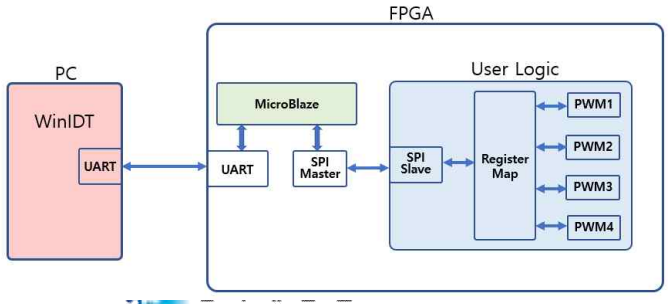
SoC 를 위한 *Peripheral* 설계

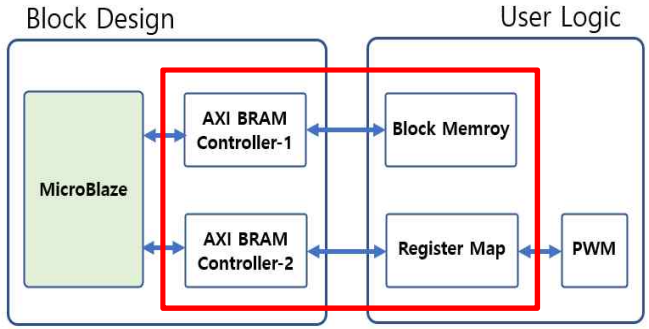
[*Serial Bus* ➔ *SPI* 구현]

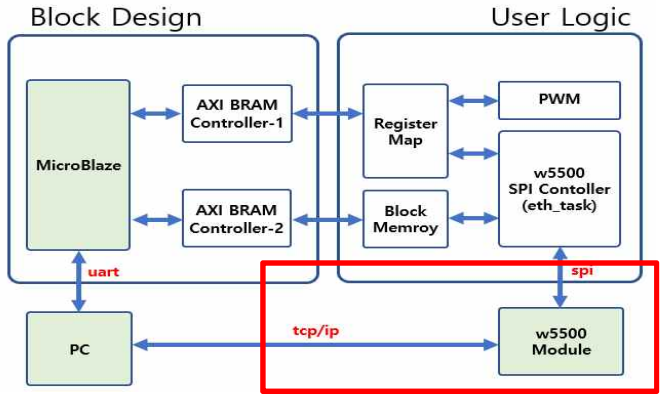
[*Reference*]

- *https://ko.wikipedia.org/wiki/%EC%A7%81%EB%A0%AC_%ED%86%B5%EC%8B%A0*
- *https://ko.wikipedia.org/wiki/UART*
- *MicroBlaze.v15 [IHIL]*
- *https://hanbulkr.tistory.com/5*
- *https://electriceng.tistory.com/422*

Table of Contents







SPI Slave Implementation

- ➔ Spec and Timing 정의
- ➔ Port 정의
- ➔ State 정의
- ➔ Code Implementation
- ➔ State Transition
- ➔ Test Bench
- ➔ Simulation Result

SPI Slave Implementation

➔ *Spec and Timing 정의*

SPI Slave Implementation

➢ *spi_slave_exam_0.xpr*

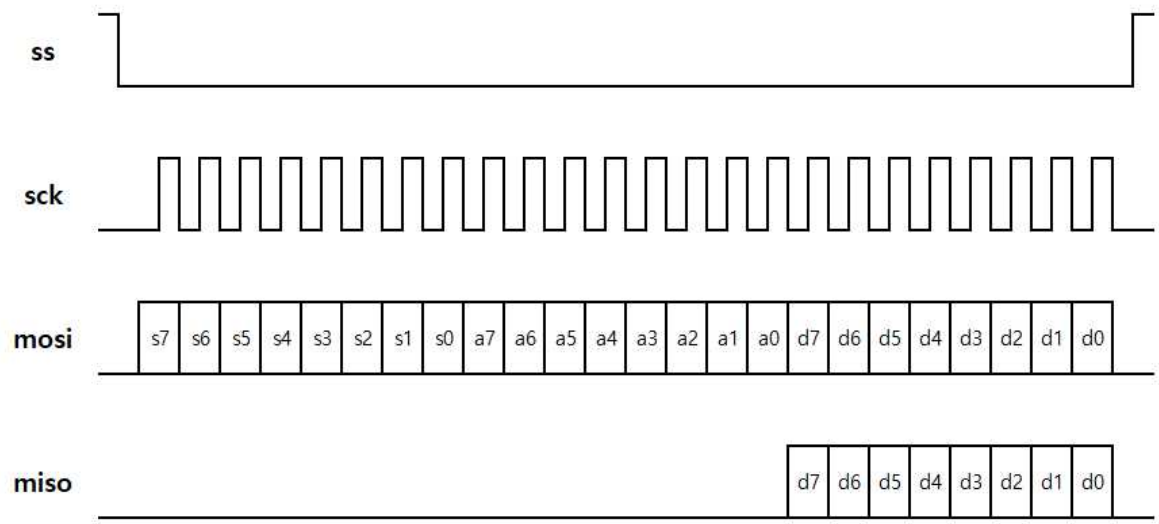
➢ *SPI (Serial Peripheral Interface Bus) Master Implementation*

❖ ***Spec*** ➔ SPI Slave 를 구현하기 위하여 스펙을 정의(SPI Master 와 동일)

- *slave_id : 0x64 (for write), 0x65 (for read)*, (*write, read*는 *Master* 기준)
- *Address : 8bits , Data : 8bits* (연속해서 Access 하는 것은 구현하지 않음)

❖ ***SPI Timing*** ➔ SPI Slave : 총 3바이트를 전송 (SPI Master 와 동일)

- Slave에 데이터를 저장하기 위해서는 *slave_id(0x64), address, data*를 전송
- Slave에서 데이터를 읽기 위해서는 *slave_id(0x65), address*를 전송→ miso를 통하여 데이터를 read
- mosi, miso는 sck에 동기 되어 동작→ sck의 negative edge(or Low 구간)에서 데이터를 변경하고, sck 의 positive edge (or High 구간)에서 데이터를 read
- spi_slave는 통상적으로 내부에 Register Map 보유 ➔Address 와 각 Address에 해당하는 read/write Data를 가지고 있음 ➔ spi_slave는 spi_master로 부터 address와 data를 받아서 해당 Address 영역의 Data를 write 하거나, 해당 Address 영역의 Data를 spi_master에 전송
- 내부에 **4개의 Register Map**을 가진 ***spi_slave***는 **spi_master와의 통신**을 통하여 **4개의 Register map에 read/write** 하는 것을 구현



SPI Slave Implementation

➔ *Spec and Timing 정의*

➔ *Port 정의*

SPI Slave Implementation

➢ *spi_slave_exam_0.xpr*

➢ *SPI Slave Implementation* ➔ Code Implementation ➔ Port 정의

signal	in/out	size	description
reset	***input***	[0]	main reset, active low
clock	***input***	[0]	main clock, 100Mhz
ss	***input***	[0]	spi ss
sck	***input***	[0]	spi sck
mosi	***input***	[0]	spi mosi (master out slave in)
miso	***output***	[0]	spi miso (master in slave out)

- reset, clock : spi_slave 모듈에 사용되는 main reset, clock
- ss, sck, mosi, miso : spi master와 통신하는 spi data line

SPI Slave Implementation

➢ *spi_slave_exam_0.xpr*

➢ *SPI slave Implementation* ➔ Code Implementation

✓ ***Port & Register Map Address*** 정의 ➔ **Module의 Input, Output Port를 정의**

spi_slave . v (1)

```
1.  `timescale 1ns / 1ps
2.  `define          rUSER_REG1      8'h10
3.  `define          rUSER_REG2      8'h11
4.  `define          rUSER_REG3      8'h12
5.  `define          rUSER_REG4      8'h13
6.  module spi_slave(
7.              reset,
8.              clock,
9.              ss,
10.             sck,
11.             mosi ,
12.             miso
13.);
14.input                    reset ;
15.input                    clock ;
16.input                    ss    ;
17.input                    sck   ;
18.input                    mosi  ;
19.output                   miso  ;
```

❖ 2 ~ 5 : spi_slave 내부에 있는 4개의 Register Map Address

❖ 6 ~ 19 : input, output port 정의

➢ 컴파일러 지시어

❖ '<키워드> 형식 사용 : `define 과 `include 와 `timescale

`define : 텍스트 매크로를 정의하는 용도로 사용

❖ `include : 다른 verilog 소스 파일을 현재 소스 파일에 추가하는 용도로 사용 ➔ 일반적으로 헤더파일을 포함시키는 데 사용.

❖ `timescale : 모듈의 참조 시간 단위를 지정 ➔ `timescale <시간 단위>/<시간 정밀도> 형식으로 사용하며, 시간 단위는 시간 측정 단위이며, 시간 정밀도는 시뮬레이션에서 반올림된 지연의 정확도를 나타낸다.

