Digital Electronics Courses:

Verilog Final Project

Team Member:

Mohamed Hatem Adly

Islam Gamal

Maged Nagy

Table of Contents

1.0 Waveforms

1.1 Memory Testbench

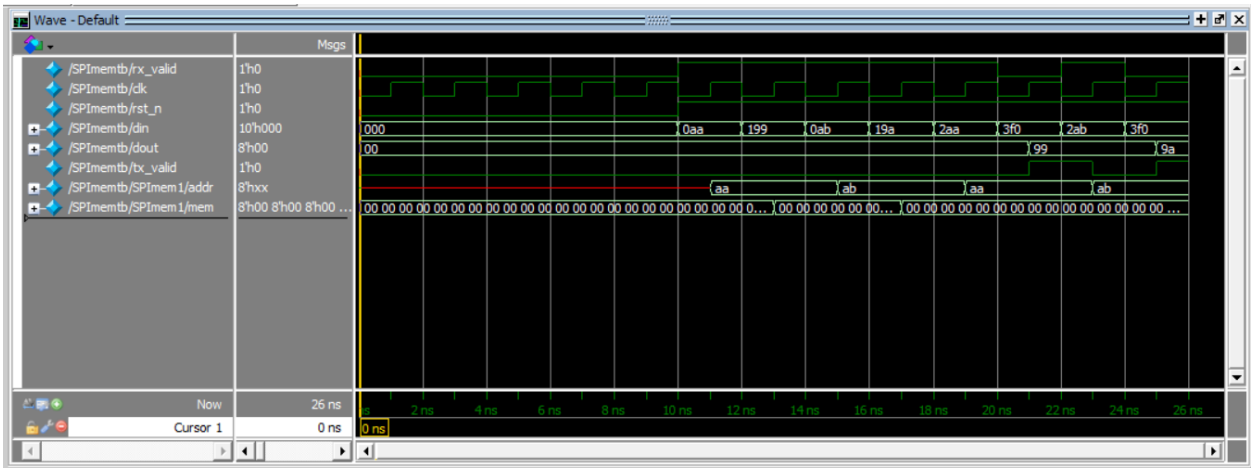

Figure 1 memory testbench waveform

First the resetting:
As the simulation start running the memory takes its data from the pre-generated file.

Second the writing address at 10ns:
The “din” changes also the “rst_n” and “rx_valid” are now 1, so the memory is ready to take the data in the next clock edge. The most significant bits are “00” so the memory save the address “aa” inside the “addr” register. The operation repeats at 14ns.

Third the writing data at 12 ns:
The “din” now has now value this its most significant bit are “01”, so the memory saves the data inside the pre-stored address. The operation repeats at 16ns.

Fourth the reading address at 18ns:
The “din” now has now value this its most significant bit are “10”, so the memory saves the data inside “addr” register. The operation repeats at 22ns.

Fifth the reading data at 20ns:
The “din” now has now value this its most significant bit are “11”, so the memory sent the data inside the pre-stored address and ignoring the rest of the sent data. We can see also the “tx_valid” is 1 now. The sent data in “dout” is exactly the data we sent in the writing part of the testing. The operation repeats at 24ns.

