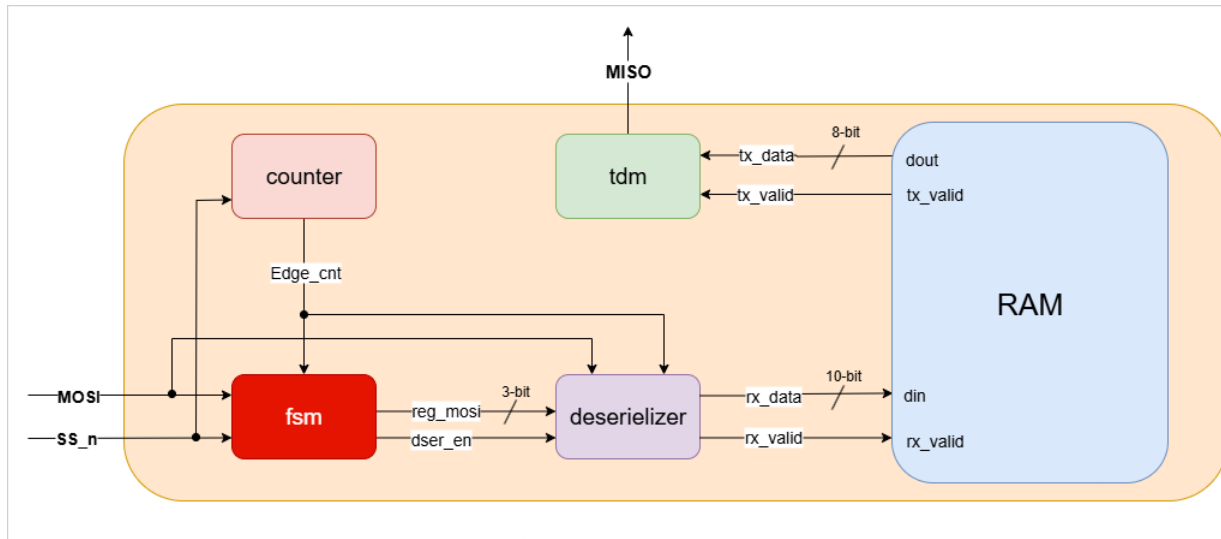


SPI Slave

named	Mohamed Safwat
group	1

SPI

Architecture



RTL Code

```
1  module spi (  
2      input  wire    MOSI ,  
3      input  wire    SS_n ,  
4      input  wire    clk ,  
5      input  wire    rst_n ,  
6      output wire    MISO  
7  ) ;  
8  
9  /*-----  
10 -- internal signals  
11 -----*/  
12     wire    [3:0]    edge_cnt ;  
13     wire          dser_en ;  
14     wire    [3:0]    reg_mosi ;  
15     wire    [9:0]    rx_data ;  
16     wire          rx_valid ;  
17     wire    [7:0]    tx_data ;  
18     wire          tx_valid ;  
19  
20 /*-----  
21 -- FSM instance  
22 -----*/  
23     fsm f1 (  
24         .ss_n(SS_n),  
25         .mosi(MOSI),  
26         .edge_cnt(edge_cnt),  
27         .clk(clk),  
28         .rst_n(rst_n),  
29         .reg_mosi(reg_mosi),  
30         .dser_en(dser_en)  
31     ) ;  
32  
33 /*-----  
34 -- counter instance  
35 -----*/  
36     counter    c1 (  
37         .clk(clk),
```

```

38     .rst_n(rst_n),
39     .ss_n(SS_n),
40     .edge_cnt(edge_cnt)
41 );
42
43 /*-----
44 -- deserializer instance
45 -----*/
46 deserializer d1 (
47     .mosi(MOSI) ,
48     .reg_mosi(reg_mosi) ,
49     .dser_en(dser_en) ,
50     .edge_cnt(edge_cnt) ,
51     .clk(clk) ,
52     .rst_n(rst_n) ,
53     .rx_data(rx_data) ,
54     .rx_valid(rx_valid)
55 );
56
57 /*-----
58 -- tdm instance
59 -----*/
60 tdm t1 (
61     .tx_data(tx_data),
62     .tx_valid(tx_valid),
63     .clk(clk),
64     .rst_n(rst_n),
65     .miso(MISO)
66 );
67
68 /*-----
69 -- ram instance
70 -----*/
71 ram #(.MEM_DEPTH(256) , .ADDR_SIZE(8)) r1 (
72     .clk(clk) ,

