SPI Communication Protocol

TIMELINE

01

Project definition

02

Architecture and Logic flow

03

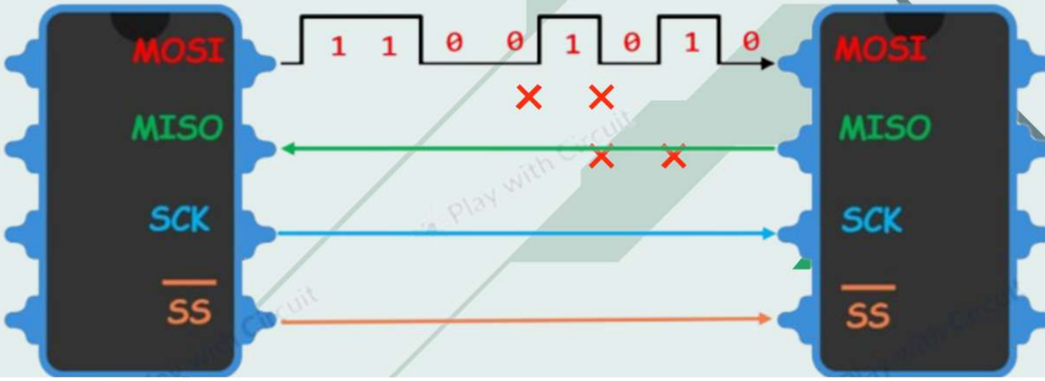
Results

01

Project definition

Project definition

- Serial Peripheral Interface, is a synchronous serial communication protocol used for short-distance, high-speed data transfer between microcontrollers and peripheral devices
- It's known for its simplicity and full-duplex capability, allowing simultaneous data transmission and reception

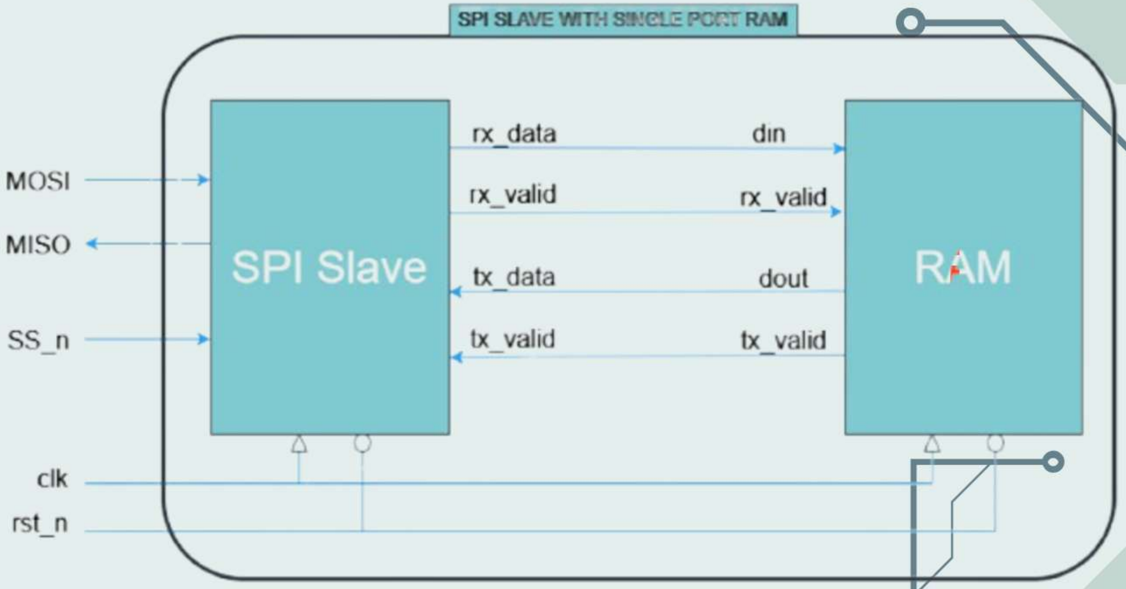


02

Architecture and logic flow

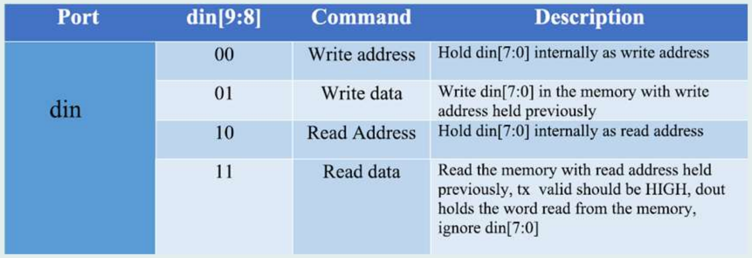
System Architecture

- This is the architecture for SPI_WRAPPER, which combines both the SPI_SLAVE and single port RAM.
- This allows the Master to send data to the Slave, and the Slave stores it in the RAM.
- Conversely, when the Slave sends data to the Master, it fetch the data from the memory location sent by the Master and sends it to the Master.



