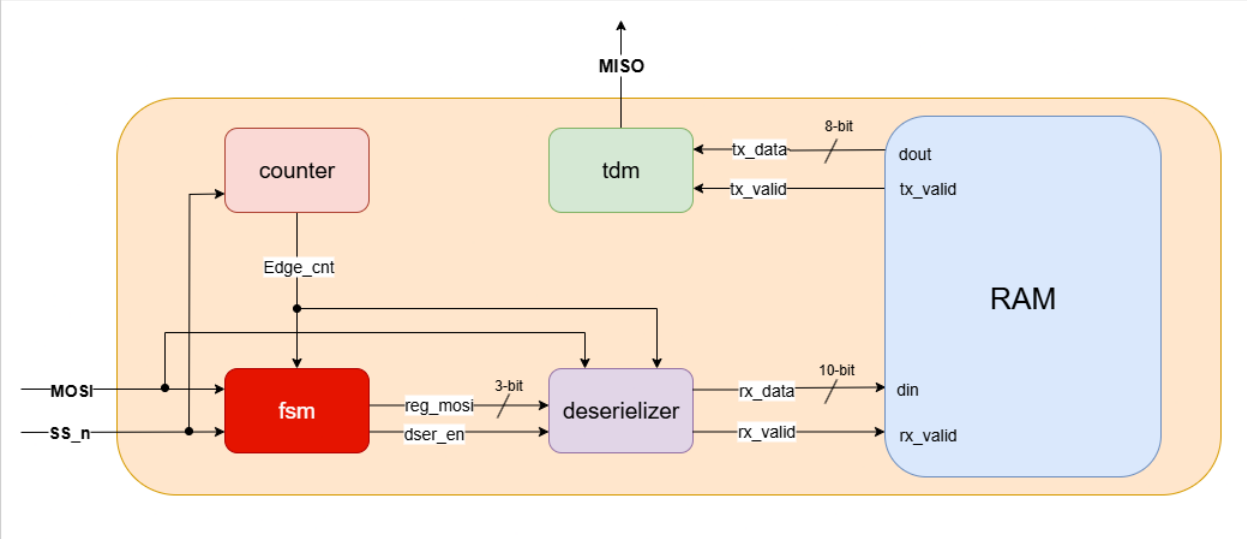
SPI Slave

named	Mohamed Safwat
group	1

SPI

Architecture



RTL Code

```
module spi (
    input   wire    MOSI ,
    input   wire    SS_n ,
    input   wire    clk ,
    input   wire    rst_n ,
    output  wire    MISO
) ;

/*----------------------------------------
-- internal signals
----------------------------------------*/
    wire     [3:0]   edge_cnt ;
    wire             dser_en ;
    wire     [3:0]   reg_mosi ;
    wire     [9:0]   rx_data ;
    wire             rx_valid ;
    wire     [7:0]   tx_data ;
    wire             tx_valid ;

/*----------------------------------------
-- FSM instance
----------------------------------------*/
    fsm f1 (
        .ss_n(SS_n),
        .mosi(MOSI),
        .edge_cnt(edge_cnt),
        .clk(clk),
        .rst_n(rst_n),
        .reg_mosi(reg_mosi),
        .dser_en(dser_en)
    ) ;

/*----------------------------------------
-- counter instance
----------------------------------------*/
    counter     c1 (
        .clk(clk),
```

```
        .rst_n(rst_n),
        .ss_n(SS_n),
        .edge_cnt(edge_cnt)
    );

/*-----------------------------------------
-- deserielizer instance
-----------------------------------------*/
    deserielizer    d1 (
        .mosi(MOSI) ,
        .reg_mosi(reg_mosi) ,
        .dser_en(dser_en) ,
        .edge_cnt(edge_cnt) ,
        .clk(clk) ,
        .rst_n(rst_n) ,
        .rx_data(rx_data) ,
        .rx_valid(rx_valid)
    );

/*-----------------------------------------
-- tdm instance
-----------------------------------------*/
    tdm    t1 (
        .tx_data(tx_data),
        .tx_valid(tx_valid),
        .clk(clk),
        .rst_n(rst_n),
        .miso(MISO)
    );

/*-----------------------------------------
-- ram instance
-----------------------------------------*/
    ram   #(.MEM_DEPTH(256) , .ADDR_SIZE(8))    r1 (
        .clk(clk) ,
```

```
            .rst_n(rst_n) ,
            .din(rx_data) ,
            .rx_valid(rx_valid) ,
            .tx_valid(tx_valid) ,
            .dout(tx_data)
        );

endmodule
```

```
module fsm (
    input       wire            ss_n ,
    input       wire            mosi ,
    input       wire    [3:0]   edge_cnt ,
    input       wire            clk ,
    input       wire            rst_n ,
    output      reg     [3:0]   reg_mosi ,
    output      reg             dser_en
) ;

/*-----------------------------------------------------------------
-- states
-----------------------------------------------------------------*/
    parameter   IDLE        = 3'b000 ,
                CHK_CMD     = 3'b001 ,
                WRITE       = 3'b010 ,
                READ_ADD    = 3'b011 ,
                READ_DATA   = 3'b100 ;

    //(*fsm_encoding = "one_hot"*)
    //(*fsm_encoding = "sequential"*)
    (*fsm_encoding = "gray"*)

    reg    [2:0]   next_state , current_state ;

/*-----------------------------------------------------------------------------
-- signal to allow spi slave to differitiate between
    read and address operation for write and read so
    i will store first 3 bits also i will send them to deserielizer
    because deserielizer is waiting 4 clock cycle to turn on (deser_en = 1)
    so all this time deserielizer won't be able to accept first 3  bits
-----------------------------------------------------------------------------*/
    always @ (posedge clk or negedge rst_n) begin
        if(!rst_n) begin
            reg_mosi <= 'b0 ;
        end
        else if(!ss_n) begin
```

```
            case(edge_cnt)
                0 : reg_mosi[3] <= mosi ;  //start bit
                1 : reg_mosi[2] <= mosi ;  //first bit
                2 : reg_mosi[1] <= mosi ;  //second bit
                3 : reg_mosi[0] <= mosi ;  //third bit
            endcase
        end
    end
/*-----------------------------------------------------------------
-- current state logic
-----------------------------------------------------------------*/
    always @ (posedge clk or negedge rst_n) begin
        if(!rst_n) begin
            current_state <= IDLE ;
        end

        else begin
            current_state <= next_state ;
        end
    end

/*-----------------------------------------------------------------
-- next state logic and output logic
-----------------------------------------------------------------*/
    always @ (*) begin
        dser_en = 0 ;

        case(current_state)
            IDLE : begin
                if(!ss_n) begin
                    next_state = CHK_CMD ;
                end
                else begin
```

```
                next_state = IDLE ;
            end
        end

        CHK_CMD : begin
            if(!ss_n && (edge_cnt == 3)) begin
                case(reg_mosi[3:1])
                    'b000 : next_state = WRITE ;
                    'b001 : next_state = WRITE ;
                    'b110 : next_state = READ_ADD ;