```

```

73     .rst_n(rst_n) ,
74     .din(rx_data) ,
75     .rx_valid(rx_valid) ,
76     .tx_valid(tx_valid) ,
77     .dout(tx_data)
78 );
79
80 endmodule

```

```

1  module fsm (
2      input    wire    ss_n ,
3      input    wire    mosi ,
4      input    wire    [3:0] edge_cnt ,
5      input    wire    clk ,
6      input    wire    rst_n ,
7      output   reg     [3:0] reg_mosi ,
8      output   reg     dser_en
9  ) ;
10
11  /*-----
12  -- states
13  -----*/
14      parameter IDLE      = 3'b000 ,
15                 CHK_CMD  = 3'b001 ,
16                 WRITE    = 3'b010 ,
17                 READ_ADD = 3'b011 ,
18                 READ_DATA = 3'b100 ;
19
20      /*(*fsm_encoding = "one_hot"*)
21      /*(*fsm_encoding = "sequential"*)
22      (*fsm_encoding = "gray"*)
23
24      reg [2:0] next_state , current_state ;
25
26  /*-----
27  -- signal to allow spi slave to differitiate between
28  read and address operation for write and read so
29  i will store first 3 bits also i will send them to deserielizer
30  because deserielizer is waiting 4 clock cycle to turn on (dser_en = 1)
31  so all this time deserielizer won't be able to accept first 3 bits
32  -----*/
33      always @ (posedge clk or negedge rst_n) begin
34          if(!rst_n) begin
35              reg_mosi <= 'b0 ;
36          end
37          else if(!ss_n) begin

```

```

38              case(edge_cnt)
39                  0 : reg_mosi[3] <= mosi ; //start bit
40                  1 : reg_mosi[2] <= mosi ; //first bit
41                  2 : reg_mosi[1] <= mosi ; //second bit
42                  3 : reg_mosi[0] <= mosi ; //third bit
43              endcase
44          end
45      end
46  /*-----
47  -- current state logic
48  -----*/
49      always @ (posedge clk or negedge rst_n) begin
50          if(!rst_n) begin
51              current_state <= IDLE ;
52          end
53
54          else begin
55              current_state <= next_state ;
56          end
57      end
58
59  /*-----
60  -- next state logic and output logic
61  -----*/
62      always @ (*) begin
63          dser_en = 0 ;
64
65          case(current_state)
66              IDLE : begin
67                  if(!ss_n) begin
68                      next_state = CHK_CMD ;
69                  end
70                  else begin

```

```

71         next_state = IDLE ;
72     end
73 end
74
75     CHK_CMD : begin
76         if(!ss_n && (edge_cnt == 3)) begin
77             case(reg_mosi[3:1])
78                 'b000 : next_state = WRITE ;
79                 'b001 : next_state = WRITE ;
80                 'b110 : next_state = READ_ADD ;
81                 'b111 : next_state = READ_DATA ;
82                 default : next_state = IDLE ;
83             endcase
84
85         end
86
87         else begin
88             next_state = CHK_CMD ;    //waiting 3 clock cycles
89         end
90     end
91
92     WRITE : begin
93         dser_en = 1 ;
94         if(!ss_n) begin
95             next_state = WRITE ;
96         end
97
98         else begin
99             next_state = IDLE ;
100         end
101     end
102 end

```

```

103     READ_ADD : begin
104         dser_en = 1 ;
105         if(!ss_n) begin
106             next_state = READ_ADD ;
107         end
108
109         else begin
110             next_state = IDLE ;
111         end
112     end
113
114     READ_DATA : begin
115         dser_en = 1 ;
116         if(!ss_n) begin
117             next_state = READ_DATA ;
118         end
119
120         else begin
121             next_state = IDLE ;
122         end
123     end
124
125     default : next_state = IDLE ;
126 endcase
127 end
128
129 endmodule