Flow

1) Master activates the slave by sending ‘0’ to the SS_n (slave select) signal to enable the slave and mark the start of the communication.
2) Master sends a 10-bit word, serially, through the MOSI line. The most significant 2 bits are the command bits and then 8 bits data. Then the 10 bits data are sent in parallel in the rx_data[9:0] and rx_valid is enabled to inform the RAM to accept din[9:0] as an input.
3) The RAM checks the command bits, din[9:8], to select the operation that will be done according to the following table



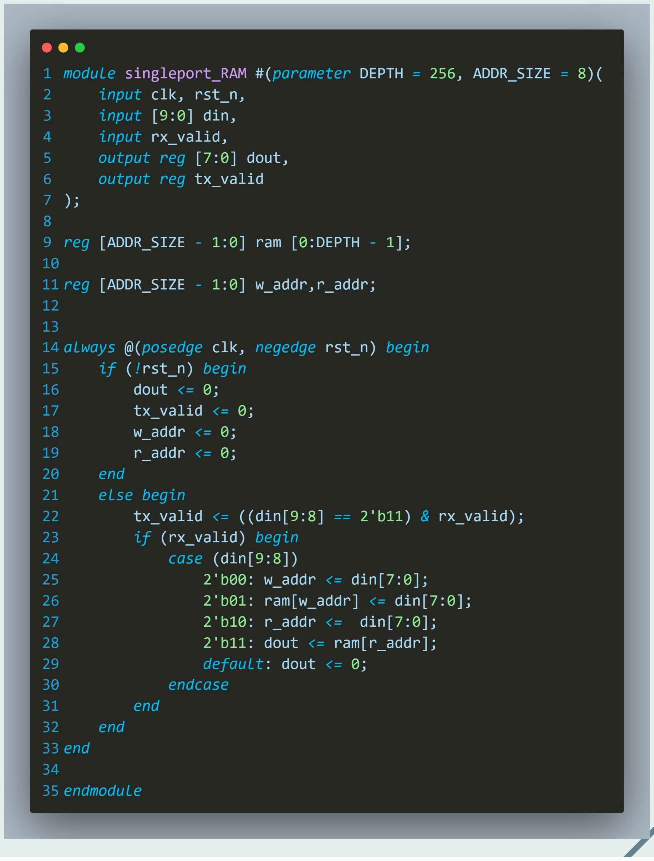
Port	din[9:8]	Command	Description
din	00	Write address	Hold din[7:0] internally as write address
	01	Write data	Write din[7:0] in the memory with write address held previously
	10	Read Address	Hold din[7:0] internally as read address
	11	Read data	Read the memory with read address held previously, tx valid should be HIGH, dout holds the word read from the memory, ignore din[7:0]

4) Now according to the command, it will define whether it’s a read or write operation.

RTL

RAM

- Asynchronous RAM for storing data

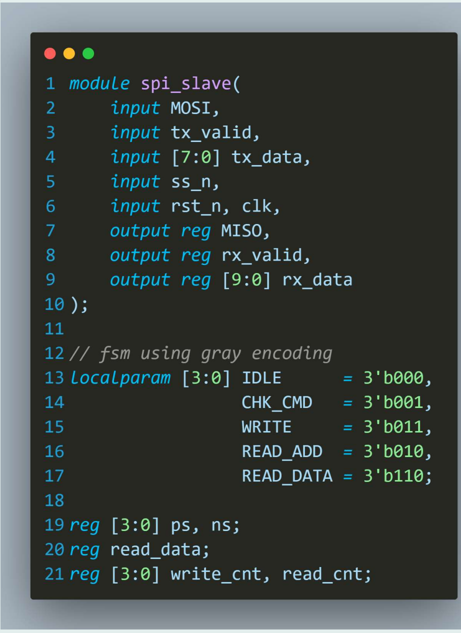


```verilog
module singleport_RAM #(parameter DEPTH = 256, ADDR_SIZE = 8)(
    input clk, rst_n,
    input [9:0] din,
    input rx_valid,
    output reg [7:0] dout,
    output reg tx_valid
);

reg [ADDR_SIZE - 1:0] ram [0:DEPTH - 1];

reg [ADDR_SIZE - 1:0] w_addr,r_addr;


always @(posedge clk, negedge rst_n) begin
    if (!rst_n) begin
        dout <= 0;
        tx_valid <= 0;
        w_addr <= 0;
        r_addr <= 0;
    end
    else begin
        tx_valid <= ((din[9:8] == 2'b11) & rx_valid);
        if (rx_valid) begin
            case (din[9:8])
                2'b00: w_addr <= din[7:0];
                2'b01: ram[w_addr] <= din[7:0];
                2'b10: r_addr <=  din[7:0];
                2'b11: dout <= ram[r_addr];
                default: dout <= 0;
            endcase
        end
    end
end

endmodule
```