                    'b111 : next_state = READ_DATA ;
                    default : next_state = IDLE ;
                endcase

            end

            else begin
                next_state = CHK_CMD ;   //waiting 3 clock cycles
            end
        end

        WRITE : begin
            dser_en = 1 ;
            if(!ss_n) begin
                next_state = WRITE ;
            end

            else begin
                next_state = IDLE ;
            end
        end

        READ_ADD : begin
            dser_en = 1 ;
            if(!ss_n) begin
                next_state = READ_ADD ;
            end

            else begin
                next_state = IDLE ;
            end
        end

        READ_DATA : begin
            dser_en = 1 ;
            if(!ss_n) begin
                next_state = READ_DATA ;
            end

            else begin
                next_state = IDLE ;
            end
        end

        default : next_state = IDLE ;
    endcase
end

endmodule
```

```
module counter(
    input   wire            clk ,
    input   wire            rst_n ,
    input   wire            ss_n ,
    output  reg     [3:0]   edge_cnt    //size 4-bits so i can sample 10 bits from MOSI
);

/*-----------------------------------------------------------------
-- edge counter
-----------------------------------------------------------------*/
    always @ (posedge clk or negedge rst_n) begin
        if(!rst_n) begin
            edge_cnt <= 'b0 ;
        end
        else if(!ss_n) begin
            edge_cnt <= edge_cnt + 1 ;
        end
        else begin
            edge_cnt <= 'b0 ;
        end
    end
endmodule
```

```
module deserielizer (
    input       wire            mosi ,
    input       wire    [3:0]   reg_mosi ,
    input       wire            dser_en ,
    input       wire    [3:0]   edge_cnt ,
    input       wire            clk ,
    input       wire            rst_n ,
    output      reg     [9:0]   rx_data ,
    output      reg             rx_valid
) ;

/*-----------------------------------------------------------
-- always block
-----------------------------------------------------------*/
    always @ (posedge clk or negedge rst_n) begin
        if(!rst_n) begin
            rx_data  <= 'b0 ;
            rx_valid <= 'b0 ;
        end

        else begin
            rx_valid <= 1'b0 ;
            if(dser_en && edge_cnt != 11) begin
                rx_data[9:7] <= reg_mosi[2:0] ;    //registered first 3 bits excluding start bit
                rx_data[6:0] <= {rx_data[6:0] , mosi} ;
            end

            if(edge_cnt == 11) begin
                rx_valid <= 1'b1 ;
            end
        end
    end
endmodule
```

```
module tdm (
    input   wire    [7:0]    tx_data ,
    input   wire             tx_valid ,
    input   wire             clk ,
    input   wire             rst_n ,
    output  reg              miso
) ;
    reg     [2:0]       cnt ;
    reg                 flag ;
    reg     [7:0]       tx_data_reg ;

/*-----------------------------------------------------
-- register tx_data
-----------------------------------------------------*/
    always @ (posedge clk or negedge rst_n) begin
        if(!rst_n) begin
            tx_data_reg <= 0 ;
        end

        else if(tx_valid) begin
            tx_data_reg <= tx_data ;
        end
    end

 /*-----------------------------------------------------
-- counter
-----------------------------------------------------*/
    always @ (posedge clk or negedge rst_n) begin
        if(!rst_n) begin
            cnt <= 0 ;
            flag <= 0 ;
        end

        else if(tx_valid) begin
            cnt <= 0 ;
            flag <= 1 ;
        end
```

```
        else if(flag) begin
            cnt <= cnt + 1 ;
            if(cnt == 6) begin
                flag <= 0 ;
            end
        end
    end

/*-----------------------------------------------------
-- mux
-----------------------------------------------------*/
    always @ (posedge clk or negedge rst_n) begin
        if(!rst_n) begin
            miso <= 0 ;
        end
        else begin
            case(cnt)
                0 : miso <= tx_data_reg[7] ;
                1 : miso <= tx_data_reg[6] ;
                2 : miso <= tx_data_reg[5] ;
                3 : miso <= tx_data_reg[4] ;
                4 : miso <= tx_data_reg[3] ;
                5 : miso <= tx_data_reg[2] ;
                6 : miso <= tx_data_reg[1] ;
                7 : miso <= tx_data_reg[0] ;
                default : miso <= 1'b0 ;
            endcase
        end
    end
/*-----------------------------------------------------------------
-- kind note : mosi will get sampled after 2 clock
   cycles from which tx_valid get high because i
   registered inputs of mux and registered mosi
-----------------------------------------------------------------*/
endmodule
```