❖ 예시

- `define SIZE 10 // `SIZE 를 10으로 사용
- `define END $stop // `END 를 $stop 으로 사용
- `define WORD_REG reg[31:0] // define a 32-bit register as 'WORD_REG reg32;
- `include headers.h
- `timescale 150ns/1ns // 시간 단위는 150ns 정밀도는 1ns

SPI Slave Implementation

➔ *Spec and Timing 정의*

➔ *Port 정의*

➔ *State 정의*

SPI Slave Implementation

➢ *spi_slave_exam_0.xpr*

➢ *SPI slave Implementation* ➔ Code Implementation

✓ ***State*** 정의 ➔ SM에서 사용할 State 정의

➔ *spi_slave* 모듈의 *SM*은 *7*개로 구성 (필요에 따라 다르게 구성 가능)

➔ 각각의 *state*는 입력되는 *ss, sck, mosi* 의 데이터 값에 따라서 이동

Next Slide

State	Transition	Description
IDLE	*ss* 신호가 *active(enable)* 되면 *SLAVEID* 상태로 이동	idle state
SLAVEID	***sck 8 clock*** 후에 ***WADDR or RADDR*** 로 이동	***slave id*** 수신
WADDR	*sck 8 clock* 후(*8bit Data*)에 WDATA 로	***write address*** 수신
WDATA	ss 신호가 *deactive(disable)* 되면 *DONE* 로 이동	***write data*** 수신
RADDR	*sck 8 clock* 후에 *RDATA* 로 이동	***read address*** 수신
RDATA	*ss* 신호가 *deactive(disable)* 되면 *DONE* 로 이동	***read data*** 수신
DONE	*done counter = 3* 이면 *IDLE* 로 이동	완료

SPI Slave Implementation

➢ *spi_slave_exam_0.xpr*

Continue

➢ *SPI slave Implementation* ➔ Code Implementation

✓ State 정의 ➔ SM에서 사용할 State 정의

➔ **spi_slave 모듈의 SM은 7개로 구성** (필요에 따라 다르게 구성 가능)

➔ 각각의 state는 입력되는 ss, sck, mosi 의 데이터 값에 따라서 이동

- IDLE : 통신이 없는 상태 ➔ ss 신호가 active 되면 SLAVEID 상태로 이동
- ***SLAVEID*** : **sck, mosi를 통하여 slave_id**를 받음 ➔ **slave_id 값이 0x64 이면 WADDR 상태**로 이동하고, **slave_id 값이 0x65이면 RADDR 상태**로 이동하고, **그 외의 값이면 IDLE로 이동** ➔ 0x64, 0x65가 아닐 때 IDLE 상태로 이동하는 것이 매우 중요
- WADDR : write address를 받음 ➔ 8bits 데이터를 모두 받으면 WDATA 상태로 이동
- WDATA : write data를 받음 ➔ 8bits 데이터를 모두 받으면 해당 Register의 값을 Update 하고, ss 신호가 disable 되면DONE 상태로 이동
- RADDR : read address를 받음 ➔ 8bits 데이터를 모두 받으면 RDATA 상태로 이동
- RDATA : 해당 Address의 데이터를, sck clock 에 맞게 miso로 전송 ➔ ss 신호가 disable 되면 DONE 상태로 이동
- DONE : 수 clock 후에 IDLE 상태로 이동

SPI Slave Implementation

➢ *spi_slave_exam_0.xpr*

➢ *SPI Slave Implementation* ➔ Code Implementation : State 정의 ➔ SM에서 사용할 State 정의

spi_slave. v (2)

```
20. //----------------------------------------
21. parameter     SLAVE_IDW  = 8'h64 ;
22. parameter     SLAVE_IDR  = 8'h65 ;
```

❖ 20 ~ 22 : 내부에서 사용하는 parameter 정의, slave_id for write / read

```
23. //----------------------------------------
24. // 1) define state
25. parameter          IDLE     = 3'd0 ;
26. parameter          SLAVEID  = 3'd1 ;
27. parameter          WADDR    = 3'd2 ;
28. parameter          WDATA    = 3'd3 ;
29. parameter          RADDR    = 3'd4 ;
30. parameter          RDATA    = 3'd5 ;
31. parameter          DONE     = 3'd6 ;
```

❖ 25 ~ 31 : state 정의

state	transition	description
IDLE	ss 신호가 active(enable) 되면 SLAVEID 상태로 이동	idle state
SLAVEID	sck 8 clock 후에 WADDR or RADDR 로 이동	slave id 수신
WADDR	sck 8 clock 후(8bit Data)에 WDATA 로	write address 수신
WDATA	ss 신호가 deactive(disable) 되면 DONE 로 이동	write data 수신
RADDR	sck 8 clock 후에 RDATA 로 이동	read address 수신
RDATA	ss 신호가 deactive(disable) 되면 DONE 로 이동	read data 수신
DONE	done counter = 3 이면 IDLE 로 이동	완료

```
32. //----------------------------------------
33. // 2) state flag
34. reg      [2:0]        s_state; // 7가지 state
35. wire                  s_idle   = (s_state==IDLE   ) ? 1'b1 : 1'b0;
36. wire                  s_slaveid= (s_state==SLAVEID) ? 1'b1 : 1'b0;
37. wire                  s_waddr  = (s_state==WADDR  ) ? 1'b1 : 1'b0;
38. wire                  s_wdata  = (s_state==WDATA  ) ? 1'b1 : 1'b0;
39. wire                  s_raddr  = (s_state==RADDR  ) ? 1'b1 : 1'b0;
40. wire                  s_rdata  = (s_state==RDATA  ) ? 1'b1 : 1'b0;
41. wire                  s_done   = (s_state==DONE   ) ? 1'b1 : 1'b0;
```

❖ 34 ~ 41 : state flag 정의

SPI Slave Implementation

- ➔ *Spec and Timing 정의*
- ➔ *Port 정의*
- ➔ *State 정의*
- ➔ ***Code Implementation***

SPI Slave Implementation

➢ spi_slave_exam_0.xpr

➢ SPI Slave Implementation ➔ Verilog Code Implementation ➔ **spi_slave . v (3)**

spi_slave . v (3)

```
33. // ----------------------------------------
34. // 3) code implementation
35. reg                 ss_1d, ss_2d ;
36. wire                ss_pedge =  ss_1d & ~ss_2d;
37. wire                ss_nedge = ~ss_1d &  ss_2d;
38. always @(posedge clock or negedge reset)
39. begin
40.         if(!reset)      begin
41.                         ss_1d <= 1'b1;
42.                         ss_2d <= 1'b1;
43.         end
44.         else            begin
45.                         ss_1d <= ss;
46.                         ss_2d <= ss_1d ;
47.         end
48. end
```

```
49. reg                 sck_1d, sck_2d ;
50. wire                sck_pedge =  sck_1d & ~sck_2d;
51. wire                sck_nedge = ~sck_1d &  sck_2d;
52. always @(posedge clock or negedge reset)
53. begin
54.         if(!reset)      begin
55.                         sck_1d <= 1'b0;
56.                         sck_2d <= 1'b0;
57.         end
58.         else            begin
59.                         sck_1d <= sck;
60.                         sck_2d <= sck_1d ;
61.         end
62. end
```

state	transition	description
IDLE	ss 신호가 active(enable) 되면 SLAVEID 상태로 이동	idle state
SLAVEID	sck 8 clock 후에 WADDR or RADDR 로 이동	slave id 수신
WADDR	sck 8 clock 후(8bit Data)에 WDATA 로	write address 수신
WDATA	ss 신호가 deactive(disable) 되면 DONE 로 이동	write data 수신
RADDR	sck 8 clock 후에 RDATA 로 이동	read address 수신
RDATA	ss 신호가 deactive(disable) 되면 DONE 로 이동	read data 수신
DONE	done counter = 3 이면 IDLE 로 이동	완료

❖ 35 ~ 48 : ss 신호의 *positive edge, negative edge* 를 검출

➔ *ss_nedge*는 *IDLE* ➔ *SLAVEID* 로 전환될 때 사용

➔ *ss_pedge*는 DONE ➔ IDLE 로 전환될 때 사용

➔ ss 신호 의 [1클럭 delay : ss_1d], [2클럭 delay : ss_2d]