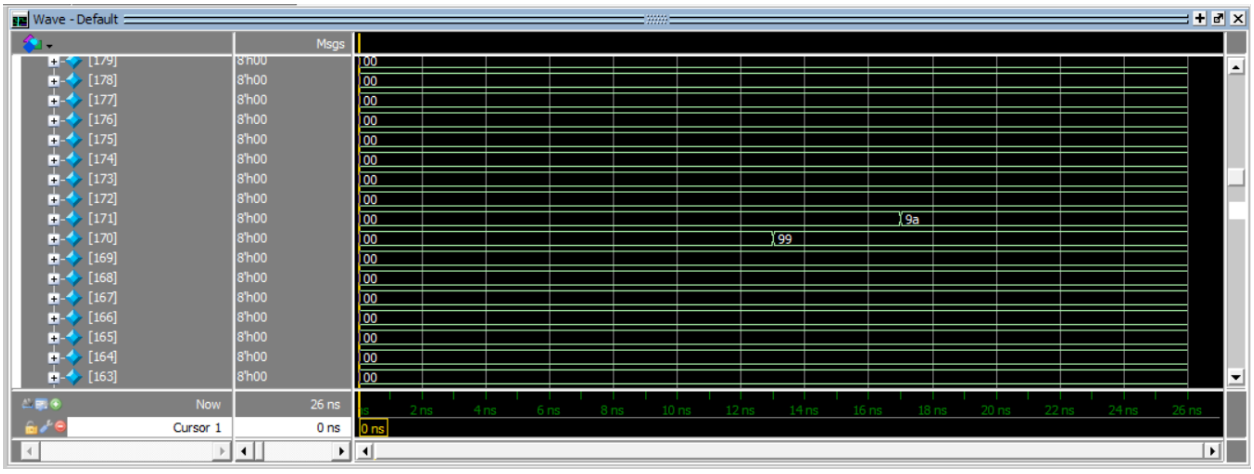

Figure 2 the changing of the data inside the memory

1.2 The Slave Testbench

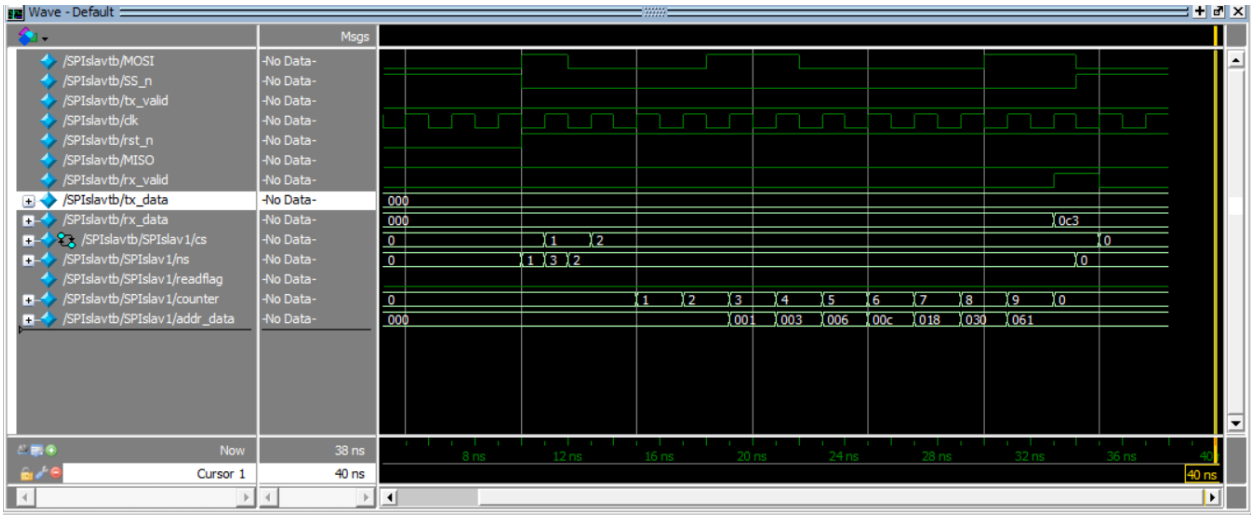

Figure 3 slave testbench waveform

First the resetting:
All the salve outputs are now zero.

Second the serial to parallel conversation:
Now the “rst_n” is 1 also the “SS_n” is 1, the master is starting the communication and the next cycle the slave in on the “CHK_CMD” state. After this cycle the “MOSI” is 0, so the slave knows it’s a writing command. The next 10 cycles the slave is storing the input data in the “addr_data” so convert is later into parallel signal. At 34ns the slave is sending the data to the memory. Notice the slave doesn’t save the last bit as it sends it as it arrives. The conversion from parallel to serial is tested in the last test bench.

