The process of the master in transmitting

- **Declaration of some parameters**
- **reg to keep the data to be sent to the slave, and another one to keep the data to be received from the slave.**
- **MOSI is the output form master to the slave.**
- **MISO is the output from the slave to the master.**
- **There are 4 states for the master.**
    1. **Default**
    2. **The clock is positive edge.**
    3. **The clock is negative edge.**
    4. **Data change is the last state.**

```
module SPI_Transmitter #(
    parameter SPI_DATALENGTH = 4'd8 //
)
(
    input clk,
    input rst,
    output reg sendComplete,
    output MOSI,
    output reg SCLK,
    output reg CS,
    input MISO,
    input [7:0] sendData,
    output reg [7:0] recvData
);
    reg [7:0] sendData_reg;
reg [7:0] recvData_reg;  //
    reg[4:0] counter;
    reg[1:0] state;
    localparam DEFSTATE   = 2'b00;
    localparam CLKUP      = 2'b01;
    localparam CLKDOWN    = 2'b10;
    localparam DATACHANGE = 2'b11;

    assign MOSI = sendData_reg[7];
    assign MISO = recvData_reg[7]; //
    reg[7:0] recvData_tmp;
    reg[7:0] sentData_tmp;        //
```

Explanation of the four states: -

Firstly, if we want to reset the modules, we will revert them to the default values and the status will be the default state.

1) <u>DEFSTATE</u>

a) **Slave clock will be at the negative edge.**
b) **Chip select will be low to make the slave start working.**
c) **Reg of receiving data and send data will remain as it.**
d) **State will be the next one which is Clock up.**

```
always @(posedge clk or negedge rst)begin
    if(rst == 1'b0)begin
        sendComplete <= 1'b0;
        recvData <= 8'b0000_0000;
        recvData_tmp <= 8'b0000_0000;
        SCLK     <= 1'b0;
        CS       <= 1'b1;
        counter  <= 4'b0000;
        sendData_reg <= sendData;
        state <= DEFSTATE;
    end else begin
        case (state)
            DEFSTATE : begin
                sendComplete <= 1'b0;
                SCLK  <= 1'b0;
                CS    <= 1'b0;
                state <= CLKUP;
                recvData_reg <= recvData;
                sendData_reg <= sendData;
                recvData_tmp <= recvData;
                sentData_tmp <= sendData;
            end
```

2) **CLKUP**

a) **Slave clock will be positive edge.**
b) **Chip select will be low to make the slave start working.**
c) **State will be the next one which is Clock down.**

```
CLKUP : begin
    sendComplete <= 1'b0;
    SCLK  <= 1'b1;
    CS    <= 1'b0;
    state <= CLKDOWN;
    recvData <= recvData;
    recvData_tmp <= recvData_tmp;
    sentData_tmp <= sentData_tmp;
end
```

3) **CLKDOWN**

a) **Slave clock will be negative edge.**
b) **Chip select will be low to make the slave start working.**
c) **The data in the master will be shifted left by input sent from the slave.**
d) **The data to be sent to the slave will be shifted left by the output from the master.**
e) **State will be the next one which is data change.**

```
CLKDOWN : begin
    sendComplete <= 1'b0;
    SCLK  <= 1'b0;
    CS    <= 1'b0;
    state <= DATACHANGE;
    recvData_tmp <= {recvData_tmp[6:0],MISO};
    sentData_tmp <= {sentData_tmp[6:0],MOSI};
    recvData <= recvData;
end
```

4) **DATACHANGE**

a) **Slave clock will remain as it.**
b) **If all the data in the master is transmitted to the slave (counter = 8), complete will be set as 1 (true) and chip select of the slave will be high to stop receiving.**
c) **If not, we will copy data from the temporary register of sentData to the main reg.**
d) **State will be the next one which is CLKUP.**
e) **Counter will be incremented.**

```
DATACHANGE : begin
    SCLK <= SCLK;
    recvData_tmp <= recvData_tmp;
    sentData_tmp <= sentData_tmp;
    if (counter == SPI_DATALENGTH - 4'b0001)begin
        sendComplete <= 1'b1;
        CS           <= 1'b1;
        state        <= DATACHANGE;
        counter      <= counter;
        recvData     <= recvData_tmp;
    end else begin
        sendComplete <= 1'b0;
        sendData_reg <= sentData_tmp;
        recvData_reg <= recvData_tmp;
        state        <= CLKUP;
        counter      <= counter + 4'b0001;
        recvData     <= recvData;
    end
end
```