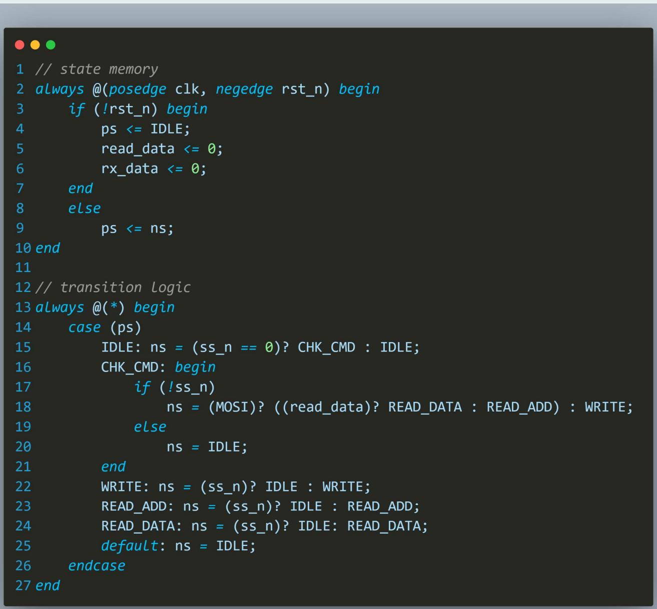
RTL

SPI_SLAVE

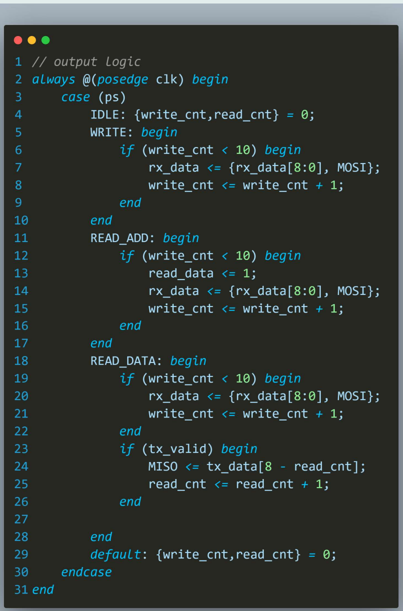
- Main module responsible for communication between Master and Slave
- Determines the state of operation (FSM)



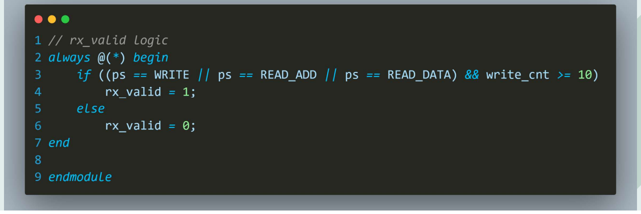
```
module spi_slave(
    input MOSI,
    input tx_valid,
    input [7:0] tx_data,
    input ss_n,
    input rst_n, clk,
    output reg MISO,
    output reg rx_valid,
    output reg [9:0] rx_data
);

// fsm using gray encoding
localparam [3:0] IDLE      = 3'b000,
                 CHK_CMD   = 3'b001,
                 WRITE     = 3'b011,
                 READ_ADD  = 3'b010,
                 READ_DATA = 3'b110;

reg [3:0] ps, ns;
reg read_data;
reg [3:0] write_cnt, read_cnt;
```



```
// state memory
always @(posedge clk, negedge rst_n) begin
    if (!rst_n) begin
        ps <= IDLE;
        read_data <= 0;
        rx_data <= 0;
    end
    else
        ps <= ns;
end

// transition logic
always @(*) begin
    case (ps)
        IDLE: ns = (ss_n == 0)? CHK_CMD : IDLE;
        CHK_CMD: begin
            if (!ss_n)
                ns = (MOSI)? ((read_data)? READ_DATA : READ_ADD) : WRITE;
            else
                ns = IDLE;
        end
        WRITE: ns = (ss_n)? IDLE : WRITE;
        READ_ADD: ns = (ss_n)? IDLE : READ_ADD;
        READ_DATA: ns = (ss_n)? IDLE: READ_DATA;
        default: ns = IDLE;
    endcase
end
```



```
// output logic
always @(posedge clk) begin
    case (ps)
        IDLE: {write_cnt,read_cnt} = 0;
        WRITE: begin
            if (write_cnt < 10) begin
                rx_data <= {rx_data[8:0], MOSI};
                write_cnt <= write_cnt + 1;
            end
        end
        READ_ADD: begin
            if (write_cnt < 10) begin
                read_data <= 1;
                rx_data <= {rx_data[8:0], MOSI};
                write_cnt <= write_cnt + 1;
            end
        end
        READ_DATA: begin
            if (write_cnt < 10) begin
                rx_data <= {rx_data[8:0], MOSI};
                write_cnt <= write_cnt + 1;
            end
            if (tx_valid) begin
                MISO <= tx_data[8 - read_cnt];
                read_cnt <= read_cnt + 1;
            end

        end
        default: {write_cnt,read_cnt} = 0;
    endcase
end
```

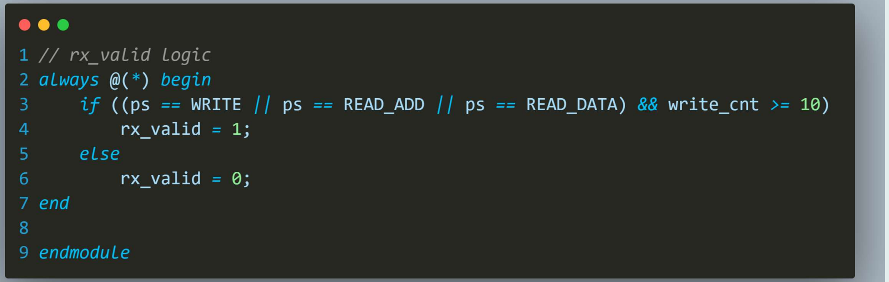


```
// rx_valid logic
always @(*) begin
    if ((ps == WRITE || ps == READ_ADD || ps == READ_DATA) && write_cnt >= 10)
        rx_valid = 1;
    else
        rx_valid = 0;
end

endmodule
```