```

```

1  module counter(
2      input  wire      clk ,
3      input  wire      rst_n ,
4      input  wire      ss_n ,
5      output reg  [3:0] edge_cnt //size 4-bits so i can sample 10 bits from MOSI
6  );
7
8  /*-----
9  -- edge counter
10 -----*/
11  always @ (posedge clk or negedge rst_n) begin
12      if(!rst_n) begin
13          edge_cnt <= 'b0 ;
14      end
15      else if(!ss_n) begin
16          edge_cnt <= edge_cnt + 1 ;
17      end
18      else begin
19          edge_cnt <= 'b0 ;
20      end
21  end
22  endmodule

```

```

1  module deserielizer (
2      input  wire      mosi ,
3      input  wire  [3:0] reg_mosi ,
4      input  wire      dser_en ,
5      input  wire  [3:0] edge_cnt ,
6      input  wire      clk ,
7      input  wire      rst_n ,
8      output reg  [9:0] rx_data ,
9      output reg      rx_valid
10 );
11
12 /*-----
13 -- always block
14 -----*/
15  always @ (posedge clk or negedge rst_n) begin
16      if(!rst_n) begin
17          rx_data <= 'b0 ;
18          rx_valid <= 'b0 ;
19      end
20
21      else begin
22          rx_valid <= 1'b0 ;
23          if(dser_en && edge_cnt != 11) begin
24              rx_data[9:7] <= reg_mosi[2:0] ; //registered first 3 bits excluding start bit
25              rx_data[6:0] <= {rx_data[6:0] , mosi} ;
26          end
27
28          if(edge_cnt == 11) begin
29              rx_valid <= 1'b1 ;
30          end
31      end
32  end
33  endmodule

```

```

1  module tdm (
2      input wire [7:0] tx_data ,
3      input wire tx_valid ,
4      input wire clk ,
5      input wire rst_n ,
6      output reg miso
7  ) ;
8      reg [2:0] cnt ;
9      reg flag ;
10     reg [7:0] tx_data_reg ;
11
12     /*-----
13     -- register tx_data
14     -----*/
15     always @ (posedge clk or negedge rst_n) begin
16         if(!rst_n) begin
17             tx_data_reg <= 0 ;
18         end
19
20         else if(tx_valid) begin
21             tx_data_reg <= tx_data ;
22         end
23     end
24
25     /*-----
26     -- counter
27     -----*/
28     always @ (posedge clk or negedge rst_n) begin
29         if(!rst_n) begin
30             cnt <= 0 ;
31             flag <= 0 ;
32         end
33
34         else if(tx_valid) begin
35             cnt <= 0 ;
36             flag <= 1 ;
37         end
38
39         else if(flag) begin
40             cnt <= cnt + 1 ;
41             if(cnt == 6) begin
42                 flag <= 0 ;
43             end
44         end
45     end
46
47     /*-----
48     -- mux
49     -----*/
50     always @ (posedge clk or negedge rst_n) begin
51         if(!rst_n) begin
52             miso <= 0 ;
53         end
54         else begin
55             case(cnt)
56                 0 : miso <= tx_data_reg[7] ;
57                 1 : miso <= tx_data_reg[6] ;
58                 2 : miso <= tx_data_reg[5] ;
59                 3 : miso <= tx_data_reg[4] ;
60                 4 : miso <= tx_data_reg[3] ;
61                 5 : miso <= tx_data_reg[2] ;
62                 6 : miso <= tx_data_reg[1] ;
63                 7 : miso <= tx_data_reg[0] ;
64                 default : miso <= 1'b0 ;
65             endcase
66         end
67     end
68
69     /*-----
70     -- kind note : mosi will get sampled after 2 clock
71     cycles from which tx_valid get high because i
72     registered inputs of mux and registered mosi
73     -----*/
74 endmodule