Master Module

- **Declaration of some parameters**
- **Reg for the current state and the next state.**
- **Reg for the starting signal**
- **Reg to determine which transmitting process completed or not.**
- **There are 3 states for the master.**
  - **(1) Initialization.**
  - **(2) Sending data to the slave.**
  - **(3) Final state at the final clock.**
- **Calling for the transmitting module to transmit data from master to slave.**

```
module Master(

input clk,
      rst,
      sendStart,
input[7:0] sendData,
output [7:0] recvData,
output SPI_SCLK,
[2:0]  SPI_CS,
output SPI_MOSI,
input  SPI_MISO
);


    localparam STATE_INITIALIZE  = 2'b00;
    localparam STATE_SENDMSG     = 2'b01;
    localparam STATE_FINALIZE    = 2'b11;
    parameter SPI_DATALENGTH = 4'd8;

    reg [1:0] currentState;
    reg [1:0] nextState;
    reg sendStart_reg;
    wire sendComplete;


    SPI_Transmitter #(SPI_DATALENGTH) transmitter(
        .clk(clk),
        .rst(sendStart_reg),
        .sendComplete(sendComplete),
        .MOSI(SPI_MOSI),
        .SCLK(SPI_SCLK),
        .CS(SPI_CS),
        .MISO(SPI_MISO),
        .sendData(sendData),
        .recvData(recvData)
    );
```

Explanation of the three states: -

Firstly, if we want to reset the modules, we will revert them to the default values and the status will be Initialization.

1) **STATE_INTITIALIZE**
   - **If the signal is high, we will go to the next state.**
   - **If not, we will stay in the current state.**

```
always @ (posedge clk or negedge rst) begin
    if(rst == 1'b0)begin
        currentState <= STATE_INITIALIZE;
        sendStart_reg <= 1'b0;
    end else begin
        case (currentState)
            STATE_INITIALIZE : begin
                if(sendStart == 1'b1)begin
                    currentState <= STATE_SENDMSG;
                    sendStart_reg <= 1'b1;
                end else begin
                    currentState <= currentState;
                    sendStart_reg <= 1'b0;
                end
            end
```

2) **STATE_SENDMSG**

- **If the start signal is high, we will go to the final state and set the register of the signal high.**
- **If not, we will remain at the current state.**

```
STATE_SENDMSG : begin
    if(sendComplete == 1'b1) begin
        currentState <= STATE_FINALIZE;
        sendStart_reg <= 1'b1;
    end else begin
        currentState <= currentState;
        sendStart_reg <= 1'b1;
    end
end
```

3) **STATE_FINALIZE.**

- **If the start signal is low, the state will be the first on (Initialization), and the signal remains as it.**
- **If not, the signal will be high.**

```
STATE_FINALIZE : begin
    if(sendStart == 1'b0)begin
        currentState <= STATE_INITIALIZE;
        sendStart_reg <= 1'b0;
    end else begin
        currentState <= currentState;
        sendStart_reg <= 1'b1;
    end
end
endcase
```

The process of the slave in receiving.

- **Declaration of some parameters**
- **reg to keep the data to be sent to the Master, and another one to keep the data to be received from the master.**
- **MOSI is the output form master to the slave.**
- **MISO is the output from the slave to the master.**
- **There are 4 states for the slave.**
    1) **Default**
    2) **The clock is positive edge.**
    3) **The clock is negative edge.**
    4) **Data change is the last state.**

```
module SPI_Transmitter22 #(
    parameter SPI_DATALENGTH = 4'd8 //
)
(
    input clk,
    input rst,
    output reg sendComplete,
    output MISO,
    output reg SCLK,
    output reg CS,
    input MOSI,
    input [7:0] sendData,
    output reg [7:0] recvData
);
    reg [7:0] sendData_reg;
    reg [7:0] recvData_reg;  //
    reg[4:0] counter;
    reg[1:0] state;
    localparam DEFSTATE   = 2'b00;
    localparam CLKUP      = 2'b01;
    localparam CLKDOWN    = 2'b10;
    localparam DATACHANGE = 2'b11;

    assign MISO = sendData_reg[7];
    assign MOSI = recvData_reg[7]; //
    reg[7:0] recvData_tmp;
    reg[7:0] sentData_tmp;        //
```

Explanation of the four states: -

Firstly, if we want to reset the modules, we will revert them to the default values and the status will be the default state.