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

/*-----------------------------------------------------------
-- memory
-----------------------------------------------------------*/
    reg    [ADDR_SIZE-1 : 0]    mem     [MEM_DEPTH-1 : 0] ;

/*-----------------------------------------------------------
-- internal register
-----------------------------------------------------------*/
    reg    [ADDR_SIZE-1 : 0]    internal_reg ;

/*-----------------------------------------------------------
-- always block
-----------------------------------------------------------*/
    always @ (posedge clk) begin    // i made it sync reset to map on memory for fpga
        if(!rst_n) begin
            dout <= 'b0 ;
            tx_valid <= 1'b0 ;
            internal_reg <= 'b0 ;
        end

        else begin
            tx_valid <= 0 ;
            if(rx_valid) begin
                case(din[9:8])
                    'b00 : begin
                        internal_reg <= din[7:0] ;
                        tx_valid <= 1'b0;
                    end
                    'b01 : begin
                        mem[internal_reg] <= din[7:0] ;
                        tx_valid <= 1'b0;
                    end
                    'b10 : begin
                        internal_reg <= din[7:0] ;
                        tx_valid <= 1'b0;
                    end
                    'b11 : begin
                        tx_valid <= 1'b1 ;
                        dout <= mem[internal_reg] ;
                    end
                endcase
            end
        end
    end
endmodule
```

TestBench

```
module spi_tb ();
/*-----------------------------------------
-- signals
-----------------------------------------*/
    reg             MOSI_tb ;
    reg             SS_n_tb ;
    reg             clk_tb ;
    reg             rst_n_tb ;
    wire            MISO_tb ;

    bit     [7:0]   address_reg ;
    bit     [7:0]   data_reg ;
    integer         i ;
/*-----------------------------------------
-- instance
-----------------------------------------*/
    spi    DUT    (
        .MOSI(MOSI_tb) ,
        .SS_n(SS_n_tb) ,
        .clk(clk_tb) ,
        .rst_n(rst_n_tb) ,
        .MISO(MISO_tb)
    );