```

```

1  module ram #(
2      parameter MEM_DEPTH = 256 ,
3      parameter ADDR_SIZE = 8
4  ) (
5      input  wire      clk ,
6      input  wire      rst_n ,
7      input  wire [9:0] din ,
8      input  wire      rx_valid ,
9      output reg       tx_valid ,
10     output reg [7:0]  dout
11 );
12
13 /*-----
14 -- memory
15 -----*/
16     reg [ADDR_SIZE-1 : 0] mem [MEM_DEPTH-1 : 0] ;
17
18 /*-----
19 -- internal register
20 -----*/
21     reg [ADDR_SIZE-1 : 0] internal_reg ;
22
23 /*-----
24 -- always block
25 -----*/
26     always @ (posedge clk) begin // i made it sync reset to map on memory for fpga
27         if(!rst_n) begin
28             dout <= 'b0 ;
29             tx_valid <= 1'b0 ;
30             internal_reg <= 'b0 ;
31         end
32
33         else begin
34             tx_valid <= 0 ;
35             if(rx_valid) begin
36                 case(din[9:8])
37                     'b00 : begin
38                         internal_reg <= din[7:0] ;
39                         tx_valid <= 1'b0;
40                     end
41                     'b01 : begin
42                         mem[internal_reg] <= din[7:0] ;
43                         tx_valid <= 1'b0;
44                     end
45                     'b10 : begin
46                         internal_reg <= din[7:0] ;
47                         tx_valid <= 1'b0;
48                     end
49                     'b11 : begin
50                         tx_valid <= 1'b1 ;
51                         dout <= mem[internal_reg] ;
52                     end
53                 endcase
54             end
55         end
56     end
57 endmodule