1) **DEFSTATE**

    a. **Slave clock will be at the negative edge.**
    b. **Chip select will be low to make the slave start working.**
    c. **Reg of receiving data and send data will remain as it.**
    d. **State will be the next one which is Clock up.**

```
always @(posedge clk or negedge rst)begin
    if(rst == 1'b0)begin
        sendComplete <= 1'b0;
        recvData <= 8'b0000_0000;
        recvData_tmp <= 8'b0000_0000;
        SCLK     <= 1'b0;
        CS       <= 1'b1;
        counter  <= 4'b0000;
        sendData_reg <= sendData; // default
        state <= DEFSTATE;
    end else begin
        case (state)
            DEFSTATE : begin
                sendComplete <= 1'b0;
                SCLK  <= 1'b0;
                CS    <= 1'b0;
                state <= CLKUP;
                recvData_reg <= recvData;
                sendData_reg <= sendData;
                recvData_tmp <= recvData;
                sentData_tmp <= sendData;
            end
```

2) **CLKUP**

a. **Slave clock will be positive edge.**
b. **Chip select will be low to make the slave start working.**
c. **State will be the next one which is Clock down.**

```
CLKUP : begin
    sendComplete <= 1'b0;
    SCLK  <= 1'b1;
    CS    <= 1'b0;
    state <= CLKDOWN;
    recvData <= recvData;          //
    recvData_tmp <= recvData_tmp;
    sentData_tmp <= sentData_tmp; //
end
```

3) **CLKDOWN**

a. **Slave clock will be negative edge.**
b. **Chip select will be low to make the slave start working.**
c. **The data in the slave will be shifted left by input sent from the master.**
d. **The data to be sent to the master will be shifted left by the output from the slave.**
e. **State will be the next one which is data change.**

```
CLKDOWN : begin
    sendComplete <= 1'b0;
    SCLK  <= 1'b0;
    CS    <= 1'b0;
    state <= DATACHANGE;
    recvData_tmp <= {recvData_tmp[6:0],MISO};
    sentData_tmp <= {sentData_tmp[6:0],MOSI};
    recvData <= recvData;
end
```

4) **DATACHANGE**

a. **Slave clock will remain as it.**
b. **If all the data in the slave is transmitted to the master (counter = 8), complete will be set as 1 (true) and chip select of the slave will be high to stop receiving.**
c. **If not, we will copy data from the temporary register of sentData to the main reg.**
d. **State will be the next one which is CLKUP.**
e. **Counter will be incremented.**

```
DATACHANGE : begin
    SCLK <= SCLK;
    recvData_tmp <= recvData_tmp;
    sentData_tmp <= sentData_tmp;
    if (counter == SPI_DATALENGTH - 4'b0001)begin
        sendComplete <= 1'b1;
        CS           <= 1'b1;
        state        <= DATACHANGE;
        counter      <= counter;
        recvData     <= recvData_tmp;
    end else begin
        sendComplete <= 1'b0;
        sendData_reg <= sentData_tmp;
        recvData_reg <= recvData_tmp;
        state        <= CLKUP;
        counter      <= counter + 4'b0001;
        recvData     <= recvData;
    end
end
```

Slave Module

- **Declaration of some parameters**
- **Reg for the current state and the next state.**
- **Reg for the starting signal**
- **Reg to determine which transmitting process completed or not.**
- **There are 3 states for the master.**
  - **(1) Initialization.**
  - **(2) Sending data to the master.**
  - **(3) Final state at the final coc.**
- **Calling for the transmitting module to transmit data from slave to master.**

```
module Slave(

input  rst,
input[7:0] sendData,
output  [7:0] recvData,
output SCLK,
output  [2:0]  CS,
input MOSI,
output  MISO
);

    localparam STATE_INITIALIZE  = 2'b00;
    localparam STATE_SENDMSG     = 2'b01;
    localparam STATE_FINALIZE    = 2'b11;
    parameter SPI_DATALENGTH = 4'd8;

    reg clk;
    reg [1:0] currentState;
    reg [1:0] nextState;
    reg sendStart_reg;
    wire sendComplete;
    wire sendStart;

    SPI_Transmitter22 #(SPI_DATALENGTH) transmitter5(
        .clk(clk),
        .rst(sendStart_reg),
        .sendComplete(sendComplete),
        .MISO(MISO),
        .SCLK(SCLK),
        .CS(CS),
        .MOSI(MOSI),
        .sendData(sendData),
        .recvData(recvData)
    );
```

Explanation of the three states: -

Firstly, if we want to reset the modules, we will revert them to the default values and the status will be Initialization.