STATE	NUMBER
IDEL	0
CHK_CMD	1
WRITING	2
READ_ADD	3
READ_DATA	4

1.3 The Last Testbench

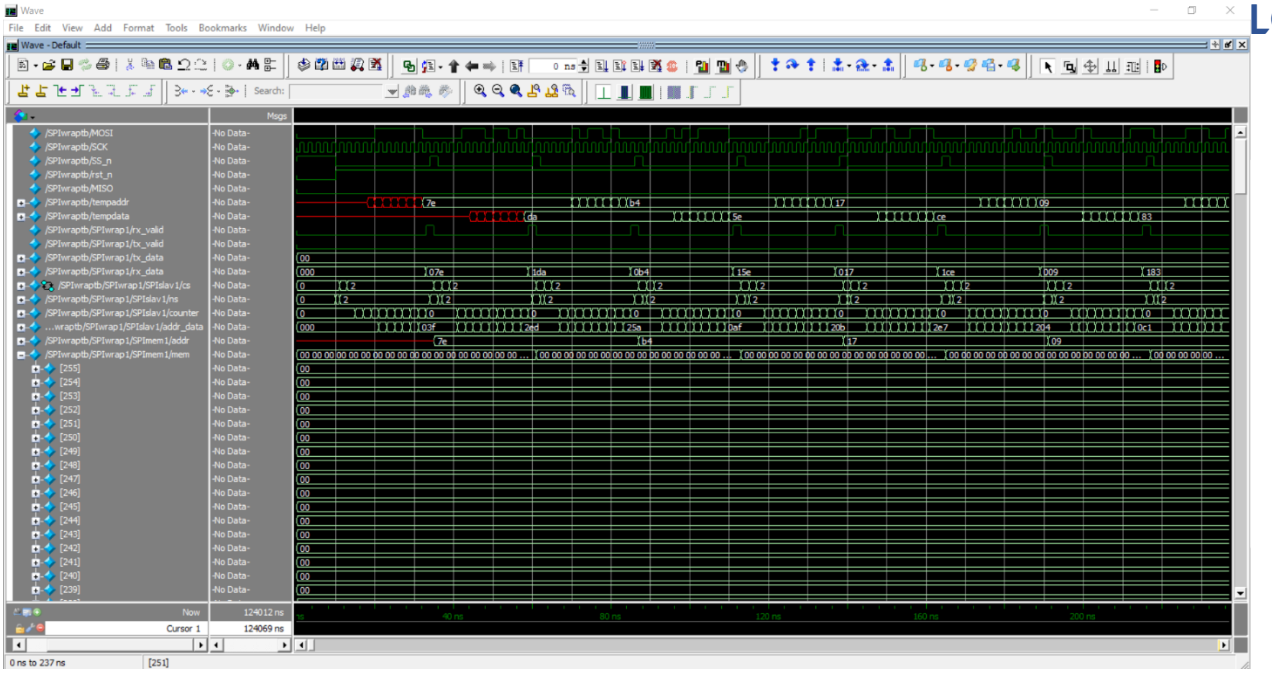

Figure 4 the wrapper testbench waveform

Notice that the “tempaddr” and “tempdata” are testbench signals just storing the sent and received data and compare it with the data inside the memory to catch the error if happened.

First the resetting:
All the wrapper outputs are zero and the memory takes its data from the pre-generated file.

Second the writing:
Using a constrained randomization to make sure we starting with writing address then the writing data and randomize the rest of the bits. We can see the “rx_data” is the same value as the “tempaddr” or the “tempdata” according to the state and also remember that the testbench code is comparing the “tempaddr” or the “tempdata” with the data inside the memory.