❖ 49 ~ 62 : sck 신호의 positive edge, negative edge 를 검출

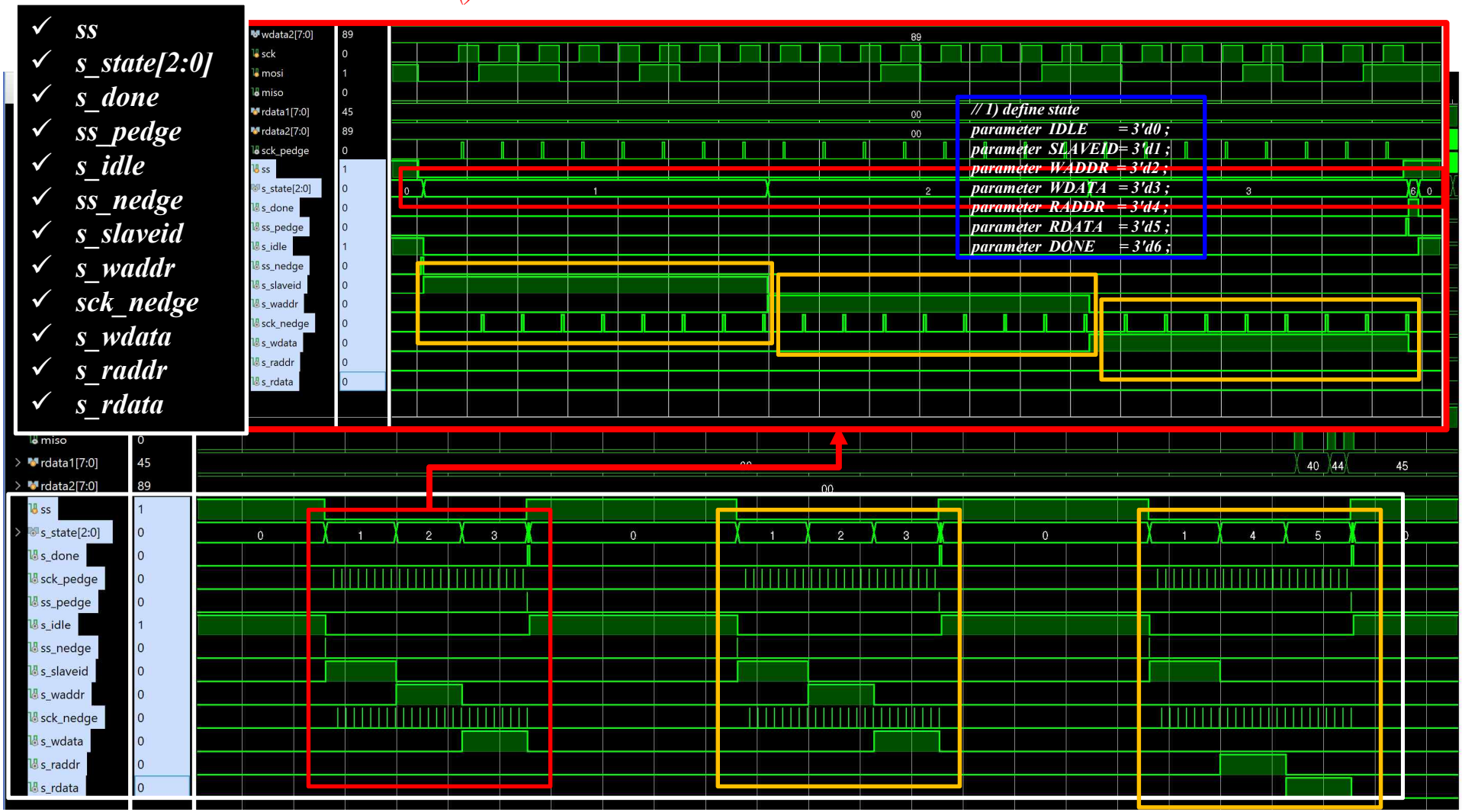
➔ 데이터 송수신시, 몇 번째 비트 값인지 check 하기 위한 용도와 다음 state 로 전환하는데 사용

SPI Slave Implementation

➢ *spi_slave_exam_0.xpr*

➢ *SPI slave Implementation* ➔ *Verilog Code Implementation* ➔ *Test Bench :* ***spi_slave_tb . v ()***

- ***Simulation Result Check ()***



SPI Slave Implementation

➢ spi_slave_exam_0.xpr

➢ SPI Slave Implementation ➔ Verilog Code Implementation ➔ **spi_slave . v (4)**

spi_slave . v (4)

```
63. reg                 mosi_1d, mosi_2d;
64. always @(posedge clock or negedge reset)
65. begin
66.             if(!reset)      begin
67.                             mosi_1d <= 1'b0;
68.                             mosi_2d <= 1'b0;
69.             end
70.             else            begin
71.                             mosi_1d <= mosi;
72.                             mosi_2d <= mosi_1d ;
73.             end
74. end
```

❖ 63 ~ 74 : spi 신호들은 보통 외부(다른 칩)에서 인가되는 경우가 많기때문에 신호들을 바로 사용하는 것보다는 내부 clock을 통하여 flip/flop 를 거쳐서 사용하는 것이 안정적 ➔ *mosi* 신호에 *2-clock delay* 주어서 *mosi_2d* 신호 사용

```
75. reg         [3:0]           sid_sckn_cnt;
76. always @(posedge clock or negedge reset)
77. begin
78.             if(!reset)      sid_sckn_cnt <= 4'b0;
79.             else            sid_sckn_cnt <= ~s_slaveid ? 4'b0 : sck_nedge ? (sid_sckn_cnt+1'b1) : sid_sckn_cnt ;
80. end
```

❖ 75 ~ 80 : slaveid state 에서 사용하기 위하여 sck 신호의 negative edge 를 counting

```
81. reg         [7:0]           slave_id;
82. always @(posedge clock or negedge reset)
83. begin
84.             if(!reset)      slave_id <= 8'b0;
85.             else            begin
86.                             slave_id[7] <= s_idle ? 1'b0 : (s_slaveid & sck_pedge & (sid_sckn_cnt==4'd0)) ? mosi_2d : slave_id[7] ;
87.                             slave_id[6] <= s_idle ? 1'b0 : (s_slaveid & sck_pedge & (sid_sckn_cnt==4'd1)) ? mosi_2d : slave_id[6] ;
88.                             slave_id[5] <= s_idle ? 1'b0 : (s_slaveid & sck_pedge & (sid_sckn_cnt==4'd2)) ? mosi_2d : slave_id[5] ;
89.                             slave_id[4] <= s_idle ? 1'b0 : (s_slaveid & sck_pedge & (sid_sckn_cnt==4'd3)) ? mosi_2d : slave_id[4] ;
90.                             slave_id[3] <= s_idle ? 1'b0 : (s_slaveid & sck_pedge & (sid_sckn_cnt==4'd4)) ? mosi_2d : slave_id[3] ;
91.                             slave_id[2] <= s_idle ? 1'b0 : (s_slaveid & sck_pedge & (sid_sckn_cnt==4'd5)) ? mosi_2d : slave_id[2] ;
92.                             slave_id[1] <= s_idle ? 1'b0 : (s_slaveid & sck_pedge & (sid_sckn_cnt==4'd6)) ? mosi_2d : slave_id[1] ;
93.                             slave_id[0] <= s_idle ? 1'b0 : (s_slaveid & sck_pedge & (sid_sckn_cnt==4'd7)) ? mosi_2d : slave_id[0] ;
94.             end
95. end
```