/*-----------------------------------------
-- clock generation
-----------------------------------------*/
    initial begin
        clk_tb = 0 ;
        forever begin
            #5  clk_tb = ~clk_tb ;
        end
    end

/*-----------------------------------------
-- initial block
-----------------------------------------*/
```

```
    initial begin
        $readmemb("mem.dat" , DUT.r1.mem) ;

        //reset functionality
        rst_n_tb = 0 ;
        MOSI_tb = 0 ;
        SS_n_tb = 1 ;
        @(negedge clk_tb) ;

        //write adderess operation
        rst_n_tb = 1 ;
        SS_n_tb = 0 ;   // start communication
        for(i = 0 ; i < 3 ; i = i+1) begin  //specify operation
            MOSI_tb = 1'b0 ;
            @(negedge clk_tb) ;
        end

        for(i = 0 ; i<8 ; i = i+1) begin
            MOSI_tb = $random ;
            address_reg[7-i] = MOSI_tb ;
            @(negedge clk_tb) ;
        end
        SS_n_tb = 1 ; //end communication
        repeat(2) @(negedge clk_tb) ;


        //write data operation
        rst_n_tb = 1 ;
        SS_n_tb = 0 ;   // start communication
        for(i = 0 ; i < 3 ; i = i+1) begin  //specify operation
            if(i == 0)       MOSI_tb = 0 ;
            else if(i == 1)  MOSI_tb = 0 ;
            else if(i == 2)  MOSI_tb = 1 ;
            @(negedge clk_tb) ;
        end

```

```
        for(i = 0 ; i<8 ; i = i+1) begin
            MOSI_tb = $random ;
            data_reg[7-i] = MOSI_tb ;
            @(negedge clk_tb) ;
        end
        SS_n_tb = 1 ; //end communication
        repeat(2) @(negedge clk_tb) ;


        //read address operation
        rst_n_tb = 1 ;
        SS_n_tb = 0 ;   // start communication
        for(i = 0 ; i < 3 ; i = i+1) begin  //specify operation
            if(i == 0)   MOSI_tb = 1 ;
            else if(i == 1)  MOSI_tb = 1 ;
            else if(i == 2)  MOSI_tb = 0 ;
            @(negedge clk_tb) ;
        end

        for(i = 0 ; i<8 ; i = i+1) begin
            MOSI_tb = address_reg[7-i] ;
            @(negedge clk_tb) ;
        end
        SS_n_tb = 1 ; //end communication
        repeat(2) @(negedge clk_tb) ;

        //read data operation
        rst_n_tb = 1 ;
        SS_n_tb = 0 ;   // start communication
        for(i = 0 ; i < 3 ; i = i+1) begin  //specify operation
            if(i == 0)   MOSI_tb = 1 ;
            else if(i == 1)  MOSI_tb = 1 ;
            else if(i == 2)  MOSI_tb = 1 ;
            @(negedge clk_tb) ;
        end
```

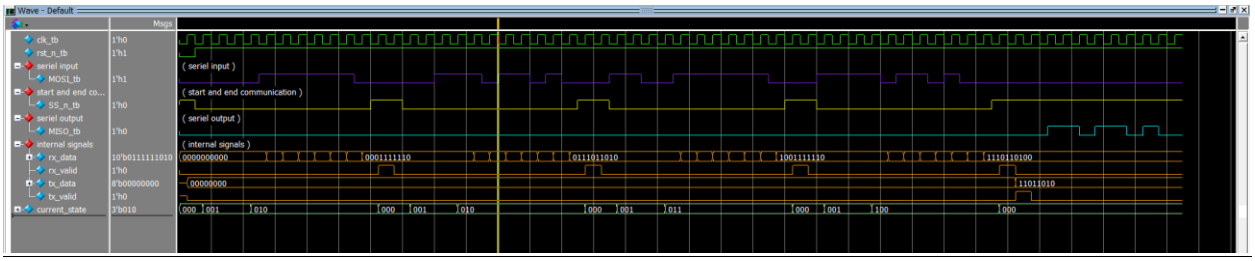
```
        for(i = 0 ; i<8 ; i = i+1) begin
            MOSI_tb = $random ;  //dummy bits in read data operation
            @(negedge clk_tb) ;
        end
        SS_n_tb = 1 ; //end communication
        repeat(10) @(negedge clk_tb) ;

        //comparing mechanism
        if(DUT.tx_data != data_reg) begin
            $display("failed operation") ;
        end
        else begin
            $display("successfull read-write operation") ;
        end