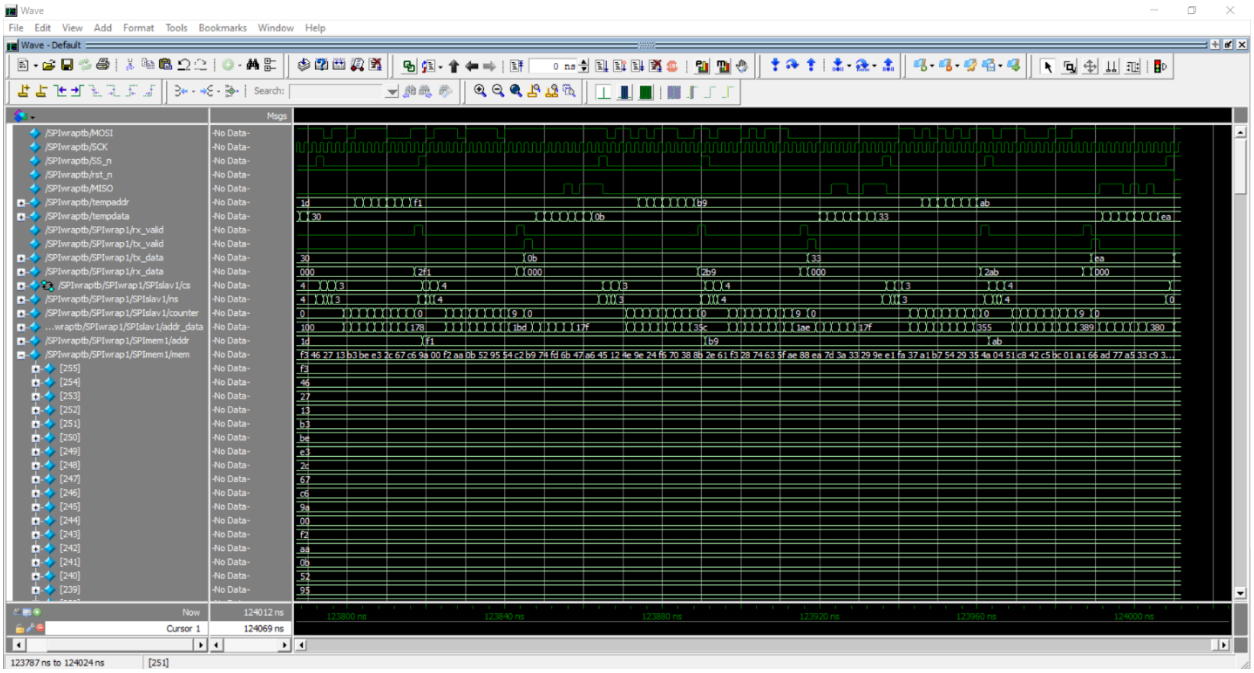

Figure 5 the wrapper testbench waveform

Notice that the memory is filled with data now.

Third the reading:
Using a constrained randomization to make sure we are starting with reading address then the reading command and randomize the rest of the bits. We can see the “rx_data” and the “tx_data” are taking the same value as the “tempaddr” or the “tempdata” according to the state and also remember that the testbench code is comparing the “tempaddr” or the “tempdata” with the data inside the memory.