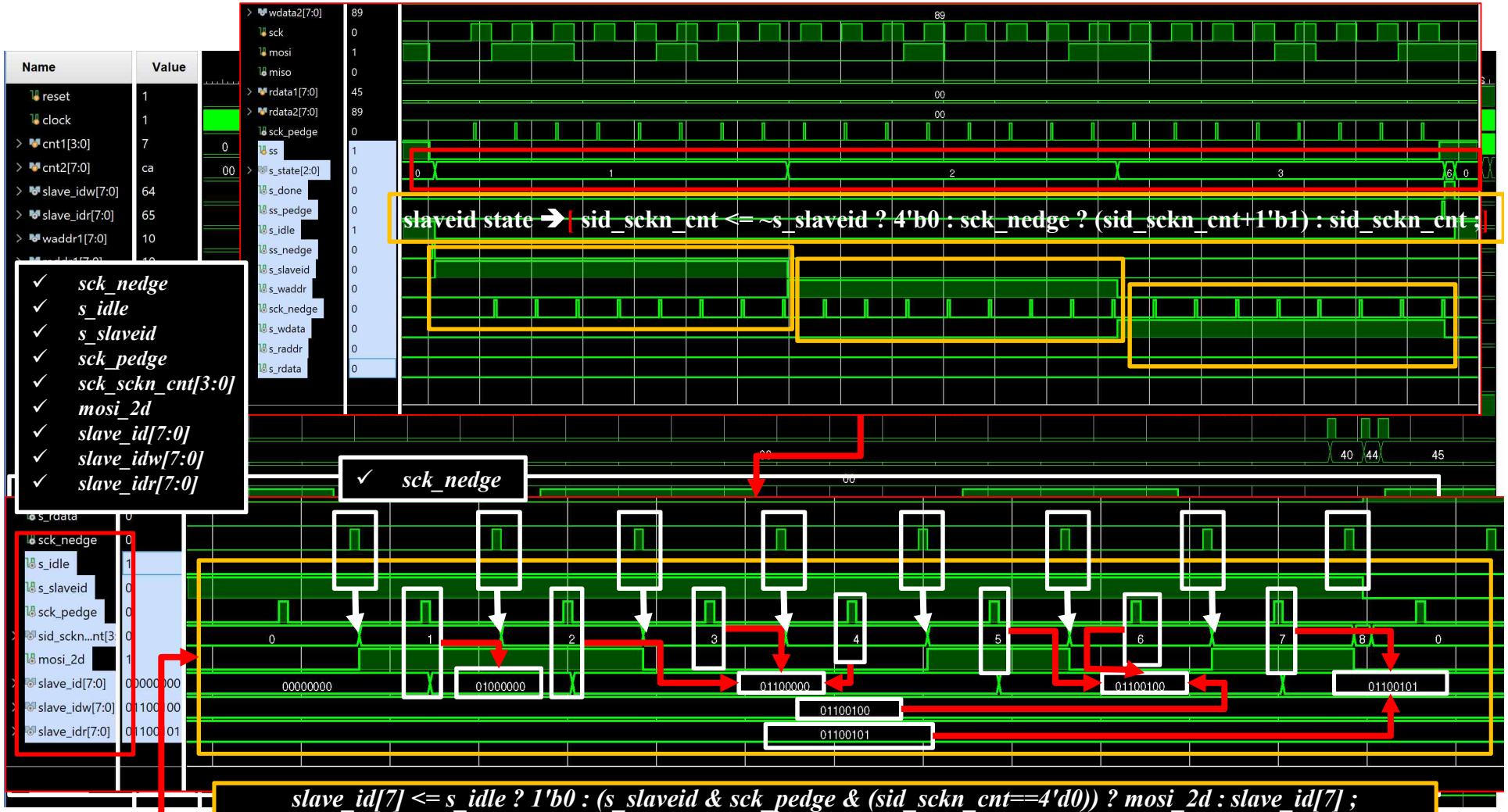
❖ 81 ~ 95 : mosi 신호로 부터 slave_id 를 검출합니다. mosi 신호는 sck positive edge 에서 read

SPI Slave Implementation

➢ *spi_slave_exam_0.xpr*

➢ *SPI slave Implementation* ➔ *Verilog Code Implementation* ➔ *Test Bench :* ***spi_slave_tb . v ()***

- ***Simulation Result Check ()***



SPI Slave Implementation

➢ spi_slave_exam_0.xpr

➢ SPI Slave Implementation ➔ Verilog Code Implementation ➔ **spi_slave . v (5)**

spi_slave . v (5)

```
96. reg         [3:0]       wa_sckn_cnt;
97. always @(posedge clock or negedge reset)
98. begin
99.             if(!reset)      wa_sckn_cnt <= 4'b0;
100.            else            wa_sckn_cnt <= ~s_waddr ? 4'b0 : sck_nedge ? (wa_sckn_cnt+1'b1) : wa_sckn_cnt ;
101.end
```

❖ 96 ~ 101 : waddr state 에서 사용하기 위하여 sck 신호의 *negative edge*를 counting

```
102.reg         [7:0]       waddr;
103.always @(posedge clock or negedge reset)
104.begin
105.            if(!reset)      waddr <= 8'b0;
106.            else            begin
107.                            waddr[7] <= s_idle ? 1'b0 : (s_waddr & sck_pedge & (wa_sckn_cnt==4'd0)) ? mosi_2d : waddr[7] ;
108.                            waddr[6] <= s_idle ? 1'b0 : (s_waddr & sck_pedge & (wa_sckn_cnt==4'd1)) ? mosi_2d : waddr[6] ;
109.                            waddr[5] <= s_idle ? 1'b0 : (s_waddr & sck_pedge & (wa_sckn_cnt==4'd2)) ? mosi_2d : waddr[5] ;
110.                            waddr[4] <= s_idle ? 1'b0 : (s_waddr & sck_pedge & (wa_sckn_cnt==4'd3)) ? mosi_2d : waddr[4] ;
111.                            waddr[3] <= s_idle ? 1'b0 : (s_waddr & sck_pedge & (wa_sckn_cnt==4'd4)) ? mosi_2d : waddr[3] ;
112.                            waddr[2] <= s_idle ? 1'b0 : (s_waddr & sck_pedge & (wa_sckn_cnt==4'd5)) ? mosi_2d : waddr[2] ;
113.                            waddr[1] <= s_idle ? 1'b0 : (s_waddr & sck_pedge & (wa_sckn_cnt==4'd6)) ? mosi_2d : waddr[1] ;
114.                            waddr[0] <= s_idle ? 1'b0 : (s_waddr & sck_pedge & (wa_sckn_cnt==4'd7)) ? mosi_2d : waddr[0] ;
115.            end
116.end
```

❖ 102 ~ 116 : mosi 신호로부터 waddr 를 검출

➔ mosi 신호는 sck positive edge 에서 read

```
117.reg         [3:0]       wd_sckn_cnt;
118.always @(posedge clock or negedge reset)
119.begin
120.            if(!reset)      wd_sckn_cnt <= 4'b0;
121.            else            wd_sckn_cnt <= ~s_wdata ? 4'b0 : sck_nedge ? (wd_sckn_cnt+1'b1) : wd_sckn_cnt ;
122.end
```

❖ 117 ~ 122 : *wdata state* 에서 사용하기 위하여 sck 신호의 *negative edge*를 counting

```
123.reg         [7:0]       wdata;
124.always @(posedge clock or negedge reset)
125.begin
126.            if(!reset)      wdata <= 8'b0;
127.            else            begin
128.                            wdata[7] <= s_idle ? 1'b0 : (s_wdata & sck_pedge & (wd_sckn_cnt==4'd0 )) ? mosi_2d : wdata[7] ;
129.                            wdata[6] <= s_idle ? 1'b0 : (s_wdata & sck_pedge & (wd_sckn_cnt==4'd1 )) ? mosi_2d : wdata[6] ;
130.                            wdata[5] <= s_idle ? 1'b0 : (s_wdata & sck_pedge & (wd_sckn_cnt==4'd2 )) ? mosi_2d : wdata[5] ;
131.                            wdata[4] <= s_idle ? 1'b0 : (s_wdata & sck_pedge & (wd_sckn_cnt==4'd3 )) ? mosi_2d : wdata[4] ;
132.                            wdata[3] <= s_idle ? 1'b0 : (s_wdata & sck_pedge & (wd_sckn_cnt==4'd4 )) ? mosi_2d : wdata[3] ;
133.                            wdata[2] <= s_idle ? 1'b0 : (s_wdata & sck_pedge & (wd_sckn_cnt==4'd5 )) ? mosi_2d : wdata[2] ;
134.                            wdata[1] <= s_idle ? 1'b0 : (s_wdata & sck_pedge & (wd_sckn_cnt==4'd6 )) ? mosi_2d : wdata[1] ;
135.                            wdata[0] <= s_idle ? 1'b0 : (s_wdata & sck_pedge & (wd_sckn_cnt==4'd7 )) ? mosi_2d : wdata[0] ;
136.            end
137.end
```

❖ 123 ~137 : mosi 신호로 부터 wdata를 검출

➔ mosi 신호는 sck positive edge 에서 read

SPI Slave Implementation

➢ *spi_slave_exam_0.xpr*

➢ *SPI Slave Implementation* ➔ *Verilog Code Implementation* ➔ ***spi_slave . v (5)***

spi_slave . v (5)

```
96.  reg          [3:0]          wa_sckn_cnt;
97.  always @(posedge clock or negedge reset)
98.  begin
99.               if(!reset)     wa_sckn_cnt <= 4'b0;
100.              else           wa_sckn_cnt <= ~s_waddr ? 4'b0 : sck_nedge ? (wa_sckn_cnt+1'b1) : wa_sckn_cnt ;
101.end

102.reg           [7:0]          waddr;
103.always @(posedge clock or negedge reset)
104.begin
105.              if(!reset)     waddr <= 8'b0;
106.              else           begin
107.                             waddr[7] <= s_idle ? 1'b0 : (s_waddr & sck_pedge & (wa_sckn_cnt==4'd0)) ? mosi_2d : waddr[7] ;
108.                             waddr[6] <= s_idle ? 1'b0 : (s_waddr & sck_pedge & (wa_sckn_cnt==4'd1)) ? mosi_2d : waddr[6] ;
109.                             waddr[5] <= s_idle ? 1'b0 : (s_waddr & sck_pedge & (wa_sckn_cnt==4'd2)) ? mosi_2d : waddr[5] ;
110.                             waddr[4] <= s_idle ? 1'b0 : (s_waddr & sck_pedge & (wa_sckn_cnt==4'd3)) ? mosi_2d : waddr[4] ;
111.                             waddr[3] <= s_idle ? 1'b0 : (s_waddr & sck_pedge & (wa_sckn_cnt==4'd4)) ? mosi_2d : waddr[3] ;
112.                             waddr[2] <= s_idle ? 1'b0 : (s_waddr & sck_pedge & (wa_sckn_cnt==4'd5)) ? mosi_2d : waddr[2] ;
113.                             waddr[1] <= s_idle ? 1'b0 : (s_waddr & sck_pedge & (wa_sckn_cnt==4'd6)) ? mosi_2d : waddr[1] ;
114.                             waddr[0] <= s_idle ? 1'b0 : (s_waddr & sck_pedge & (wa_sckn_cnt==4'd7)) ? mosi_2d : waddr[0] ;
115.              end
116.end

117.reg           [3:0]          wd_sckn_cnt;
118.always @(posedge clock or negedge reset)
```