1) STATE_INTITIALIZE

- **If the signal is high, we will go to the next state.**
- **If not, we will stay in the current state.**

```
always @ (posedge clk or negedge rst) begin
    if(rst == 1'b0)begin
        currentState <= STATE_INITIALIZE;
        sendStart_reg <= 1'b0;
    end else begin
        case (currentState)
            STATE_INITIALIZE : begin
                if(sendStart == 1'b1)begin
                    currentState <= STATE_SENDMSG;
                    sendStart_reg <= 1'b1;
                end else begin
                    currentState <= currentState;
                    sendStart_reg <= 1'b0;
                end
            end
```

2) STATE_SENDMSG

- **If the start signal is high, we will go to the final state and set the register of the signal high.**
- **If not, we will remain at the current state.**

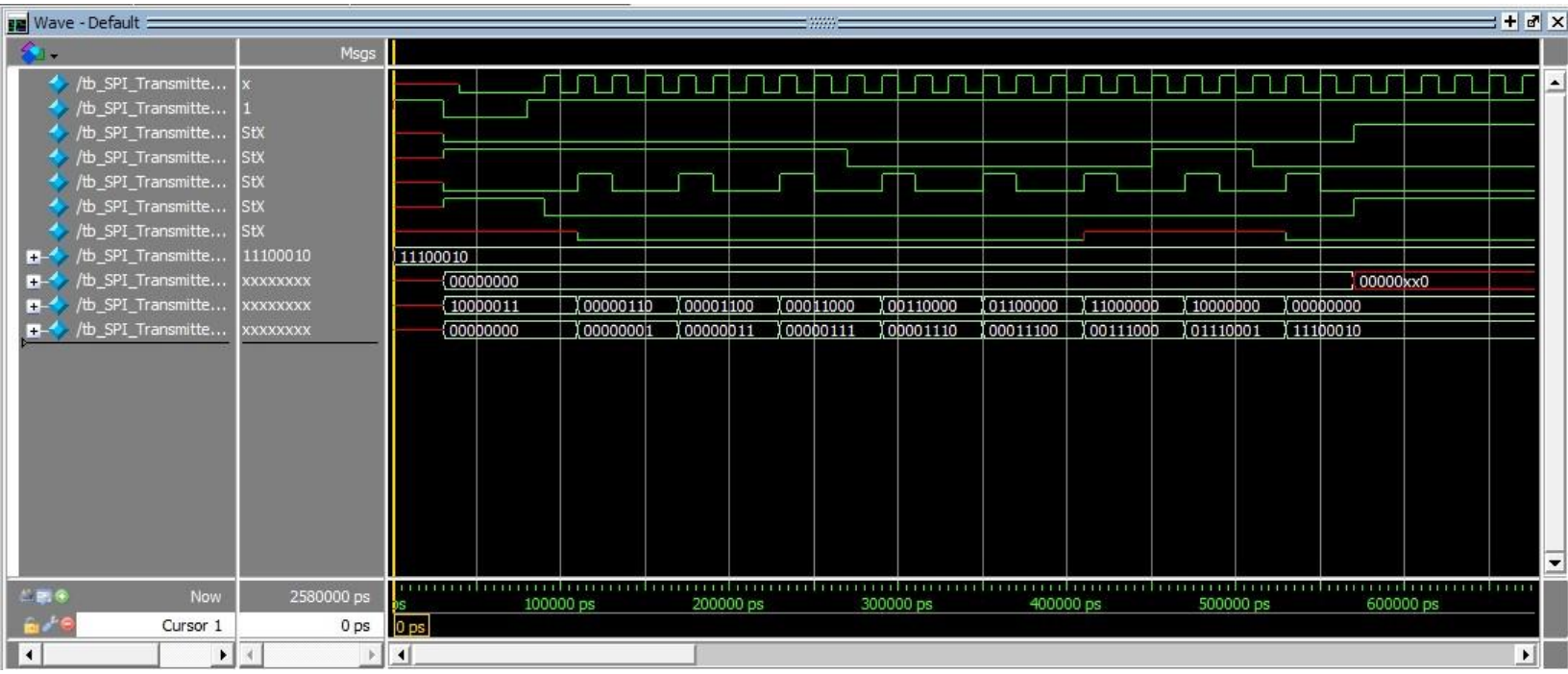
```
STATE_SENDMSG : begin
    if(sendComplete == 1'b1) begin
        currentState <= STATE_FINALIZE;
        sendStart_reg <= 1'b1;
    end else begin
        currentState <= currentState;
        sendStart_reg <= 1'b1;
    end
end
```

3-STATE_FINALIZE.