1.4 The Testbench report

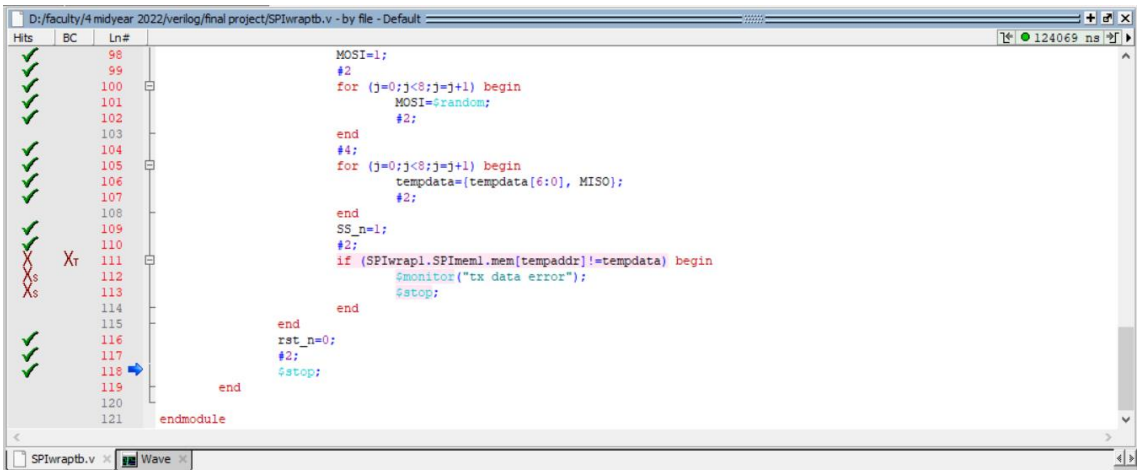

Figure 6 the simulation stops without errors

Transcript

File Edit View Bookmarks Window Help

Transcript

```
# Model Technology ModelSim SE-64 vcover 10.7 Coverage Utility 2017.12 Dec  7 2017
# Start time: 12:04:22 on Mar 09,2022
# vcover report SPIwraptb.ucdb
# Coverage Report Summary Data by file
#
# ================================================================================
# === File: SPImem.v
# ================================================================================
#     Enabled Coverage            Active      Hits    Misses % Covered
#     ----------------            ------      ----    ------ ---------
#     Stmts                           14        14         0    100.00
#     Branches                         8         8         0    100.00
#     FEC Condition Terms              6         5         1     83.33
#     Toggle Bins                     62        61         1     98.38
#
# ================================================================================
# === File: SPIslav.v
# ================================================================================
#     Enabled Coverage            Active      Hits    Misses % Covered
#     ----------------            ------      ----    ------ ---------
#     Stmts                           57        53         4     92.98
#     Branches                        36        34         2     94.44
#     FEC Condition Terms             12        10         2     83.33
#     FSMs                                                    93.75
#         States                       5         5         0    100.00
#         Transitions                  8         7         1     87.50
#     Toggle Bins                     92        91         1     98.91
#
# ================================================================================
# === File: SPIwrap.v
# ================================================================================
#     Enabled Coverage            Active      Hits    Misses % Covered
#     ----------------            ------      ----    ------ ---------
#     Toggle Bins                     50        49         1     98.00
#
# ================================================================================
# === File: SPIwraptb.v
# ================================================================================
#     Enabled Coverage            Active      Hits    Misses % Covered
#     ----------------            ------      ----    ------ ---------
#     Stmts                           93        85         8     91.39
#     Branches                         8         4         4     50.00
#     FEC Condition Terms              4         0         4      0.00
#     Toggle Bins                    170        69       101     40.58
#
#
# Total Coverage By File (code coverage only, filtered view): 83.05%
```

Figure 7 coverage report

2.0 Appendix

2.1 Memory Code

```
module SPImem(din, rx_valid, clk, rst_n, dout, tx_valid);

	input rx_valid, clk, rst_n;
	parameter MEM_DEPTH=256, ADDR_SIZE=8;
	input [ADDR_SIZE+1:0] din;
	output reg [ADDR_SIZE-1:0]dout;
	output reg tx_valid;
	reg [ADDR_SIZE-1:0] addr;
	reg sent;

	reg [ADDR_SIZE-1:0] mem [MEM_DEPTH-1:0];

	always @(posedge clk or negedge rst_n) begin
		if (~rst_n) begin
			dout<='b0;
			tx_valid<=0;
			sent<=0;
		end
		else if (rx_valid && din[9:8]!=2'b11) begin
			if (din[9:8]==2'b01) begin
				mem[addr]<=din[7:0];
				tx_valid<=0;
			end
			else begin
				addr<=din[7:0];
				tx_valid<=0;
				if (din[9:8]==2'b10) begin
					sent<=0;
				end
				else begin
					sent<=1;
				end
			end
		end
		else if (din[9:8]==2'b11 && sent==0) begin
			dout<=mem[addr];
			tx_valid<=1;
			sent<=1;
		end
		else begin
```

```
                tx_valid<=0;
            end
        end

endmodule
```