❖ 96 ~ 101 : waddr state 에서 사용하기 위하여 sck 신호의 negative edge를 counting

❖ 102 ~ 116 : mosi 신호로부터 waddr 를 검출

➔ mosi 신호는 sck positive edge 에서 read


```
135.              wdata[0] <= s_idle ? 1'b0 : (s_wdata & sck_pedge & (wd_sckn_cnt==4'd7)) ? mosi_2d : wdata[0] ;
136.              end
137.end
```

SPI Slave Implementation

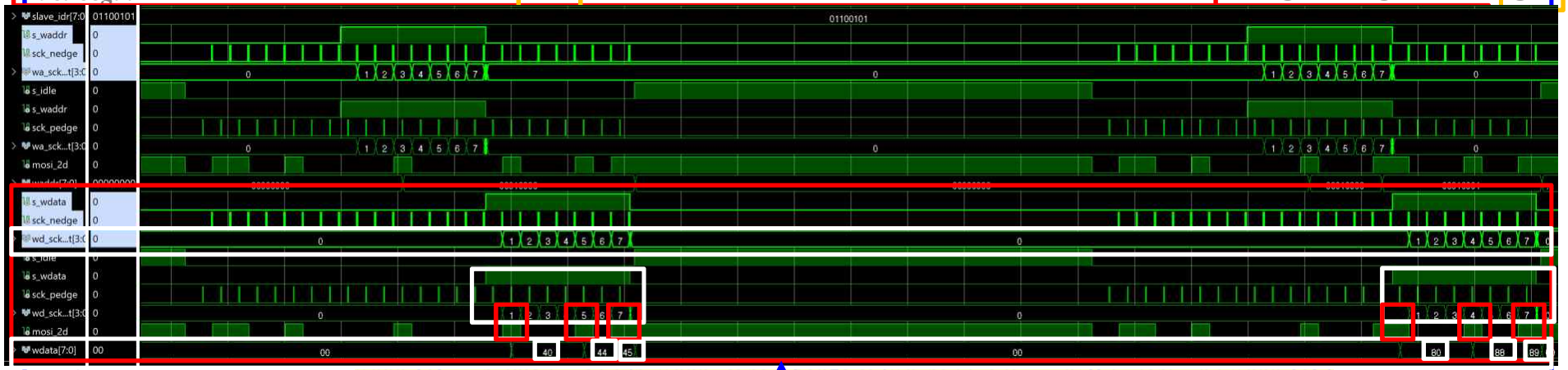
➢ *spi_slave_exam_0.xpr*

➢ *SPI Slave Implementation ➔ Verilog Code Implementation ➔ spi_slave . v (5)*

spi_slave . v (5)

```
96.  reg          [3:0]          wa_sckn_cnt;
97.  always @(posedge clock or negedge reset)
98.  begin
```

❖ 96 ~ 101 : waddr state 에서 사용하기 위하여 sck 신호의 negative edge를 counting




```
115.             end
116.end
```

```
117.reg          [3:0]          wd_sckn_cnt;
118.always @(posedge clock or negedge reset)
119.begin
120.             if(!reset)     wd_sckn_cnt <= 4'b0;
121.             else           wd_sckn_cnt <= ~s_wdata ? 4'b0 : sck_nedge ? (wd_sckn_cnt+1'b1) : wd_sckn_cnt ;
122.end
```

❖ 117 ~ 122 : waddr state 에서 사용하기 위하여 sck 신호의 negative edge를 counting

```
123.reg          [7:0]          wdata;
124.always @(posedge clock or negedge reset)
125.begin
126.             if(!reset)     wdata <= 8'b0;
127.             else           begin
128.                            wdata[7] <= s_idle ? 1'b0 : (s_wdata & sck_pedge & (wd_sckn_cnt==4'd0 )) ? mosi_2d : wdata[7] ;
129.                            wdata[6] <= s_idle ? 1'b0 : (s_wdata & sck_pedge & (wd_sckn_cnt==4'd1 )) ? mosi_2d : wdata[6] ;
130.                            wdata[5] <= s_idle ? 1'b0 : (s_wdata & sck_pedge & (wd_sckn_cnt==4'd2 )) ? mosi_2d : wdata[5] ;
131.                            wdata[4] <= s_idle ? 1'b0 : (s_wdata & sck_pedge & (wd_sckn_cnt==4'd3 )) ? mosi_2d : wdata[4] ;
132.                            wdata[3] <= s_idle ? 1'b0 : (s_wdata & sck_pedge & (wd_sckn_cnt==4'd4 )) ? mosi_2d : wdata[3] ;
133.                            wdata[2] <= s_idle ? 1'b0 : (s_wdata & sck_pedge & (wd_sckn_cnt==4'd5 )) ? mosi_2d : wdata[2] ;
134.                            wdata[1] <= s_idle ? 1'b0 : (s_wdata & sck_pedge & (wd_sckn_cnt==4'd6 )) ? mosi_2d : wdata[1] ;
135.                            wdata[0] <= s_idle ? 1'b0 : (s_wdata & sck_pedge & (wd_sckn_cnt==4'd7 )) ? mosi_2d : wdata[0] ;
136.             end
137.end
```

❖ 123 ~137 : mosi 신호로 부터 slave_id 를 검출합니다. mosi 신호는 sck positive edge 에서 read

SPI Slave Implementation

➢ *spi_slave_exam_0.xpr*

➢ *SPI Slave Implementation* ➔ *Verilog Code Implementation* ➔ ***spi_slave . v (6)***

spi_slave . v (6)

```
138.reg          [3:0]          ra_sckn_cnt;
139.always @(posedge clock or negedge reset)
140.begin
141.             if(!reset)     ra_sckn_cnt <= 4'b0;
142.             else           ra_sckn_cnt <= ~s_raddr ? 4'b0 : sck_nedge ? (ra_sckn_cnt+1'b1) : ra_sckn_cnt ;
143.end
```

❖ 138 ~143 : *raddr* state 에서 사용하기 위하여 *sck* 신호의 *negative edge*를 counting

```
144.reg          [7:0]          raddr;
145.always @(posedge clock or negedge reset)
146.begin
147.             if(!reset)     raddr <= 8'b0;
148.             else           begin
149.                            raddr[7] <= s_idle ? 1'b0 : (s_raddr & sck_pedge & (ra_sckn_cnt==4'd0)) ? mosi_2d : raddr[7] ;
150.                            raddr[6] <= s_idle ? 1'b0 : (s_raddr & sck_pedge & (ra_sckn_cnt==4'd1)) ? mosi_2d : raddr[6] ;
151.                            raddr[5] <= s_idle ? 1'b0 : (s_raddr & sck_pedge & (ra_sckn_cnt==4'd2)) ? mosi_2d : raddr[5] ;
152.                            raddr[4] <= s_idle ? 1'b0 : (s_raddr & sck_pedge & (ra_sckn_cnt==4'd3)) ? mosi_2d : raddr[4] ;
153.                            raddr[3] <= s_idle ? 1'b0 : (s_raddr & sck_pedge & (ra_sckn_cnt==4'd4)) ? mosi_2d : raddr[3] ;
154.                            raddr[2] <= s_idle ? 1'b0 : (s_raddr & sck_pedge & (ra_sckn_cnt==4'd5)) ? mosi_2d : raddr[2] ;
155.                            raddr[1] <= s_idle ? 1'b0 : (s_raddr & sck_pedge & (ra_sckn_cnt==4'd6)) ? mosi_2d : raddr[1] ;
156.                            raddr[0] <= s_idle ? 1'b0 : (s_raddr & sck_pedge & (ra_sckn_cnt==4'd7)) ? mosi_2d : raddr[0] ;
157.             end
158.end
```