```


TestBench

```
1  module spi_tb ();
2  /*-----
3  -- signals
4  -----*/
5      reg        MOSI_tb ;
6      reg        SS_n_tb ;
7      reg        clk_tb ;
8      reg        rst_n_tb ;
9      wire       MISO_tb ;
10
11      bit        [7:0]  address_reg ;
12      bit        [7:0]  data_reg ;
13      integer     i ;
14  /*-----
15  -- instance
16  -----*/
17      spi    DUT (
18          .MOSI(MOSI_tb) ,
19          .SS_n(SS_n_tb) ,
20          .clk(clk_tb) ,
21          .rst_n(rst_n_tb) ,
22          .MISO(MISO_tb)
23      );
24
25  /*-----
26  -- clock generation
27  -----*/
28      initial begin
29          clk_tb = 0 ;
30          forever begin
31              #5  clk_tb = ~clk_tb ;
32          end
33      end
34
35  /*-----
36  -- initial block
37  -----*/
38      initial begin
39          $readmemb("mem.dat" , DUT.r1.mem) ;
40
41          //reset functionality
42          rst_n_tb = 0 ;
43          MOSI_tb = 0 ;
44          SS_n_tb = 1 ;
45          @(negedge clk_tb) ;
46
47          //write address operation
48          rst_n_tb = 1 ;
49          SS_n_tb = 0 ; // start communication
50          for(i = 0 ; i < 3 ; i = i+1) begin //specify operation
51              MOSI_tb = 1'b0 ;
52              @(negedge clk_tb) ;
53          end
54
55          for(i = 0 ; i < 8 ; i = i+1) begin
56              MOSI_tb = $random ;
57              address_reg[7-i] = MOSI_tb ;
58              @(negedge clk_tb) ;
59          end
60          SS_n_tb = 1 ; //end communication
61          repeat(2) @(negedge clk_tb) ;
62
63          //write data operation
64          rst_n_tb = 1 ;
65          SS_n_tb = 0 ; // start communication
66          for(i = 0 ; i < 3 ; i = i+1) begin //specify operation
67              if(i == 0)    MOSI_tb = 0 ;
68              else if(i == 1) MOSI_tb = 0 ;
69              else if(i == 2) MOSI_tb = 1 ;
70              @(negedge clk_tb) ;
71          end
72      end
73  end
```

```

74  ✓   for(i = 0 ; i<8 ; i = i+1) begin
75      MOSI_tb = $random ;
76      data_reg[7-i] = MOSI_tb ;
77      @(negedge clk_tb) ;
78  end
79  SS_n_tb = 1 ; //end communication
80  repeat(2) @(negedge clk_tb) ;
81
82
83  //read address operation
84  rst_n_tb = 1 ;
85  SS_n_tb = 0 ; // start communication
86  ✓   for(i = 0 ; i < 3 ; i = i+1) begin //specify operation
87      if(i == 0) MOSI_tb = 1 ;
88      else if(i == 1) MOSI_tb = 1 ;
89      else if(i == 2) MOSI_tb = 0 ;
90      @(negedge clk_tb) ;
91  end
92
93  ✓   for(i = 0 ; i<8 ; i = i+1) begin
94      MOSI_tb = address_reg[7-i] ;
95      @(negedge clk_tb) ;
96  end
97  SS_n_tb = 1 ; //end communication
98  repeat(2) @(negedge clk_tb) ;
99
100  //read data operation
101  rst_n_tb = 1 ;
102  SS_n_tb = 0 ; // start communication
103  ✓   for(i = 0 ; i < 3 ; i = i+1) begin //specify operation
104      if(i == 0) MOSI_tb = 1 ;
105      else if(i == 1) MOSI_tb = 1 ;
106      else if(i == 2) MOSI_tb = 1 ;
107      @(negedge clk_tb) ;
108  end

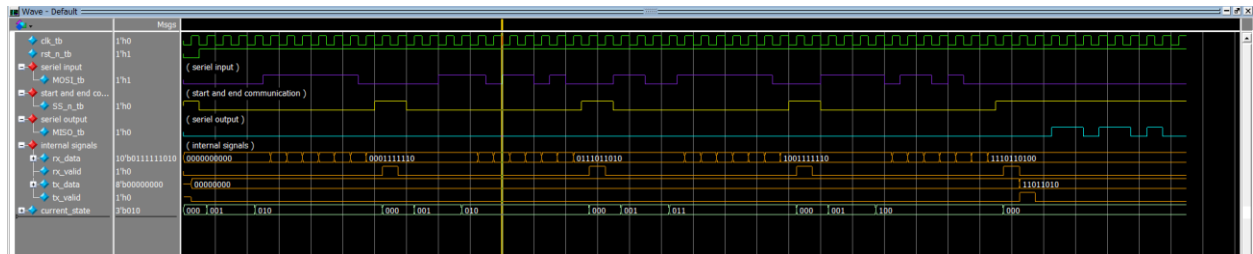
```

```

109
110      for(i = 0 ; i<8 ; i = i+1) begin
111          MOSI_tb = $random ; //dummy bits in read data operation
112          @(negedge clk_tb) ;
113      end
114      SS_n_tb = 1 ; //end communication
115      repeat(10) @(negedge clk_tb) ;
116
117  //comparing mechanism
118  if(DUT.tx_data != data_reg) begin
119      $display("failed operation") ;
120  end
121  else begin
122      $display("successfull read-write operation") ;
123  end
124
125
126
127  #20 ;
128  $stop ;
129  end
130  endmodule

```

Waveform



Do file

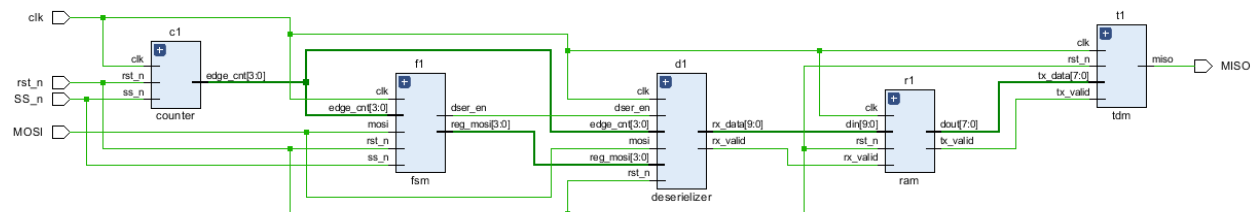
```
1 vlib work
2 vlog ../RTL_SPI/*.v spi_tb.sv
3 vopt spi_tb -o safwat +acc
4 vsim safwat
5 do wave.do
6 run -all
```