2.2 Memory Testbench Code

```
module SPImemtb();

        reg rx_valid, clk, rst_n;
        reg [9:0] din;
        wire [7:0]dout;
        wire tx_valid;

        SPImem SPImem1(din, rx_valid, clk, rst_n, dout, tx_valid);

        initial begin
            clk=0;
            forever
            #1 clk=!clk;
        end

        initial begin
            $readmemb("mem.dat",SPImem1.mem);
        end

        initial begin
            rst_n=0;
            rx_valid=0;
            din=0;
            #10;
            rst_n=1;
            rx_valid=1;
            din='b0010101010;
            #2;
            rx_valid=1;
            din='b0110011001;
            #2;
            rx_valid=1;
            din='b0010101011;
            #2;
            rx_valid=1;
            din='b0110011010;
            #2;
            rx_valid=1;
```

```
            din='b1010101010;
            #2;
            rx_valid=0;
            din='b1111110000;
            #2;
            rx_valid=1;
            din='b1010101011;
            #2;
            rx_valid=0;
            din='b1111110000;
            #2;
            $stop;
        end

endmodule
```

2.3 Slave Code

```
module SPIslav(MOSI, MISO, SS_n, clk, rst_n, rx_data, rx_valid, tx_data, tx_valid);

        parameter IDLE=3'b000, CHK_CMD=3'b001, WRITE=3'b010, READ_ADD=3'b011,
READ_DATA=3'b100;
        input MOSI, SS_n, tx_valid, clk, rst_n;
        output reg MISO, rx_valid;
        input [7:0] tx_data;
        output reg [9:0] rx_data;
        reg [2:0] cs,ns;
        reg readflag;
        reg [3:0] counter;
        reg [9:0] addr_data;

        always @(posedge clk or negedge rst_n) begin
            if (~rst_n) begin
                MISO<=0;
                counter<=0;
                readflag<=0;
                addr_data<=0;
                rx_valid<=0;
                rx_data<=0;
                cs<=IDLE;
            end
            else begin
                cs<=ns;
                if (~SS_n) begin
                    if (cs==IDLE) begin
                        rx_valid<=0;
```

```
            MISO<=0;
        end
        else if (cs==CHK_CMD) begin
            rx_valid<=0;
            MISO<=0;
        end
        else if (counter!=9 && cs!=READ_DATA) begin
            addr_data<={addr_data,MOSI};
            counter<=counter+1;
            rx_valid<=0;
            MISO<=0;
        end
        else if (cs==WRITE) begin
            rx_valid<=1;
            rx_data<={addr_data,MOSI};
            counter<=0;
        end
        else if (cs==READ_ADD) begin
            rx_valid<=1;
            rx_data<={addr_data,MOSI};
            counter<=0;
            readflag<=1;
        end
        else if (cs==READ_DATA) begin
            if (counter!=9 && readflag) begin
                addr_data<={addr_data,MOSI};
                counter<=counter+1;
                rx_valid<=0;
                MISO<=0;
            end
            else if (readflag) begin
                rx_valid<=1;
                rx_data<={addr_data,MOSI};
                readflag<=0;
            end
            else begin
                rx_valid<=0;
                rx_data<=0;
                if (~readflag && counter==9) begin
                    counter<=0;
                end
                else if (tx_valid==1) begin
                    addr_data[7:0]<=tx_data;
                    MISO<=tx_data[7];
                end
                else begin
```

```
                                                {MISO,
addr_data[6:1]}<=addr_data[6:0];
                                        end
                                    end
                                end
                                else begin
                                    rx_valid<=0;
                                    MISO<=0;
                                    counter<=0;
                                end
                            end
                            else begin
                                rx_valid<=0;
                                MISO<=0;
                                counter<=0;
                            end
                        end
                    end

                    always @(cs, SS_n, tx_valid, MOSI) begin
                        case (cs)
                            IDLE:
                                if (~SS_n) begin
                                    ns=CHK_CMD;
                                end
                                else begin
                                    ns=IDLE;
                                end
                            CHK_CMD:
                                if (~SS_n) begin
                                    if (~MOSI) begin
                                        ns=WRITE;
                                    end
                                    else if (~readflag) begin
                                        ns=READ_ADD;
                                    end
                                    else begin
                                        ns=READ_DATA;
                                    end
                                end
                                else begin
                                    ns=IDLE;
                                end
                            WRITE:
                                if (~SS_n) begin
                                    ns=WRITE;
```

```
                end
                else begin
                        ns=IDLE;
                end
        READ_ADD:
                if (~SS_n) begin
                        ns=READ_ADD;
                end
                else begin
                        ns=IDLE;
                end
        READ_DATA:
                if (~SS_n) begin
                        ns=READ_DATA;
                end
                else begin
                        ns=IDLE;
                end
        default: ns=IDLE;
    endcase
end

endmodule
```