❖ 144 ~ 158 : mosi 신호로부터 raddr 를 검출

➔ mosi 신호는 *sck positive edge*에서 read

❖ 159 ~ 164 : *raddr* state 에서 사용하기 위하여 *sck* 신호의 *negative edge*를 counting

```
159.reg          [3:0]          rd_sckn_cnt;
160.always @(posedge clock or negedge reset)
161.begin
162.             if(!reset)     rd_sckn_cnt <= 4'b0;
163.             else           rd_sckn_cnt <= ~s_rdata ? 4'b0 : sck_nedge ? (rd_sckn_cnt+1'b1) : rd_sckn_cnt;
164.end
```

```
165.reg                         s_raddr_1d, s_raddr_2d;
166.wire                        s_raddr_nedge = ~s_raddr_1d & s_raddr_2d;
167.always @(posedge clock or negedge reset)
168.begin
169.             if(!reset)     begin
170.                            s_raddr_1d <= 1'b0;
171.                            s_raddr_2d <= 1'b0;
172.             end
173.             else           begin
174.                            s_raddr_1d <= s_raddr;
175.                            s_raddr_2d <= s_raddr_1d;
176.             end
177.end
```

❖ 165 ~ 177 : rdata 검출을 위하여 관련된 신호들의 타이밍을 맞추기 위하여 추가된 코드 ➔ [통상적으로 타이밍을 맞추기 위하여, 코딩 → simulation으로 타이밍 확인 → 타이밍이 안 맞으면 *delay(1-clock flip/flop)*를 추가하여 타이밍을 맞추고, 다시 simulation으로 타이밍 확인 → 이러한 과정을 반복해서 진행]

➔ *s_raddr_1d, s_raddr_2d, sck_nedge_1d, sck_pedge_1d* 이러한 신호들이 타이밍을 맞추기 위하여 추가된 신호

SPI Slave Implementation

➢ *spi_slave_exam_0.xpr*

➢ *SPI Slave Implementation ➔ Verilog Code Implementation ➔ spi_slave . v (6)*

spi_slave . v (6)

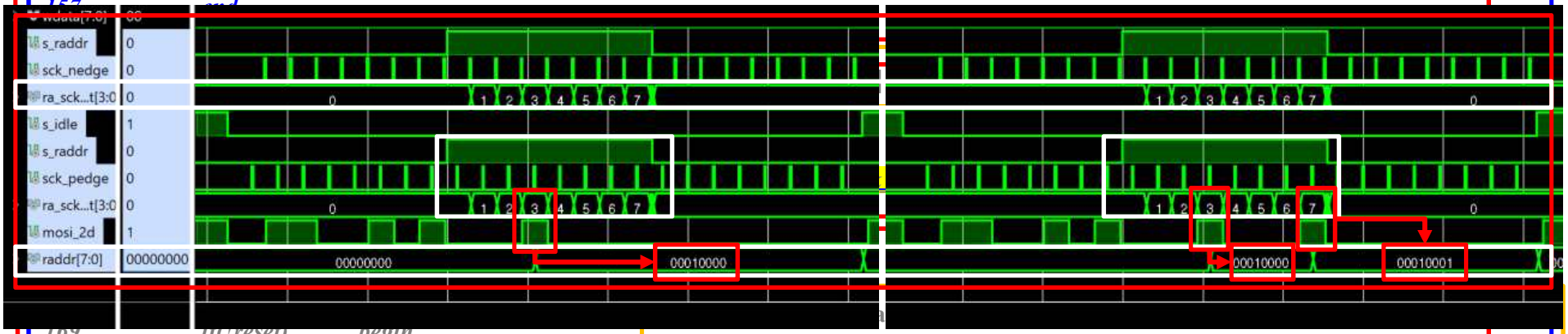
```
138.reg         [3:0]       ra_sckn_cnt;
139.always @(posedge clock or negedge reset)
140.begin
141.            if(!reset)      ra_sckn_cnt <= 4'b0;
142.            else            ra_sckn_cnt <= ~s_raddr ? 4'b0 : sck_nedge ? (ra_sckn_cnt+1'b1) : ra_sckn_cnt ;
143.end
```

❖ 138 ~143 : raddr state 에서 사용하기 위하여 sck 신호의 negative edge를 counting

```
144.reg         [7:0]       raddr;
145.always @(posedge clock or negedge reset)
146.begin
147.            if(!reset)      raddr <= 8'b0;
148.            else            begin
149.                            raddr[7] <= s_idle ? 1'b0 : (s_raddr & sck_pedge & (ra_sckn_cnt==4'd0)) ? mosi_2d : raddr[7] ;
150.                            raddr[6] <= s_idle ? 1'b0 : (s_raddr & sck_pedge & (ra_sckn_cnt==4'd1)) ? mosi_2d : raddr[6] ;
151.                            raddr[5] <= s_idle ? 1'b0 : (s_raddr & sck_pedge & (ra_sckn_cnt==4'd2)) ? mosi_2d : raddr[5] ;
152.                            raddr[4] <= s_idle ? 1'b0 : (s_raddr & sck_pedge & (ra_sckn_cnt==4'd3)) ? mosi_2d : raddr[4] ;
153.                            raddr[3] <= s_idle ? 1'b0 : (s_raddr & sck_pedge & (ra_sckn_cnt==4'd4)) ? mosi_2d : raddr[3] ;
154.                            raddr[2] <= s_idle ? 1'b0 : (s_raddr & sck_pedge & (ra_sckn_cnt==4'd5)) ? mosi_2d : raddr[2] ;
155.                            raddr[1] <= s_idle ? 1'b0 : (s_raddr & sck_pedge & (ra_sckn_cnt==4'd6)) ? mosi_2d : raddr[1] ;
156.                            raddr[0] <= s_idle ? 1'b0 : (s_raddr & sck_pedge & (ra_sckn_cnt==4'd7)) ? mosi_2d : raddr[0] ;
```

❖ 144 ~ 158 : mosi 신호로부터 raddr 를 검출

➔ mosi 신호는 sck positive edge에서 read




```
169.            if(!reset)      begin
170.                            s_raddr_1d <= 1'b0;
171.                            s_raddr_2d <= 1'b0;
172.            end
173.            else            begin
174.                            s_raddr_1d <= s_raddr;
175.                            s_raddr_2d <= s_raddr_1d;
176.            end
177.end
```

된 코드 ➔ [통상적으로 타이밍을 맞추기 위하여, 코딩 → simulation으로 타이밍 확인 → 타이밍이 안 맞으면 delay(1-clock flip/flop)를 추가하여 타이밍을 맞추고, 다시 simulation으로 타이밍 확인 → 이러한 과정을 반복해서 진행]

➔ s_raddr_1d, s_raddr_2d, sck_nedge_1d, sck_pedge_1d 이러한 신호들이 타이밍을 맞추기 위하여 추가된 신호

SPI Slave Implementation

➢ spi_slave_exam_0.xpr

➢ SPI Slave Implementation ➔ Verilog Code Implementation ➔ **spi_slave . v (7)**

spi_slave . v (7)

```
178.reg                         sck_nedge_1d;
179.always @(posedge clock or negedge reset)
180.begin
181.            if(!reset)      sck_nedge_1d <= 1'b0;
182.            else            sck_nedge_1d <= sck_nedge;
183.end
```

❖ 178 ~ 183 : rdata 검출 위해 관련된 신호들의 타이밍을 맞추기 위하여 추가된 코드

```
184.reg                         sck_pedge_1d;
185.always @(posedge clock or negedge reset)
186.begin
187.            if(!reset)      sck_pedge_1d <= 1'b0;
188.            else            sck_pedge_1d <= sck_pedge;
189.end
```

❖ 184 ~ 189 : rdata 검출 위해 관련된 신호들의 타이밍을 맞추기 위하여 추가된 코드

```
190.reg         [7:0]           rdata;
191.reg                         miso;
192.always @(posedge clock or negedge reset)
193.begin
194.            if(!reset)      miso <= 1'b0;
195.            else            miso <= s_idle ? 1'b0 :
196.                            (sck_nedge_1d & (rd_sckn_cnt==5'd0 )) ? rdata[7] :
197.                            (sck_nedge_1d & (rd_sckn_cnt==5'd1 )) ? rdata[6] :
198.                            (sck_nedge_1d & (rd_sckn_cnt==5'd2 )) ? rdata[5] :
199.                            (sck_nedge_1d & (rd_sckn_cnt==5'd3 )) ? rdata[4] :
200.                            (sck_nedge_1d & (rd_sckn_cnt==5'd4 )) ? rdata[3] :
201.                            (sck_nedge_1d & (rd_sckn_cnt==5'd5 )) ? rdata[2] :
202.                            (sck_nedge_1d & (rd_sckn_cnt==5'd6 )) ? rdata[1] :
203.                            (sck_nedge_1d & (rd_sckn_cnt==5'd7 )) ? rdata[0] : miso ;
204.end
```

