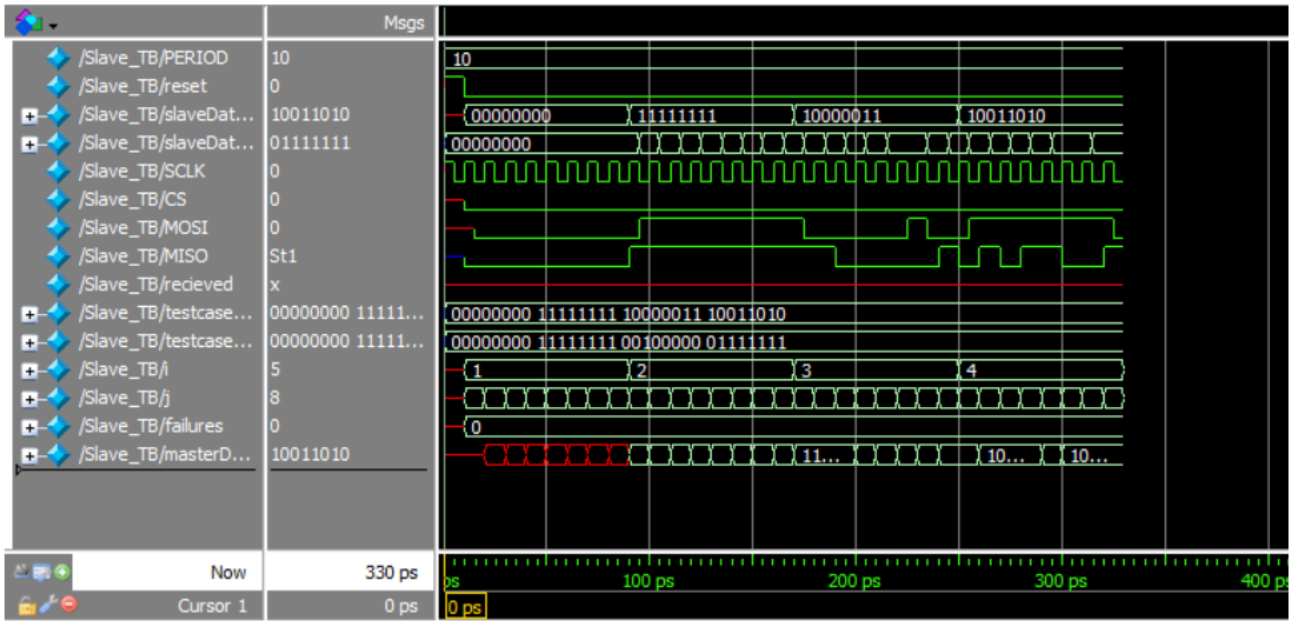




#2 Slave TB

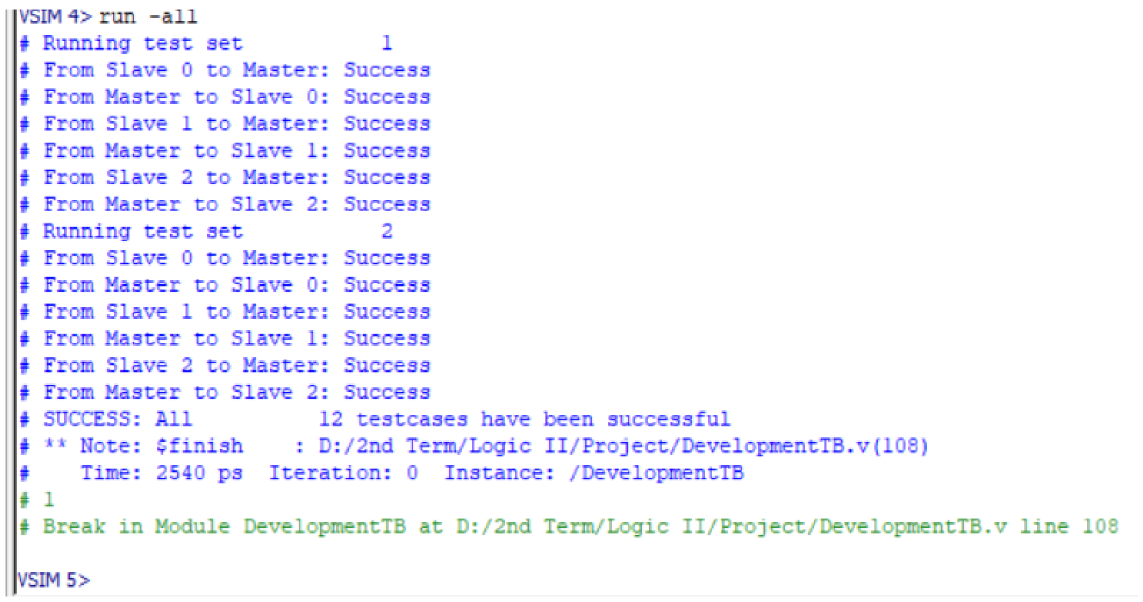


```
# 5 compiles, 0 failed with no errors.
ModelSim> vsim {SPI Library.Slave_TB}
# vsim SPI Library.Slave_TB
# Start time: 14:43:46 on May 31,2021
# Loading SPI Library.Slave_TB
# Loading SPI Library.Slave
VSIM 2> run -all
# Running test set          1
# From Master to Slave : Success
# From Slave to Master : Success
# Running test set          2
# From Master to Slave : Success
# From Slave to Master : Success
# Running test set          3
# From Master to Slave : Success
# From Slave to Master : Success
# Running test set          4
# From Master to Slave : Success
# From Slave to Master : Success
# ** Note: $finish    : D:/2nd Term/Logic II/Project/Slave_TB.v(101)
#    Time: 330 ps  Iteration: 0  Instance: /Slave_TB
# 1
# Break in Module Slave_TB at D:/2nd Term/Logic II/Project/Slave_TB.v line 101

VSIM 3>
```