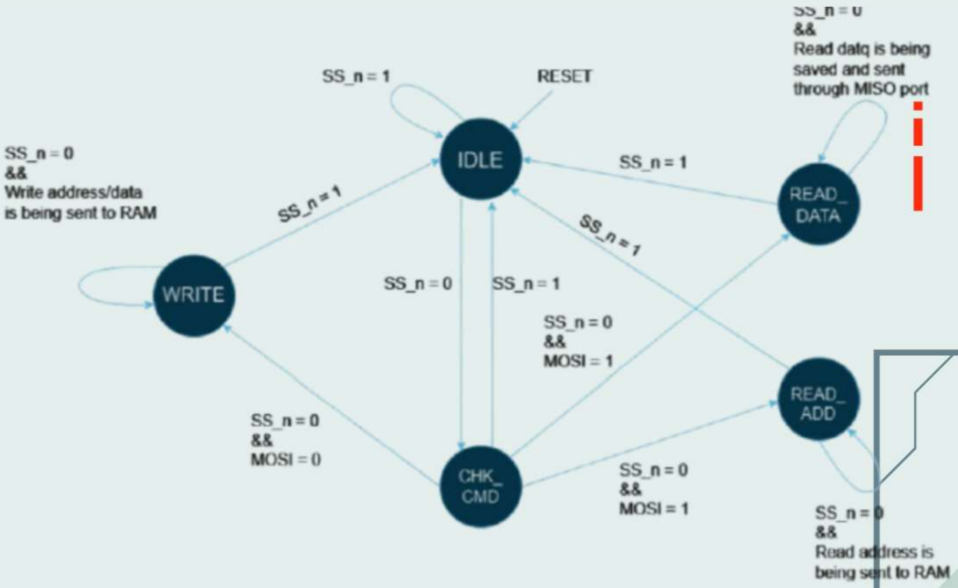
RTL

SPI_SLAVE

- Main module responsible for communication between Master and Slave
- Determines the state of operation (FSM)



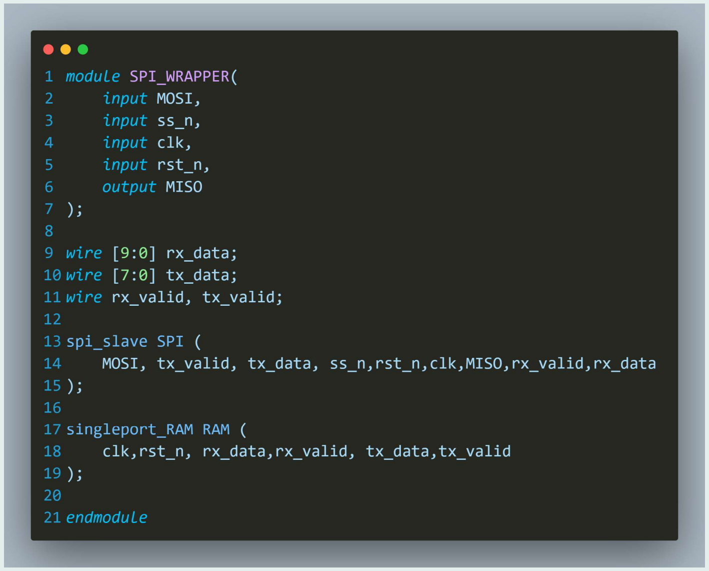
```
// rx_valid logic
always @(*) begin
    if ((ps == WRITE || ps == READ_ADD || ps == READ_DATA) && write_cnt >= 10)
        rx_valid = 1;
    else
        rx_valid = 0;
end

endmodule
```



RTL

SPI_WRAPPER

- Combining SPI_SLAVE and RAM

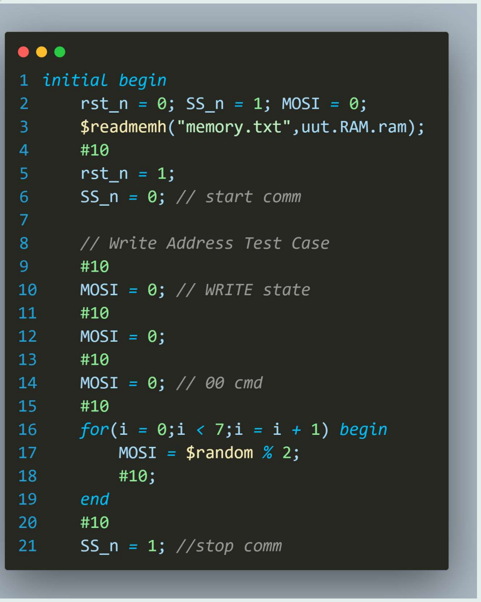


```
module SPI_WRAPPER(
    input MOSI,
    input ss_n,
    input clk,
    input rst_n,
    output MISO
);

wire [9:0] rx_data;
wire [7:0] tx_data;
wire rx_valid, tx_valid;

spi_slave SPI (
    MOSI, tx_valid, tx_data, ss_n,rst_n,clk,MISO,rx_valid,rx_data
);

singleport_RAM RAM (
    clk,rst_n, rx_data,rx_valid, tx_data,tx_valid
);

endmodule
```