❖ 190 ~ 204 : miso 신호로부터 rdata를 검출

```
205.reg         [1:0]           done_cnt;
206.always @(posedge clock or negedge reset)
207.begin
208.            if(!reset)      done_cnt <= 2'b0;
209.            else            done_cnt <= ~s_done ? 2'b0 : done_cnt+1'b1 ;
210.end
```

❖ 205 ~ 210 : done state에서 사용하기 위한 counter

SPI Slave Implementation

➢ spi_slave_exam.xpr

➢ SPI Slave Implementation ➔ Verilog Code Implementation ➔ **spi_slave . v (7)**

spi_slave . v (7)

```
178.reg                         sck_nedge_1d;
179.always @(posedge clock or negedge reset)
180.begin
181.            if(!reset)      sck_nedge_1d <= 1'b0;
182.            else            sck_nedge_1d <= sck_nedge;
183.end
```

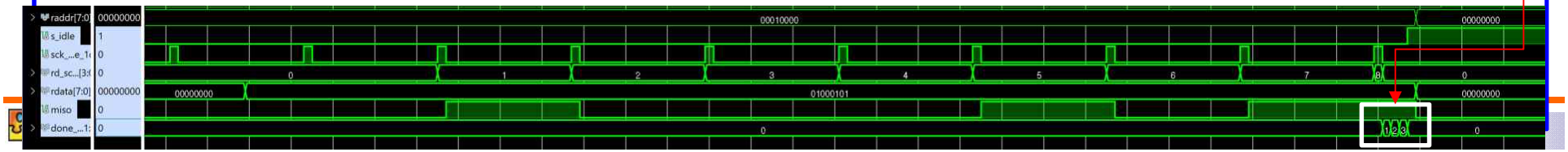
❖ 178 ~ 183 : rdata 검출 위해 관련된 신호들의 타이밍을 맞추기 위하여 추가된 코드


```
190.reg         [7:0]           rdata;
191.reg                         miso;
192.always @(posedge clock or negedge reset)
193.begin
194.            if(!reset)      miso <= 1'b0;
195.            else            miso <= s_idle ? 1'b0 :
196.                            (sck_nedge_1d & (rd_sckn_cnt==5'd0 )) ? rdata[7] :
197.                            (sck_nedge_1d & (rd_sckn_cnt==5'd1 )) ? rdata[6] :
198.                            (sck_nedge_1d & (rd_sckn_cnt==5'd2 )) ? rdata[5] :
199.                            (sck_nedge_1d & (rd_sckn_cnt==5'd3 )) ? rdata[4] :
200.                            (sck_nedge_1d & (rd_sckn_cnt==5'd4 )) ? rdata[3] :
201.                            (sck_nedge_1d & (rd_sckn_cnt==5'd5 )) ? rdata[2] :
202.                            (sck_nedge_1d & (rd_sckn_cnt==5'd6 )) ? rdata[1] :
203.                            (sck_nedge_1d & (rd_sckn_cnt==5'd7 )) ? rdata[0] : miso ;
204.end
```

❖ 190 ~ 204 : miso 신호로부터 rdata를 검출

```
205.reg         [1:0]           done_cnt;
206.always @(posedge clock or negedge reset)
207.begin
208.            if(!reset)      done_cnt <= 2'b0;
209.            else            done_cnt <= ~s_done ? 2'b0 : done_cnt+1'b1 ;
210.end
```

❖ 205 ~ 210 : done state에서 사용하기 위한 counter



SPI Slave Implementation

➔ Spec and Timing 정의

➔ Port 정의

➔ State 정의

➔ Code Implementation

➔ State Transition

SPI Slave Implementation

➢ spi_slave_exam_0.xpr

➢ SPI Slave Implementation ➔ Verilog Code Implementation ➔ spi_slave . v (8)

spi_slave . v (8)

```
211.// ----------------------------------------
212.// 4) state transition
213.always @(posedge clock or negedge reset)
214.begin
215.            if(!reset)          begin
216.                                s_state <= 3'h0;
217.            end
218.                                begin
219.                                s_state <= (s_idle    & ss_nedge ) ? SLAVEID :
220.                                (s_slaveid & (sid_sckn_cnt==4'd8)) ? ((slave_id==SLAVE_IDW) ? WADDR : (slave_id==SLAVE_IDR) ? RADDR : IDLE) :
221.
222.                                (s_waddr   & (wa_sckn_cnt==4'd8) ) ? WDATA :
223.                                (s_wdata   & ss_nedge ) ? DONE  :
224.                                (s_raddr   & (ra_sckn_cnt==4'd8) ) ? RDATA :
225.                                (s_rdata   & ss_nedge ) ? DONE  :
226.                                (s_done    & (done_cnt==2'd3)) ? IDLE  : s_state ;
227.            end
228.end
```

```
// 1) define state
parameter IDLE      = 3'd0 ;
parameter SLAVEID   = 3'd1 ;
parameter WADDR     = 3'd2 ;
parameter WDATA     = 3'd3 ;
parameter RADDR     = 3'd4 ;
parameter RDATA     = 3'd5 ;
parameter DONE      = 3'd6 ;
```

❖ 211 ~ 227 : 상태 전이를 구현 ➔ idle 상태에서 ss_nedge(negative edge)가 발생하면 slave_id 상태로 ➔ slaveid 상태에서는 slave_id 데이터(8bits)을 수신한 후, 값에 따라 waddr (write address), raddr (read address) 상태로 전환 ➔ waddr, raddr 상태에서는 address 데이터 (8bits)을 수신한 후 각각 wdata, rdata 상태로 전환 ➔

➔ wdata 상태에서는 data (8bits)을 수신한 후 done 상태로 전환 ➔ , rdata 상태에서는 miso로 데이터를 전송한 후에 done 상태로 전환 ➔ done 상태에서는 3-clock 후에 idle 상태로 전환

```
229.reg         [7:0]           user_reg1, user_reg2, user_reg3, user_reg4 ;
230.always @(posedge clock or negedge reset)
231.begin
232.            if(!reset)      begin
233.                            user_reg1  <= 8'b0;     user_reg2  <= 8'b0;     user_reg3  <= 8'b0;     user_reg4  <= 8'b0;
234.            end
235.            else            begin
236.                            user_reg1  <= (s_done & (waddr==`rUSER_REG1 )) ? wdata : user_reg1 ;
237.                            user_reg2  <= (s_done & (waddr==`rUSER_REG2 )) ? wdata : user_reg2 ;
238.                            user_reg3  <= (s_done & (waddr==`rUSER_REG3 )) ? wdata : user_reg3 ;
239.                            user_reg4  <= (s_done & (waddr==`rUSER_REG4 )) ? wdata : user_reg4 ;
240.            end
241.end
```

❖ 229 ~ 241 : spi master로 부터 수신한 wdata 값을 Address 에 따라서 사용자 Register 에 저장

```
242.always @(posedge clock or negedge reset)
243.begin
244.            if(!reset)      rdata <= 16'b0;
245.            else            rdata <= s_idle ? 16'b0 :
246.                            (s_raddr & sck_pedge_1d & (ra_sckn_cnt==4'd7) & (raddr == `rUSER_REG1)) ? user_reg1 :
247.                            (s_raddr & sck_pedge_1d & (ra_sckn_cnt==4'd7) & (raddr == `rUSER_REG2)) ? user_reg2 :
248.                            (s_raddr & sck_pedge_1d & (ra_sckn_cnt==4'd7) & (raddr == `rUSER_REG3)) ? user_reg3 :
249.                            (s_raddr & sck_pedge_1d & (ra_sckn_cnt==4'd7) & (raddr == `rUSER_REG4)) ? user_reg4 :
250.                            rdata ;
251.end
252.endmodule
```

❖ 242 ~ 252 : : read address 에 따라 사용자 Register 값 전송 위해 rdata 에 저장
➔ [주의] rdata값은 miso로 데이터를 전송하기 위하여 미리 준비되어야 함

SPI Slave Implementation

➢ spi_slave_exam_0.xpr

➢ SPI Slave Implementation ➔ Verilog Code Implementation ➔ spi_slave . v (8)