        #20 ;
        $stop ;
    end
endmodule
```

Waveform



Do file

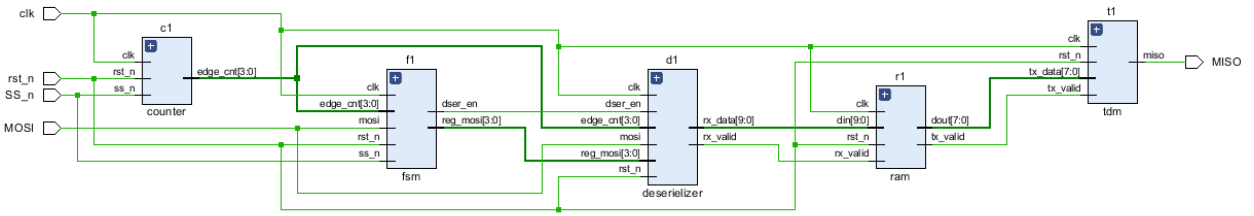
```
vlib work
vlog ../RTL_SPI/*.v   spi_tb.sv
vopt spi_tb -o safwat  +acc
vsim safwat
do wave.do
run -all
```

QuestaSim Transcript

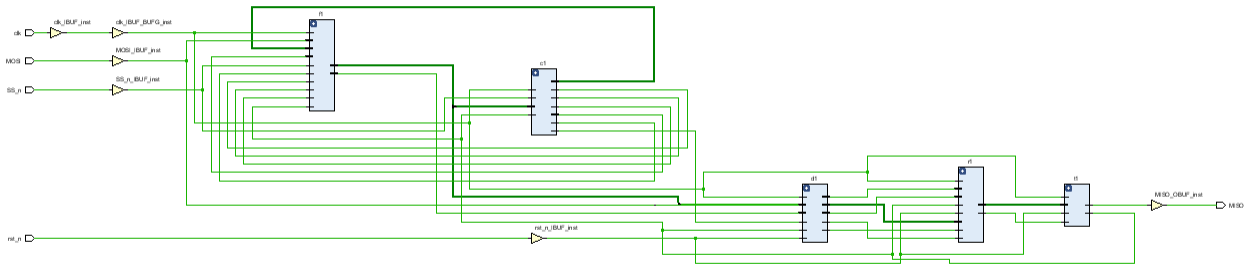
```
# successfull read-write operation
```

Binary encoding

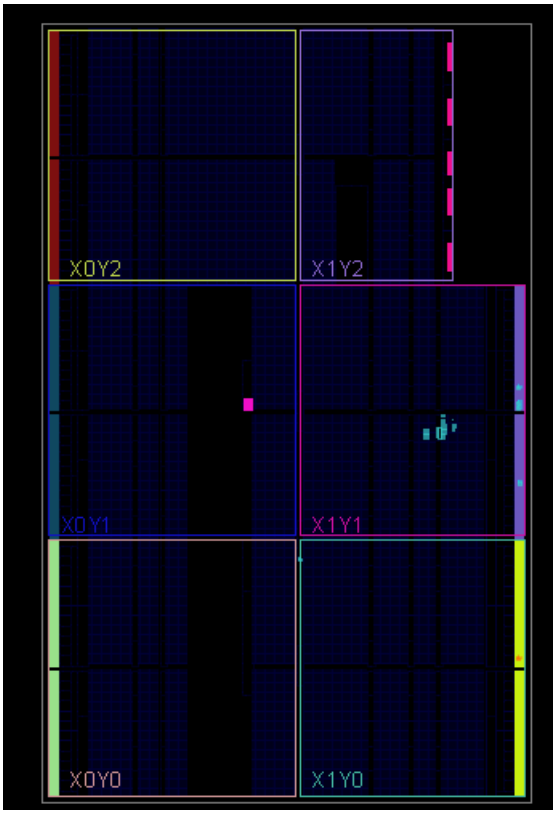
Elaboration Schematic



Synthesis Schematic



Implementation Device



Utilization synthesis

Resource	Utilization	Available	Utilization %
LUT	20	20800	0.10
FF	44	41600	0.11
BRAM	0.50	50	1.00
IO	5	106	4.72

Utilization Implementation

Resource	Utilization	Available	Utilization %
LUT	21	20800	0.10
FF	44	41600	0.11
BRAM	0.50	50	1.00
IO	5	106	4.72

Timing Synthesis

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.518 ns	Worst Hold Slack (WHS):	0.144 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	96	Total Number of Endpoints:	96	Total Number of Endpoints:	47

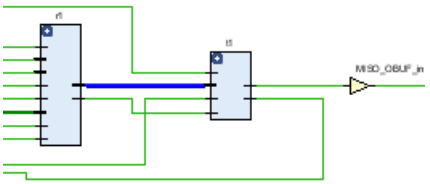
Timing Implementation

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.546 ns	Worst Hold Slack (WHS):	0.068 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	97	Total Number of Endpoints:	97	Total Number of Endpoints:	47

Encoding Report

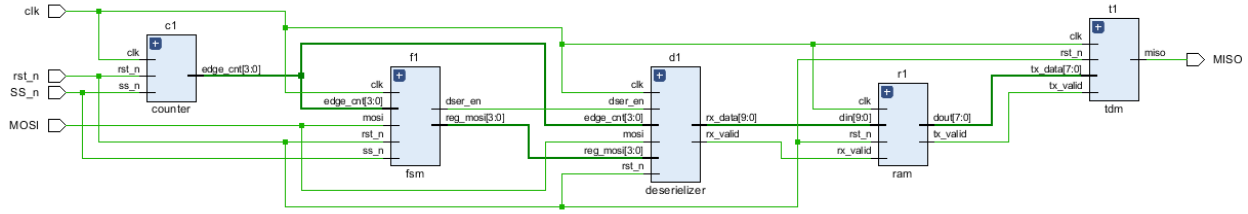
```
-------------------------------------------------------------------------------------------------
               State |                     New Encoding |                Previous Encoding
-------------------------------------------------------------------------------------------------
                IDLE |                              000 |                              000
             CHK_CMD |                              001 |                              001