- **If the start signal is low, the state will be the first on (Initialization), and the signal remains as it.**
- **If not, the signal will be high.**

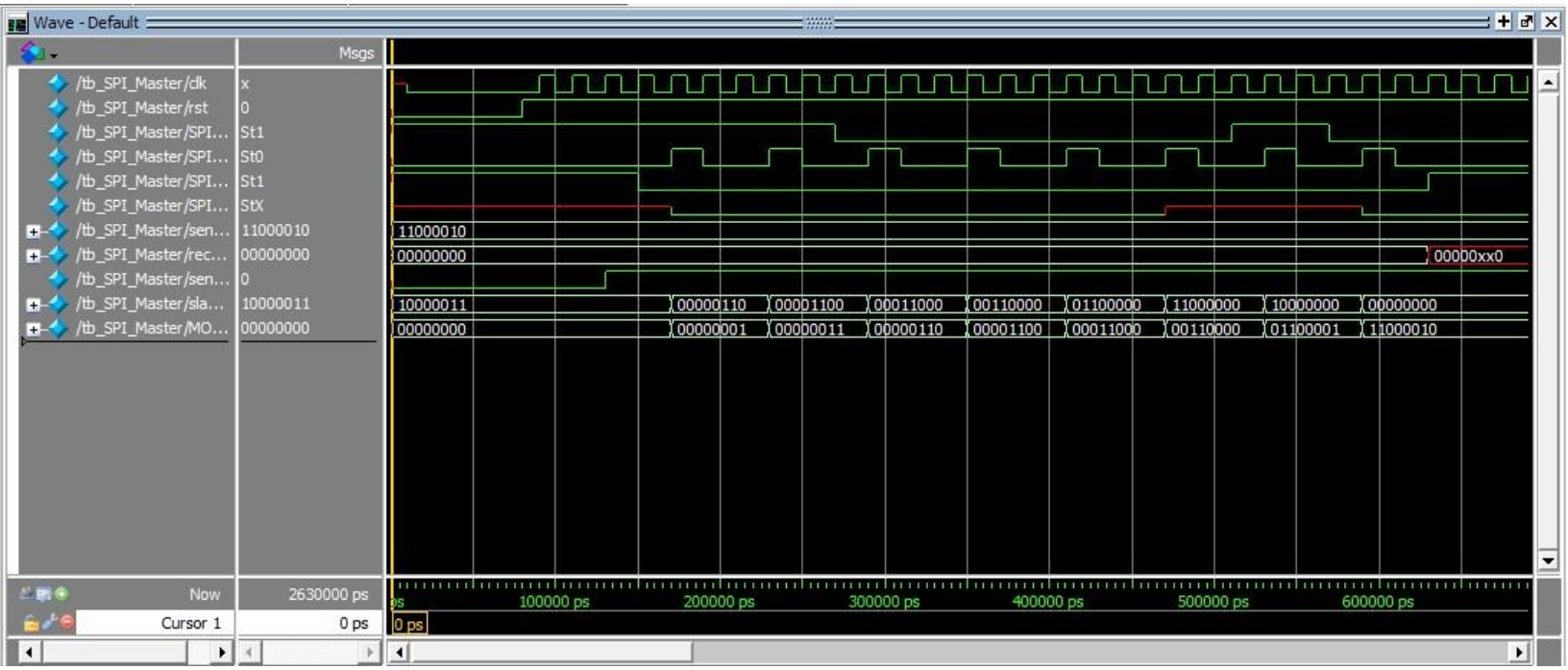
```
STATE_FINALIZE : begin
    if(sendStart == 1'b0)begin
        currentState <= STATE_INITIALIZE;
        sendStart_reg <= 1'b0;
    end else begin
        currentState <= currentState;
        sendStart_reg <= 1'b1;
    end
end
```