QuestaSim Transcript

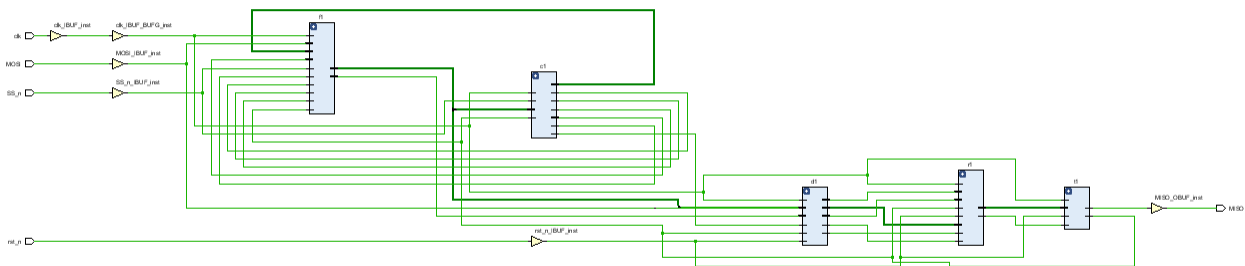
```
# successfull read-write operation
```

Binary encoding

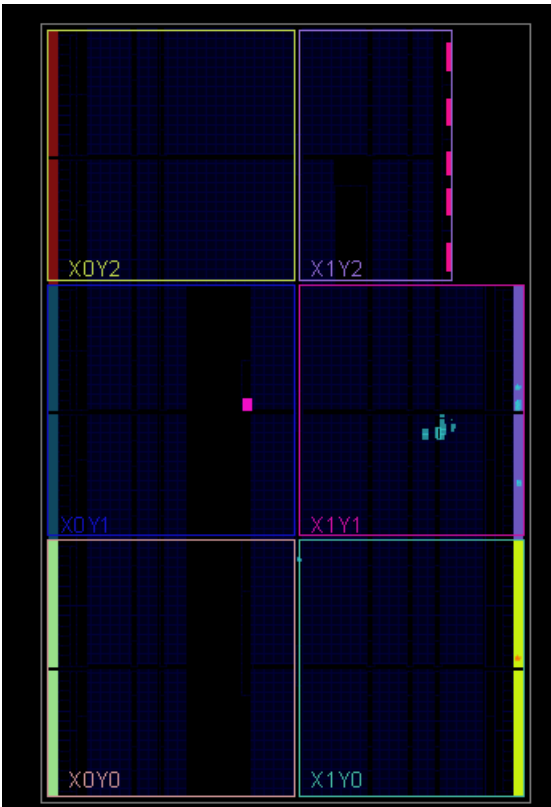
Elaboration Schematic



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

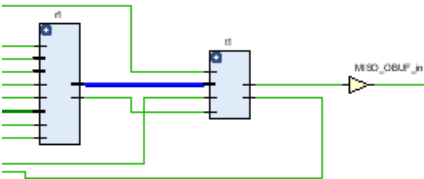
Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 6.546 ns	Worst Hold Slack (WHS): 0.068 ns	Worst Pulse Width Slack (WPWS): 4.500 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing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

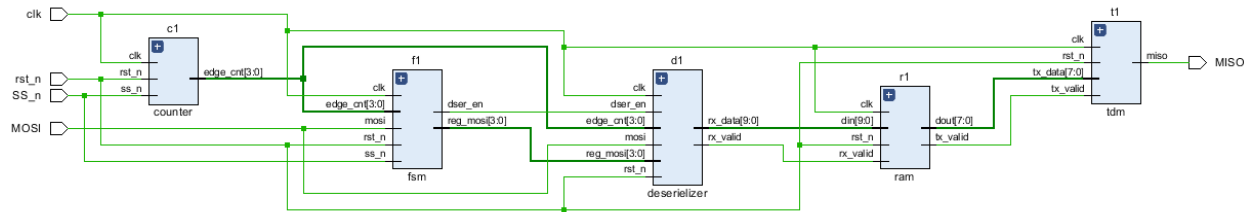
INFO: [Synth 8-3354] encoded FSM with state register 'current_state_reg' using encoding 'sequential' in module 'fsm'