               WRITE |                              010 |                              010
            READ_ADD |                              011 |                              011
           READ_DATA |                              100 |                              100
-------------------------------------------------------------------------------------------------
INFO: [Synth 8-3354] encoded FSM with state register 'current_state_reg' using encoding 'sequential' in module 'fsm'
```

Critical Path

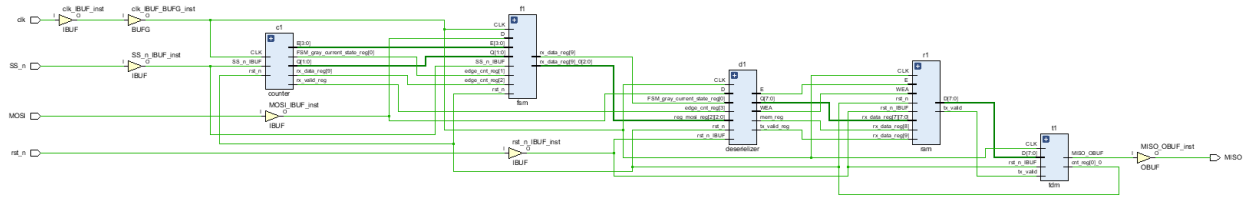


Gray encoding

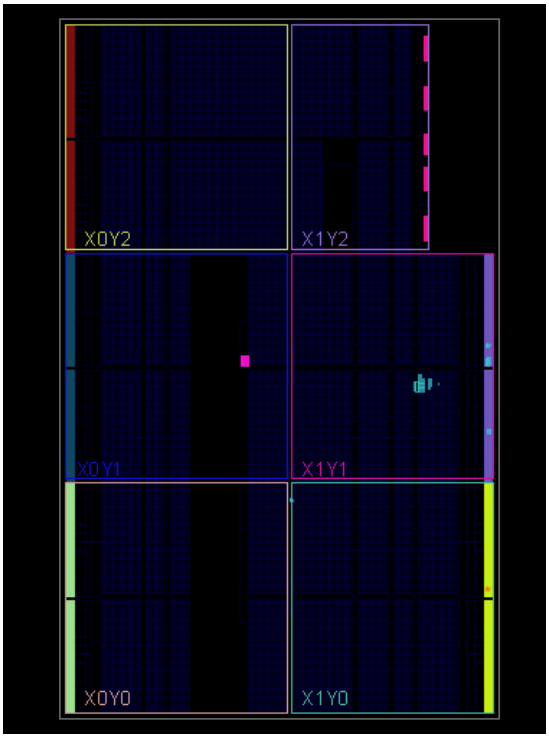
Elaboration Schematic



Synthesis Schematic



Implementation Device



Utilization synthesis

Resource	Utilization	Available	Utilization %
LUT	20	20800	0.10
FF	44	41600	0.11
BRAM	0.50	50	1.00
IO	5	106	4.72

Utilization Implementation

Resource	Utilization	Available	Utilization %
LUT	21	20800	0.10
FF	44	41600	0.11
BRAM	0.50	50	1.00
IO	5	106	4.72

Timing Synthesis

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.518 ns	Worst Hold Slack (WHS):	0.144 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	96	Total Number of Endpoints:	96	Total Number of Endpoints:	47

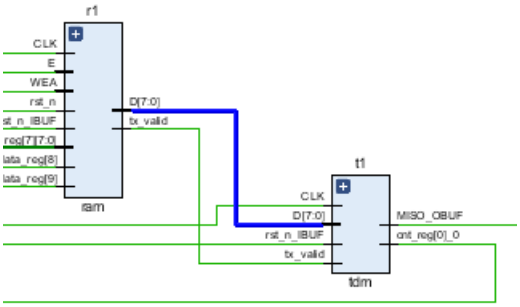
Timing Implementation

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.375 ns	Worst Hold Slack (WHS):	0.068 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	97	Total Number of Endpoints:	97	Total Number of Endpoints:	47

Encoding Report

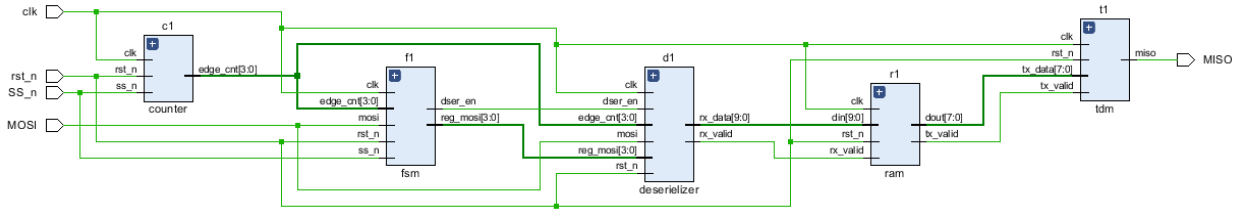
```
-------------------------------------------------------------------------------------------------
                  State |                     New Encoding |                Previous Encoding
-------------------------------------------------------------------------------------------------