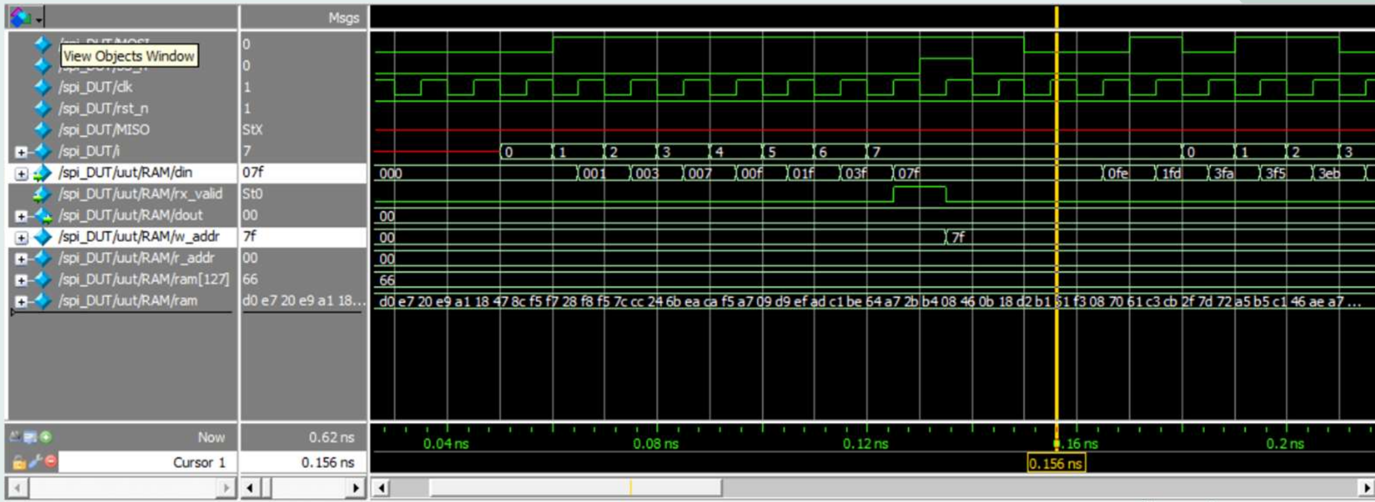
03

Results

Write Address



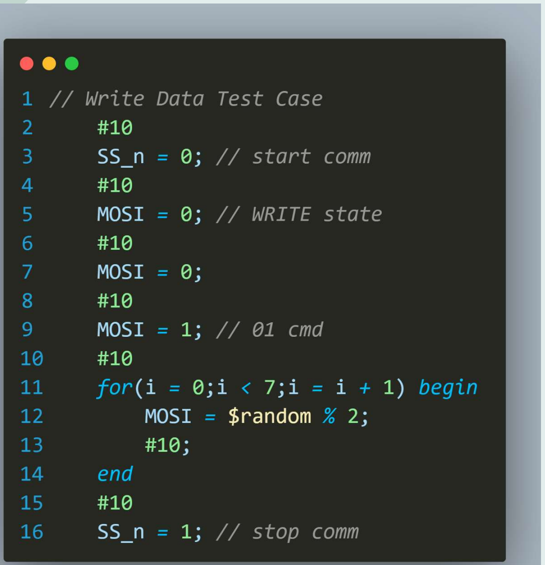
```
initial begin
    rst_n = 0; SS_n = 1; MOSI = 0;
    $readmemh("memory.txt",uut.RAM.ram);
    #10
    rst_n = 1;
    SS_n = 0; // start comm

    // Write Address Test Case
    #10
    MOSI = 0; // WRITE state
    #10
    MOSI = 0;
    #10
    MOSI = 0; // 00 cmd
    #10
    for(i = 0;i < 7;i = i + 1) begin
        MOSI = $random % 2;
        #10;
    end
    #10
    SS_n = 1; //stop comm
```