Transmitter from Master to Slave Test Bench

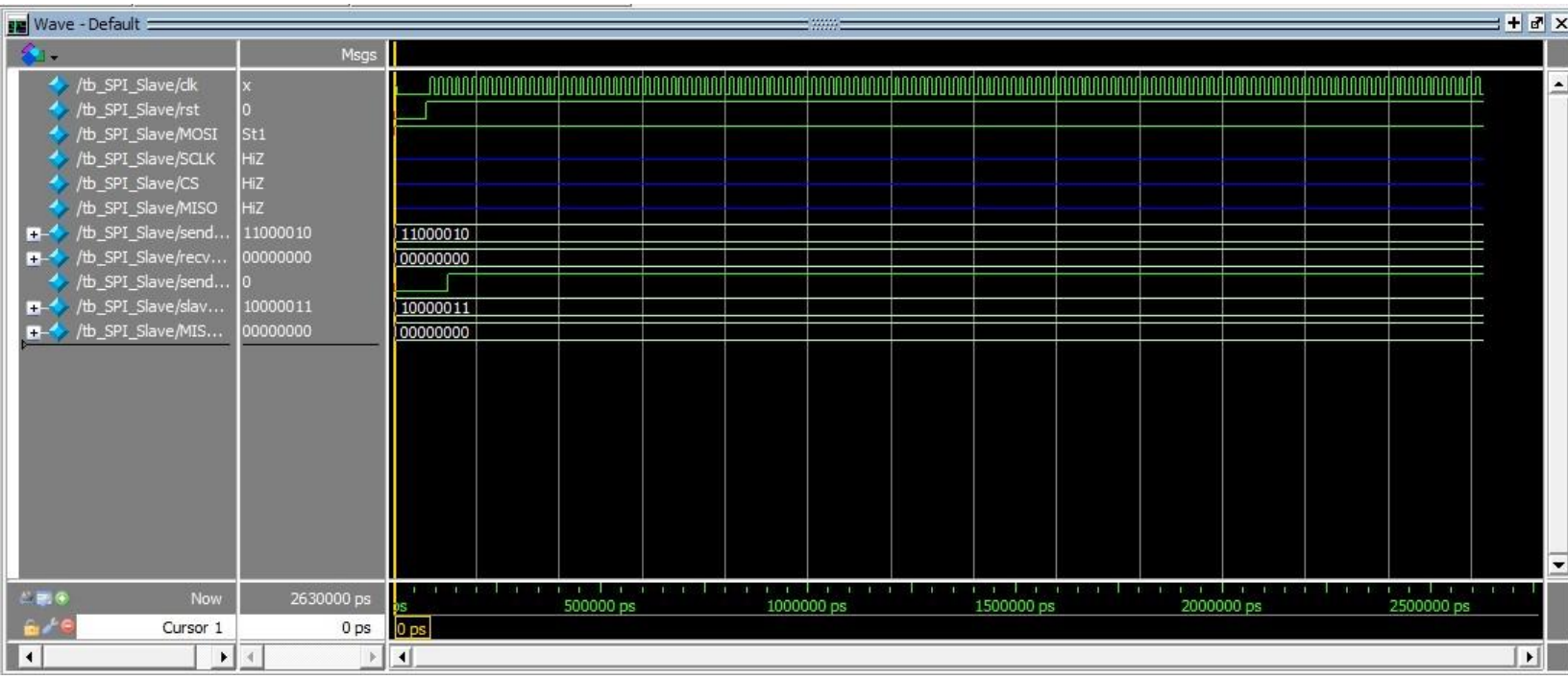


Transmitter from Slave to Master Test Bench

Master Test Bench



Slave Test Bench



Transmitter from Master to Slave Output

```
VSIM 12> run -all
# Before Transmitting --> Data to send : 11100010, MOSI_REG : 00000000
# After Transmitting --> Data to send : 00000000, MOSI_REG : 11100010
# ** Note: $finish    : E:/work/master_transmit_tb.v(59)
#    Time: 2580 ns  Iteration: 0  Instance: /tb_SPI_Transmitter
# 1
# Break in Module tb_SPI_Transmitter at E:/work/master_transmit_tb.v line 59
```

Transmitter from Slave to Master Output

```
VSIM 20> run -all
# Start Transmitting --> Data to send : 10100010, MISO_Reg : 00000000
# End Transmitting --> Data to send : 00000000, MISO_Reg : 10100010
# ** Note: $finish    : E:/work/slave_transmit_tb.v(59)
#    Time: 2580 ns  Iteration: 0  Instance: /tb_SPI_Transmitter22
# 1
# Break in Module tb_SPI_Transmitter22 at E:/work/slave_transmit_tb.v line 59
```

Master Output

```
VSIM 5> run -all
# Begin --> Data to send : 11000010, MOSI_REG : 00000000
# End --> Data to send : 00000000, MOSI_REG : 11000010
# ** Note: $finish    : E:/work/master_tb.v(67)
#    Time: 2630 ns  Iteration: 0  Instance: /tb_SPI_Master
# 1
# Break in Module tb_SPI_Master at E:/work/master_tb.v line 67
```