                   IDLE |                              000 |                              000
                CHK_CMD |                              001 |                              001
                  WRITE |                              011 |                              010
               READ_ADD |                              010 |                              011
              READ_DATA |                              111 |                              100
-------------------------------------------------------------------------------------------------
INFO: [Synth 8-3354] encoded FSM with state register 'current_state_reg' using encoding 'gray' in module 'fsm'
```

Critical Path

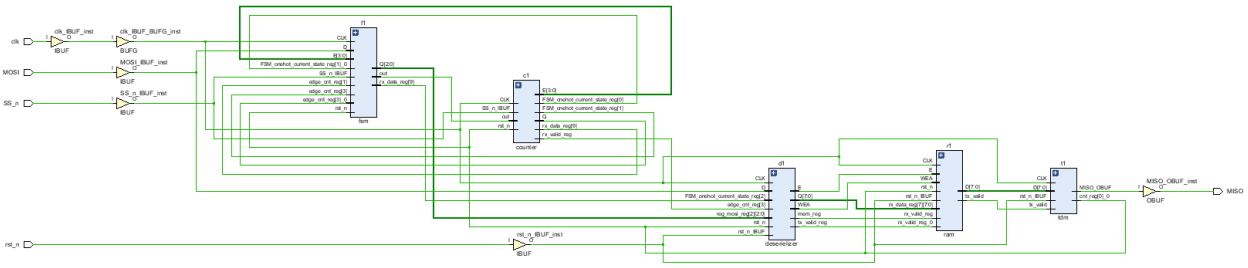


One hot encoding

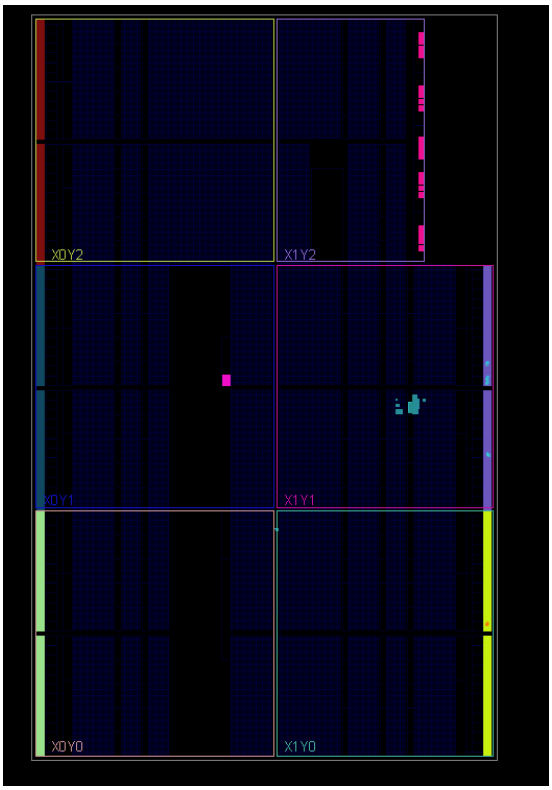
Elaboration Schematic



Synthesis Schematic



Implementation Device



Utilization synthesis

Resource	Utilization	Available	Utilization %
LUT	22	20800	0.11
FF	46	41600	0.11
BRAM	0.50	50	1.00
IO	5	106	4.72

Utilization Implementation

Resource	Utilization	Available	Utilization %
LUT	23	20800	0.11
FF	46	41600	0.11
BRAM	0.50	50	1.00
IO	5	106	4.72

Timing Synthesis

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.518 ns	Worst Hold Slack (WHS):	0.144 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	98	Total Number of Endpoints:	98	Total Number of Endpoints:	49

Timing Implementation

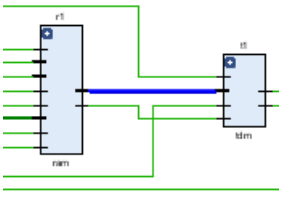
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.585 ns	Worst Hold Slack (WHS):	0.068 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	99	Total Number of Endpoints:	99	Total Number of Endpoints:	49

Encoding Report

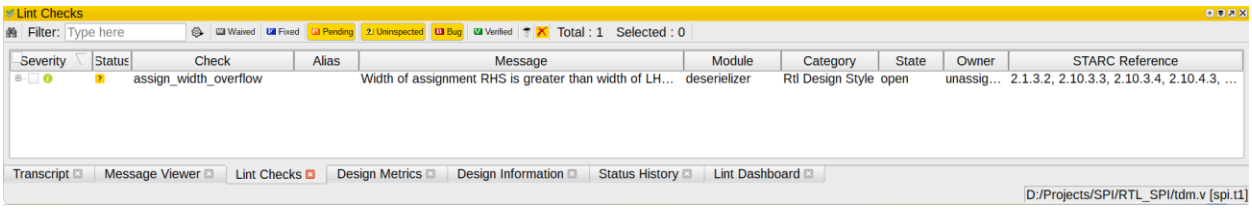
State	New Encoding	Previous Encoding
IDLE	00001	000
CHK_CMD	00010	001
WRITE	00100	010
READ_ADD	01000	011
READ_DATA	10000	100