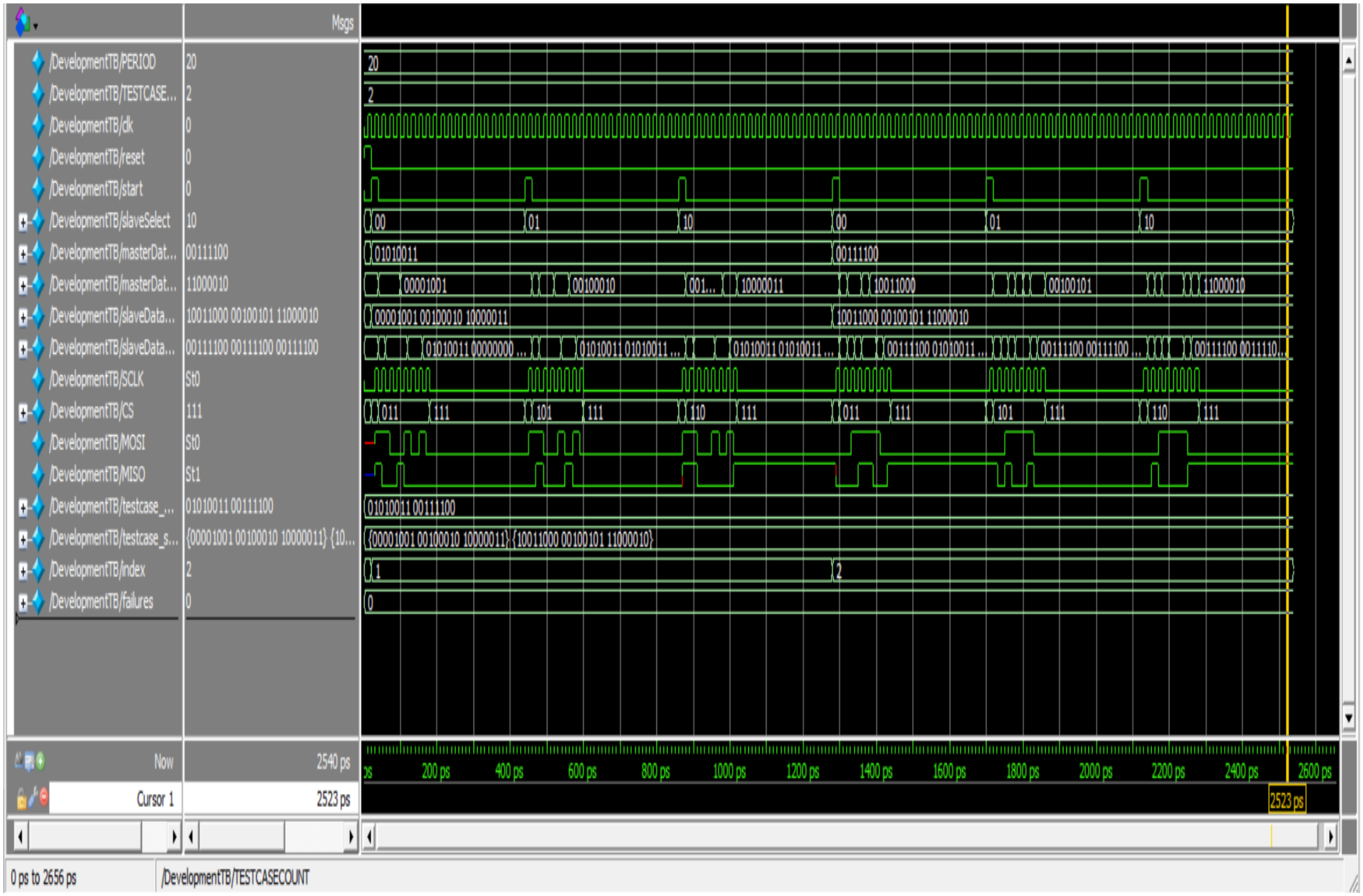


```
                         Msgs
/Slave_TB/PERIOD         10
/Slave_TB/reset          0
/Slave_TB/slaveDat...    10011010
/Slave_TB/slaveDat...    01111111
/Slave_TB/SCLK           0
/Slave_TB/CS             0
/Slave_TB/MOSI           0
/Slave_TB/MISO           St1
/Slave_TB/recieved       x
/Slave_TB/testcase...    00000000 11111...
/Slave_TB/testcase...    00000000 11111...
/Slave_TB/i              5
/Slave_TB/j              8
/Slave_TB/failures       0
/Slave_TB/masterD...     10011010

Now                      330 ps
Cursor 1                 0 ps
```

#3 Development TB



```
VSIM 4> run -all
# 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
# ** Note: $finish    : D:/2nd Term/Logic II/Project/DevelopmentTB.v(108)
#    Time: 2540 ps  Iteration: 0  Instance: /DevelopmentTB
# 1
# Break in Module DevelopmentTB at D:/2nd Term/Logic II/Project/DevelopmentTB.v line 108

VSIM 5>
```



	Msgs
/DevelopmentTB/PERIOD	20
/DevelopmentTB/TESTCASE...	2
/DevelopmentTB/clk	0
/DevelopmentTB/reset	0
/DevelopmentTB/start	0
/DevelopmentTB/slaveSelect	10
/DevelopmentTB/masterDat...	00111100
/DevelopmentTB/masterDat...	11000010
/DevelopmentTB/slaveData...	10011000 00100101 11000010
/DevelopmentTB/slaveData...	00111100 00111100 00111100
/DevelopmentTB/SCLK	St0
/DevelopmentTB/CS	111
/DevelopmentTB/MOSI	St0
/DevelopmentTB/MISO	St1
/DevelopmentTB/testcase_...	01010011 00111100
/DevelopmentTB/testcase_s...	{00001001 00100010 10000011} {10...
/DevelopmentTB/index	2
/DevelopmentTB/failures	0

Now	2540 ps	
Cursor 1	2523 ps	2523 ps

0 ps to 2656 ps /DevelopmentTB/TESTCASECOUNT