Critical Path

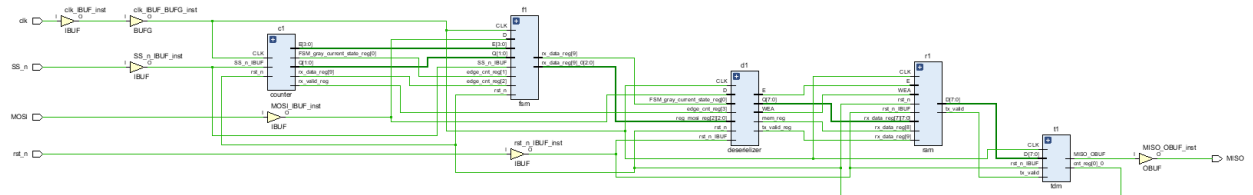


Gray encoding

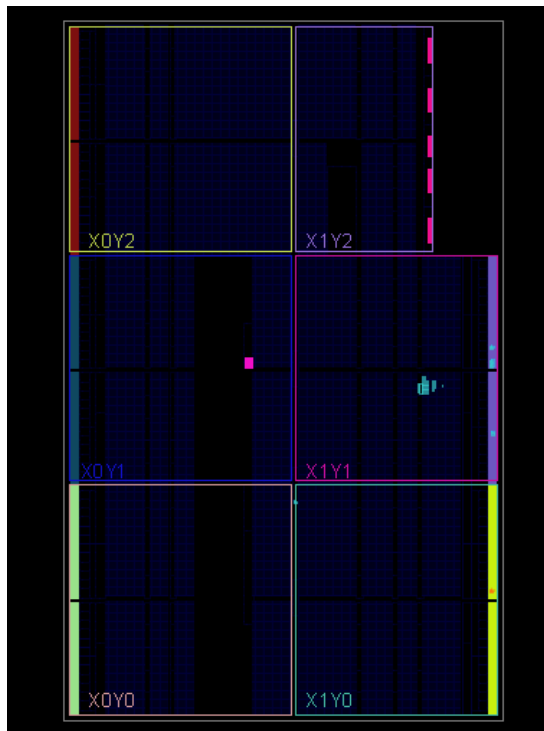
Elaboration Schematic



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

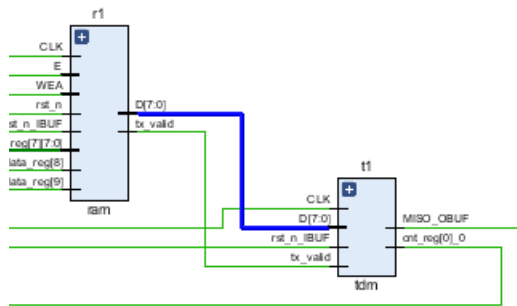
Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 6.375 ns	Worst Hold Slack (WHS): 0.068 ns	Worst Pulse Width Slack (WPWS): 4.500 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing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

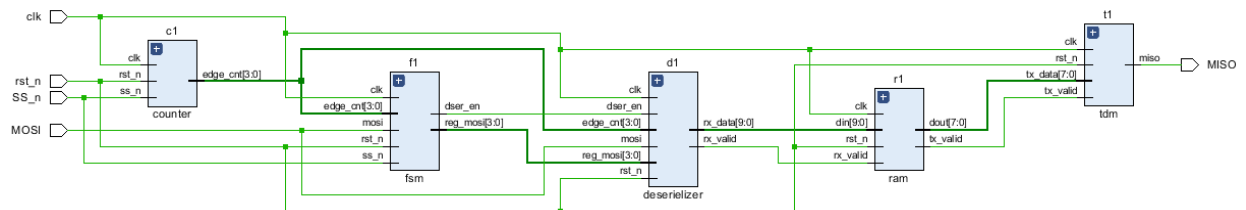
INFO: [Synth 8-3354] encoded FSM with state register 'current_state_reg' using encoding 'gray' in module 'fsm'