```
INFO: [Synth 8-3354] encoded FSM with state register 'current_state_reg' using encoding 'one-hot' in module 'fsm'
```

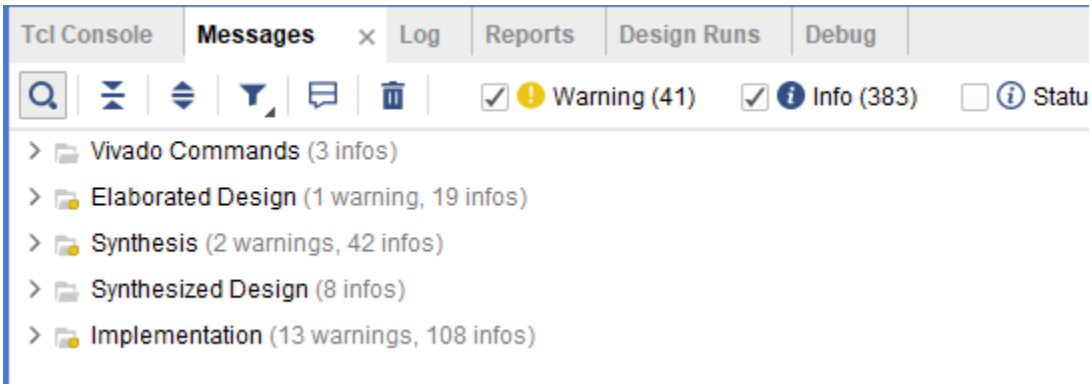
Critical Path



Check Lint



Messages


Tcl Console
Messages
Log
Reports
Design Runs
Debug
Warning (41)
Info (383)
Statu
Vivado Commands (3 infos)
Elaborated Design (1 warning, 19 infos)
Synthesis (2 warnings, 42 infos)
Synthesized Design (8 infos)
Implementation (13 warnings, 108 infos)