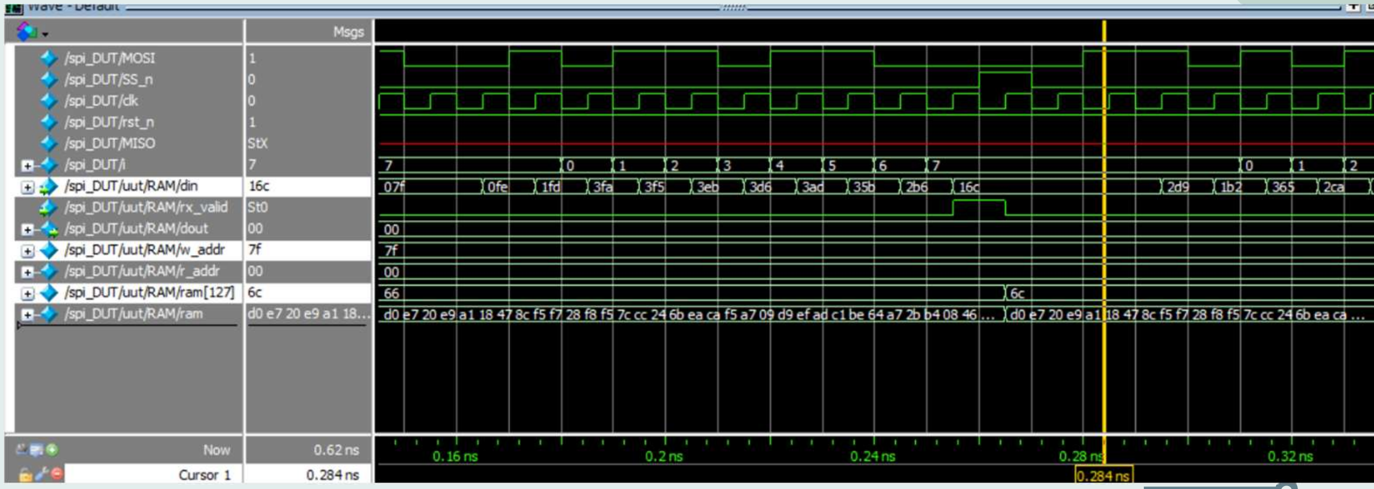
Msgs
View Objects Window
/spi_DUT/MOSI 0
/spi_DUT/SS_n 0
/spi_DUT/clk 1
/spi_DUT/rst_n 1
/spi_DUT/MISO StX
/spi_DUT/i 7
/spi_DUT/uut/RAM/din 07f
/spi_DUT/uut/RAM/rx_valid St0
/spi_DUT/uut/RAM/dout 00
/spi_DUT/uut/RAM/w_addr 7f
/spi_DUT/uut/RAM/r_addr 00
/spi_DUT/uut/RAM/ram[127] 66
/spi_DUT/uut/RAM/ram d0 e7 20 e9 a1 18...
0 1 2 3 4 5 6 7 0 1 2 3
000 001 003 007 00f 01f 03f 07f 0fe 1fd 3fa 3f5 3eb
00
00 7f
00
66
d0 e7 20 e9 a1 18 47 8c f5 f7 28 f8 f5 7c cc 24 6b ea ca f5 a7 09 d9 ef ad c1 be 64 a7 2b b4 08 46 0b 18 d2 b1 51 f3 08 70 61 c3 cb 2f 7d 72 a5 b5 c1 46 ae a7 ...
Now 0.62 ns
Cursor 1 0.156 ns
0.04 ns
0.08 ns
0.12 ns
0.16 ns
0.2 ns
0.156 ns

Wrote 0x7F (0111_1111) from MOSI line in the write address

Write Data



```
// Write Data Test Case
    #10
    SS_n = 0; // start comm
    #10
    MOSI = 0; // WRITE state
    #10
    MOSI = 0;
    #10
    MOSI = 1; // 01 cmd
    #10
    for(i = 0;i < 7;i = i + 1) begin
        MOSI = $random % 2;
        #10;
    end
    #10
    SS_n = 1; // stop comm
```



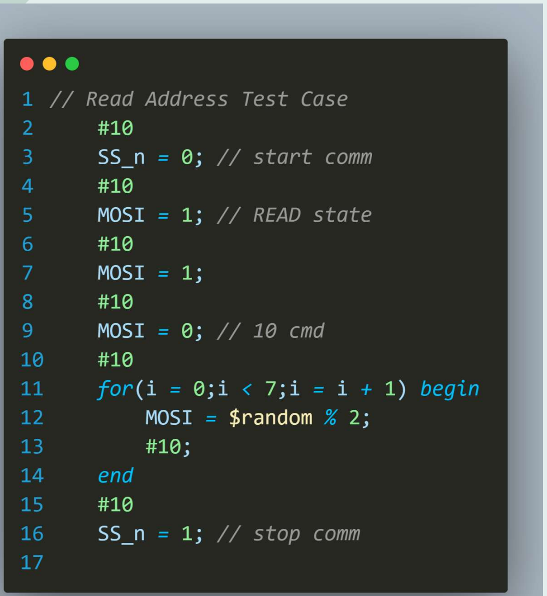
Wave - Default

	Msgs
/spi_DUT/MOSI	1
/spi_DUT/SS_n	0
/spi_DUT/clk	0
/spi_DUT/rst_n	1
/spi_DUT/MISO	StX
/spi_DUT/i	7
/spi_DUT/uut/RAM/din	16c
/spi_DUT/uut/RAM/rx_valid	St0
/spi_DUT/uut/RAM/dout	00
/spi_DUT/uut/RAM/w_addr	7f
/spi_DUT/uut/RAM/r_addr	00
/spi_DUT/uut/RAM/ram[127]	6c
/spi_DUT/uut/RAM/ram	d0 e7 20 e9 a1 18...
Now	0.62 ns
Cursor 1	0.284 ns