2.4 Slave Testbench Code

```
module SPIslavtb();

    reg MOSI, SS_n, tx_valid, clk, rst_n;
    wire MISO, rx_valid;
    reg [7:0] tx_data;
    wire [9:0] rx_data;

    SPIslav SPIslav1(MOSI, MISO, SS_n, clk, rst_n, rx_data, rx_valid, tx_data, tx_valid);

    initial begin
        clk=0;
        forever
        #1 clk=!clk;
    end


    initial begin
        rst_n=0;
        SS_n=1;
        tx_valid=0;
```

```
            MOSI=0;
            tx_data=0;
            #10;
            rst_n=1;
            SS_n=0;
            MOSI=1;
            #2;
            MOSI=0;//control
            #2;
            MOSI=0;//1
            #2;
            MOSI=0;
            #2;
            MOSI=1;
            #2;
            MOSI=1;
            #2;
            MOSI=0;//5
            #2;
            MOSI=0;//6
            #2;
            MOSI=0;//7
            #2;
            MOSI=0;//8
            #2;
            MOSI=1;//9
            #2;
            MOSI=1;//10
            #2;
            SS_n=1;
            MOSI=0;//
            #2;
            #2;
            $stop;
        end

endmodule
```

2.5 Wrapper Code

```
module SPIwrap(MOSI, SCK, SS_n, rst_n, MISO);


        input MOSI, SCK, SS_n, rst_n;

        output MISO;
```

```
        wire rx_valid, tx_valid;

        wire [7:0] tx_data;

        wire [9:0] rx_data;


        SPImem SPImem1(rx_data, rx_valid, SCK, rst_n, tx_data, tx_valid);

        SPIslav SPIslav1(MOSI, MISO, SS_n, SCK, rst_n, rx_data, rx_valid, tx_data, tx_valid);


endmodule
```