Critical Path

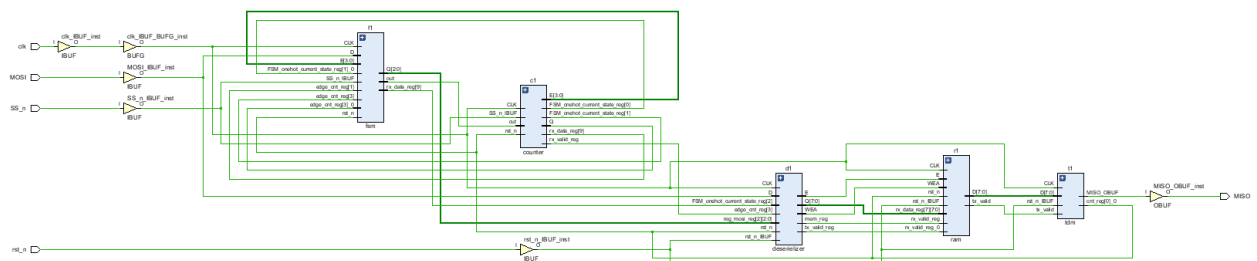


One hot encoding

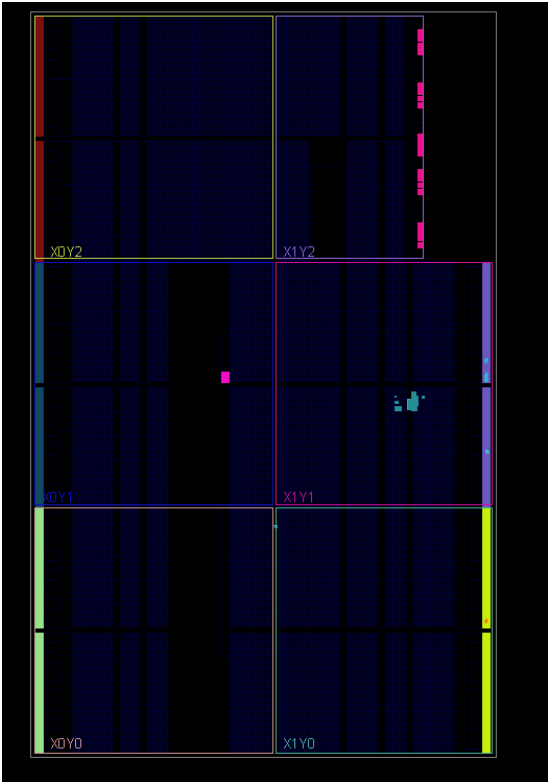
Elaboration Schematic



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

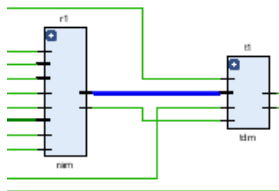
Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 6.585 ns	Worst Hold Slack (WHS): 0.068 ns	Worst Pulse Width Slack (WPWS): 4.500 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing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

Lint Checks

Filter: Type here

Waived

Fixed

Pending

2 Unresolved

Bug

Verified

Total : 1 Selected : 0

Severity	Status	Check	Alias	Message	Module	Category	State	Owner	STARC Reference
		assign_width_overflow		Width of assignment RHS is greater than width of LH...	deserializer	Rtl Design Style	open	unassign...	2.1.3.2, 2.10.3.3, 2.10.3.4, 2.10.4.3, ...

Transcript

Message Viewer

Lint Checks

Design Metrics

Design Information

Status History

Lint Dashboard

D:/Projects/SPI/RTL SPI/dm v [spi.tl

Messages

Tcl Console

Messages x

Log

Reports

Design Runs

Debug

☒ Warning (41)

☒ Info (383)

☐ Status

> Vivado Commands (3 infos)

> Elaborated Design (1 warning, 19 infos)

> Synthesis (2 warnings, 42 infos)

> Synthesized Design (8 infos)

> Implementation (13 warnings, 108 infos)