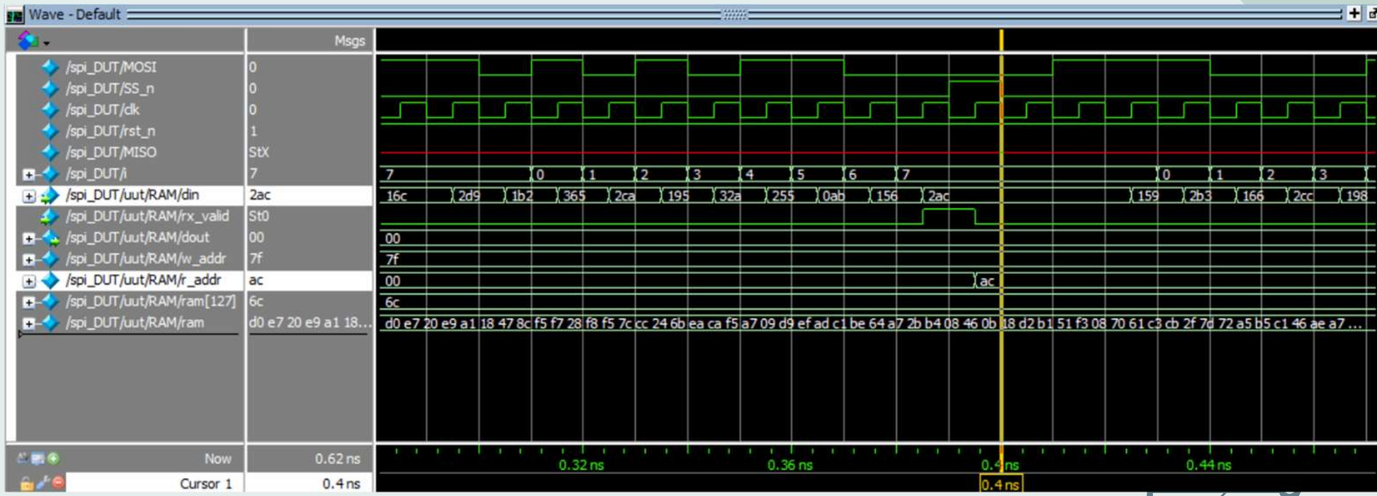
7
0
1
2
3
4
5
6
7
0
1
2
07f
0fe
1fd
3fa
3f5
3eb
3d6
3ad
35b
2b6
16c
2d9
1b2
365
2ca
00
7f
00
66
6c
d0 e7 20 e9 a1 18 47 8c f5 f7 28 f8 f5 7c cc 24 6b ea ca f5 a7 09 d9 ef ad c1 be 64 a7 2b b4 08 46 ...
d0 e7 20 e9 a1 18 47 8c f5 f7 28 f8 f5 7c cc 24 6b ea ca ...
0.16 ns
0.2 ns
0.24 ns
0.28 ns
0.32 ns
0.284 ns

Wrote 0x6C on ram[127, 7f]

Read Address



```
// Read Address Test Case
    #10
    SS_n = 0; // start comm
    #10
    MOSI = 1; // READ state
    #10
    MOSI = 1;
    #10
    MOSI = 0; // 10 cmd
    #10
    for(i = 0;i < 7;i = i + 1) begin
        MOSI = $random % 2;
        #10;
    end
    #10
    SS_n = 1; // stop comm
```