2.6 Final Testbench Code

```
module SPIwraptb();

        reg MOSI, SCK, SS_n, rst_n;
        wire MISO;
        reg [7:0] tempaddr;
        reg [7:0] tempdata;
        integer i=0;
        integer j=0;

        SPIwrap SPIwrap1(MOSI, SCK, SS_n, rst_n, MISO);

        initial begin
                SCK=0;
                forever
                #1 SCK=!SCK;
        end

        initial begin
                $readmemb("mem.dat",SPIwrap1.SPImem1.mem);
        end

        initial begin
                rst_n=0;
                SS_n=1;
                MOSI=0;
                #10;
                rst_n=1;
                for (i=0;i<1000;i=i+1) begin
                        SS_n=0;
                        MOSI=0;
```

```
#2
MOSI=0;//control
#2
MOSI=0;//1st
#2
MOSI=0;
#2
for (j=0;j<8;j=j+1) begin
        MOSI=$random;
        tempaddr={tempaddr[6:0], MOSI};
        #2;
end
SS_n=1;
#2
if (SPIwrap1.SPImem1.addr!=tempaddr) begin
        $monitor("rx address error");
        $stop;
end
SS_n=0;
MOSI=0;
#2
MOSI=0;//control
#2
MOSI=0;//1st
#2
MOSI=1;
#2
for (j=0;j<8;j=j+1) begin
        MOSI=$random;
        tempdata={tempdata, MOSI};
        #2;
end
SS_n=1;
#2;
if (SPIwrap1.SPImem1.mem[tempaddr]!=tempdata) begin
        $monitor("rx data error");
        $stop;
end
end
for (i=0;i<1000;i=i+1) begin
        SS_n=0;
        MOSI=0;
        #2
        MOSI=1;//control
        #2
        MOSI=1;//1st
```

```
                #2
                MOSI=0;
                #2
                for (j=0;j<8;j=j+1) begin
                        MOSI=$random;
                        tempaddr={tempaddr[6:0], MOSI};
                        #2;
                end
                SS_n=1;
                #2
                if (SPIwrap1.SPImem1.addr!=tempaddr) begin
                        $monitor("tx address error");
                        $stop;
                end
                SS_n=0;
                MOSI=0;
                #2
                MOSI=1;//control
                #2
                MOSI=1;//1st
                #2
                MOSI=1;
                #2
                for (j=0;j<8;j=j+1) begin
                        MOSI=$random;
                        #2;
                end
                #4;
                for (j=0;j<8;j=j+1) begin
                        tempdata={tempdata[6:0], MISO};
                        #2;
                end
                SS_n=1;
                #2;
                if (SPIwrap1.SPImem1.mem[tempaddr]!=tempdata) begin
                        $monitor("tx data error");
                        $stop;
                end
        end
        rst_n=0;
        #2;
        $stop;
    end

endmodule
```

2.7 Do File

```
vlib finalproject
vlog SPImem.v SPIslav.v SPIwrap.v SPIwraptb.v                +cover -covercells
vsim -voptargs=+acc work.SPIwraptb              -cover
add wave -position end  sim:/SPIwraptb/MOSI
add wave -position end  sim:/SPIwraptb/SCK
add wave -position end  sim:/SPIwraptb/SS_n
add wave -position end  sim:/SPIwraptb/rst_n
add wave -position end  sim:/SPIwraptb/MISO
add wave -position end  sim:/SPIwraptb/tempaddr
add wave -position end  sim:/SPIwraptb/tempdata
add wave -position end  sim:/SPIwraptb/SPIwrap1/rx_valid
add wave -position end  sim:/SPIwraptb/SPIwrap1/tx_valid
add wave -position end  sim:/SPIwraptb/SPIwrap1/tx_data
add wave -position end  sim:/SPIwraptb/SPIwrap1/rx_data
add wave -position end  sim:/SPIwraptb/SPIwrap1/SPIslav1/cs
add wave -position end  sim:/SPIwraptb/SPIwrap1/SPIslav1/ns
add wave -position end  sim:/SPIwraptb/SPIwrap1/SPIslav1/counter
add wave -position end  sim:/SPIwraptb/SPIwrap1/SPIslav1/addr_data
add wave -position end  sim:/SPIwraptb/SPIwrap1/SPImem1/addr
add wave -position end  sim:/SPIwraptb/SPIwrap1/SPImem1/mem

run -all
coverage save SPIwraptb.ucdb -onexit
vcover report SPIwraptb.ucdb
```