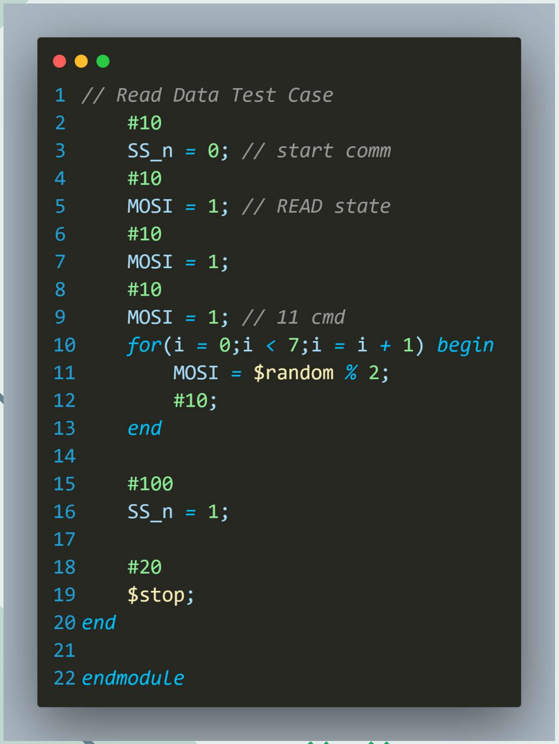


Wave - Default

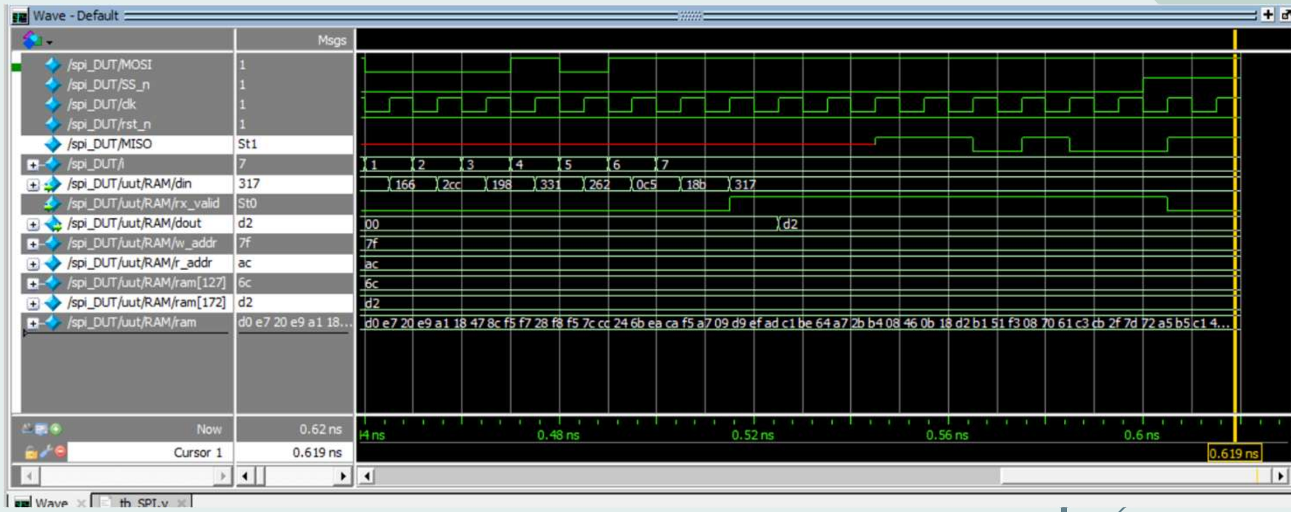
	Msgs
/spi_DUT/MOSI	0
/spi_DUT/SS_n	0
/spi_DUT/clk	0
/spi_DUT/rst_n	1
/spi_DUT/MISO	StX
/spi_DUT/i	7
/spi_DUT/uut/RAM/din	2ac
/spi_DUT/uut/RAM/rx_valid	St0
/spi_DUT/uut/RAM/dout	00
/spi_DUT/uut/RAM/w_addr	7f
/spi_DUT/uut/RAM/r_addr	ac
/spi_DUT/uut/RAM/ram[127]	6c
/spi_DUT/uut/RAM/ram	d0 e7 20 e9 a1 18...
Now	0.62 ns
Cursor 1	0.4 ns

7
0
1
2
3
4
5
6
7
0
1
2
3
16c
2d9
1b2
365
2ca
195
32a
255
0ab
156
2ac
159
2b3
166
2cc
198
00
7f
00
ac
6c
d0 e7 20 e9 a1 18 47 8c f5 f7 28 f8 f5 7c cc 24 6b ea ca f5 a7 09 d9 ef ad c1 be 64 a7 2b b4 08 46 0b 18 d2 b1 51 f3 08 70 61 c3 cb 2f 7d 72 a5 b5 c1 46 ae a7 ...
0.32 ns
0.36 ns
0.4 ns
0.44 ns
0.4 ns

Read Data



```
// Read Data Test Case
    #10
    SS_n = 0; // start comm
    #10
    MOSI = 1; // READ state
    #10
    MOSI = 1;
    #10
    MOSI = 1; // 11 cmd
    for(i = 0;i < 7;i = i + 1) begin
        MOSI = $random % 2;
        #10;
    end

    #100
    SS_n = 1;

    #20
    $stop;
end

endmodule
```



```
Wave - Default
                                    Msgs
/spi_DUT/MOSI                       1
/spi_DUT/SS_n                       1
/spi_DUT/clk                        1
/spi_DUT/rst_n                      1
/spi_DUT/MISO                       St1
/spi_DUT/i                          7
/spi_DUT/uut/RAM/din                317
/spi_DUT/uut/RAM/rx_valid           St0
/spi_DUT/uut/RAM/dout               d2
/spi_DUT/uut/RAM/w_addr             7f
/spi_DUT/uut/RAM/r_addr             ac
/spi_DUT/uut/RAM/ram[127]           6c
/spi_DUT/uut/RAM/ram[172]           d2
/spi_DUT/uut/RAM/ram                d0 e7 20 e9 a1 18...
Now                                 0.62 ns
Cursor 1                            0.619 ns
```

1
2
3
4
5
6
7
166
2cc
198
331
262
0c5
18b
317
00
d2
7f
ac
6c
d2
d0 e7 20 e9 a1 18 47 8c f5 f7 28 f8 f5 7c cc 24 6b ea ca f5 a7 09 d9 ef ad c1 be 64 a7 2b b4 08 46 0b 18 d2 b1 51 f3 08 70 61 c3 cb 2f 7d 72 a5 b5 c1 4...
4 ns
0.48 ns
0.52 ns
0.56 ns
0.6 ns
0.619 ns
Wave
tb_SPI.v

Read 0xd2 from ram[172,ac] and output it serially on MISO 1101 0010