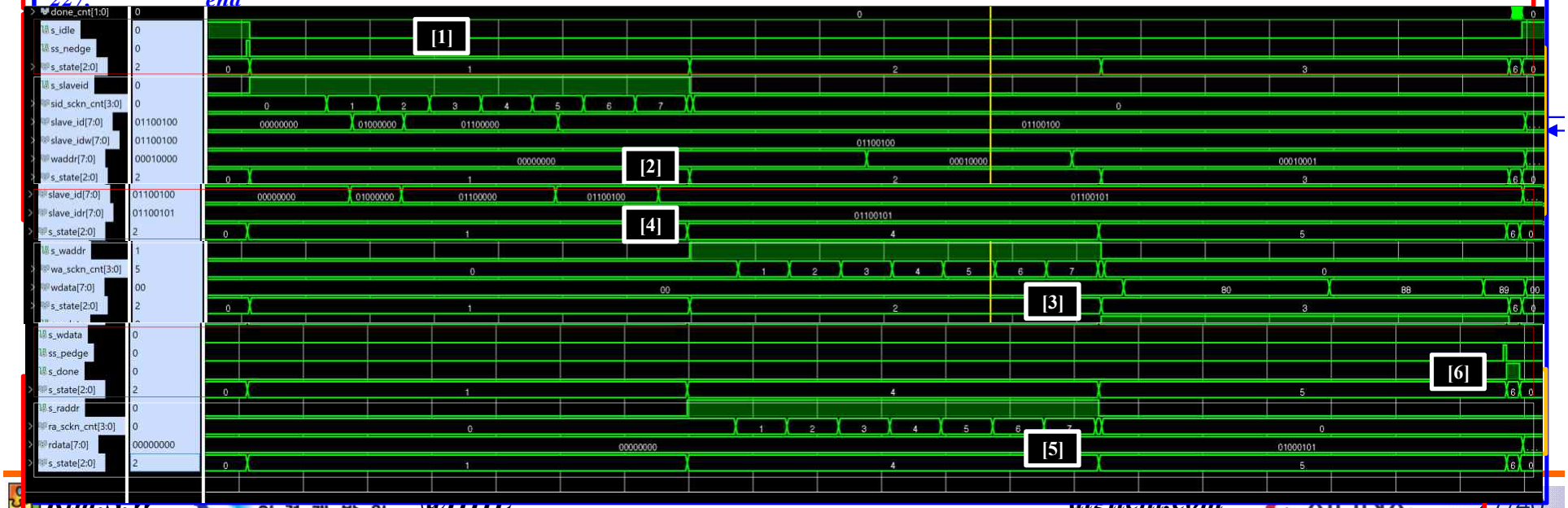
spi_slave . v (8)

```
211.// ----------------------------------------
212.// 4) state transition
213.always @(posedge clock or negedge reset)
214.begin
215.        if(!reset)      begin
216.                        s_state <= 3'h0;
217.        end
218.        begin
219.        s_state <= (s_idle    & ss_nedge ) ? SLAVEID :
220.        (s_slaveid & (sid_sckn_cnt==4'd8)) ? ((slave_id==SLAVE_IDW) ? WADDR :
221.        (slave_id==SLAVE_IDR) ? RADDR : IDLE) :
222.        (s_waddr   & (wa_sckn_cnt==4'd8) ) ? WDATA
223.        (s_wdata   & ss_pedge ) ? DONE  :
224.        (s_raddr   & (ra_sckn_cnt==4'd8) ) ? RDATA :
225.        (s_rdata   & ss_pedge ) ? DONE  :
226.        (s_done    & (done_cnt==2'd3)) ? IDLE  : s_state ;
227.        end
```

```
// 1) define state
parameter IDLE     = 3'd0 ;
parameter SLAVEID  = 3'd1 ;
parameter WADDR    = 3'd2 ;
parameter WDATA    = 3'd3 ;
parameter RADDR    = 3'd4 ;
parameter RDATA    = 3'd5 ;
parameter DONE     = 3'd6 ;
```

❖ 211 ~ 227 : 상태 전이를 구현 ➔ idle 상태에서 ss_nedge(negative edge)가 발생하면 slave_id 상태로 ➔ slaveid 상태에서는 slave_id 데이터(8bits)을 수신한 후, 값에 따라 waddr (write address), raddr (read address) 상태로 전환 ➔ waddr, raddr 상태에서는 address 데이터 (8bits)을 수신한 후 각각 wdata, rdata 상태로 전환 ➔

➔ wdata 상태에서는 data (8bits)을 수신한 후 done 상태로 전환 ➔ , rdata 상태에서는 miso로 데이터를 전송한 후에 done 상태로 전환 ➔ done 상태에서는 3-clock 후에 idle 상태로 전환



SPI Slave Implementation

➢ spi_slave_exam_0.xpr

➢ SPI Slave Implementation ➔ Verilog Code Implementation ➔ spi_slave . v (8)

spi_slave . v (8)

```
211.// ----------------------------------------
212.// 4) state transition
213.always @(posedge clock or negedge reset)
214.begin
215.            if(!reset)          begin
216.                                s_state <= 3'h0;
217.            end
218.                                begin
219.                                s_state <= (s_idle   & ss_nedge ) ? SLAVEID :
220.                                (s_slaveid & (sid_sckn_cnt==4'd8)) ? ((slave_id==SLAVE_IDW) ? WADDR :
221.                                (slave_id==SLAVE_IDR) ? RADDR : IDLE) :
222.                                (s_waddr   & (wa_sckn_cnt==4'd8) ) ? WDATA :
223.                                (s_wdata   & ss_pedge ) ? DONE  :
224.                                (s_raddr   & (ra_sckn_cnt==4'd8) ) ? RDATA :
225.                                (s_rdata   & ss_pedge ) ? DONE  :
226.                                (s_done    & (done_cnt==2'd3)) ? IDLE  : s_state ;
227.            end
228.end
```

```
// 1) define state
parameter IDLE     = 3'd0 ;
parameter SLAVEID  = 3'd1 ;
parameter WADDR    = 3'd2 ;
parameter WDATA    = 3'd3 ;
parameter RADDR    = 3'd4 ;
parameter RDATA    = 3'd5 ;
parameter DONE     = 3'd6 ;
```

❖ 211 ~ 227 : 상태 전이를 구현 ➔ idle 상태에서 ss_nedge(negative edge)가 발생하면 slave_id 상태로 ➔ slaveid 상태에서는 slave_id 데이터(8bits)을 수신한 후, 값에 따라 waddr (write address), raddr (read address) 상태로 전환 ➔ waddr, raddr 상태에서는 address 데이터 (8bits)을 수신한 후 각각 wdata, rdata 상태로 전환 ➔

➔ wdata 상태에서는 data (8bits)을 수신한 후 done 상태로 전환 ➔ , rdata 상태에서는 miso로 데이터를 전송한 후에 done 상태로 전환 ➔ done 상태에서는 3-clock 후에 idle 상태로 전환

```
229.reg         [7:0]           user_reg1, user_reg2, user_reg3, user_reg4 ;
230.always @(posedge clock or negedge reset)
231.begin
232.            if(!reset)      begin
233.                            user_reg1  <= 8'b0;     user_reg2  <= 8'b0;     user_reg3  <= 8'b0;     user_reg4  <= 8'b0;
234.            end
235.            else            begin
236.                            user_reg1  <= (s_done & (waddr==`rUSER_REG1 )) ? wdata : user_reg1 ;
237.                            user_reg2  <= (s_done & (waddr==`rUSER_REG2 )) ? wdata : user_reg2 ;
238.                            user_reg3  <= (s_done & (waddr==`rUSER_REG3 )) ? wdata : user_reg3 ;
239.                            user_reg4  <= (s_done & (waddr==`rUSER_REG4 )) ? wdata : user_reg4 ;
240.            end
241.end
```

❖ 228 ~ 240 : spi master로 부터 수신한 wdata 값을 Address 에 따라서 사용자 Register 에 저장

```
242.always @(posedge clock or negedge reset)
243.begin
244.            if(!reset)      rdata <= 16'b0;
245.            else            rdata <= s_idle ? 16'b0 :
246.                            (s_raddr & sck_pedge_1d & (ra_sckn_cnt==4'd7) & (raddr == `rUSER_REG1)) ? user_reg1 :
247.                            (s_raddr & sck_pedge_1d & (ra_sckn_cnt==4'd7) & (raddr == `rUSER_REG2)) ? user_reg2 :
248.                            (s_raddr & sck_pedge_1d & (ra_sckn_cnt==4'd7) & (raddr == `rUSER_REG3)) ? user_reg3 :
249.                            (s_raddr & sck_pedge_1d & (ra_sckn_cnt==4'd7) & (raddr == `rUSER_REG4)) ? user_reg4 :
250.                            rdata ;
251.end
252.endmodule
```

❖ 241 ~ 251 : : read address 에 따라 사용자 Register 값 전송 위해 rdata 에 저장
➔ [주의] rdata값은 miso로 데이터를 전송하기 위하여 미리 준비되어야 함

SPI Slave Implementation

- ➔ Spec and Timing 정의
- ➔ Port 정의
- ➔ State 정의
- ➔ Code Implementation
- ➔ State Transition
- ➔ **Test Bench**

SPI Slave Implementation

➢ spi_slave_exam_0.xpr

➢ SPI Slave Implementation ➔ Verilog Code Implementation ➔ **Test Bench** : spi_slave_tb . v (1)

spi_slave_tb . v (1)

```
1.  `timescale 1ns / 1ps
2.  module tb_spi_slave();
3.  reg         reset, clock;
4.  initial     begin
5.              reset = 0;
6.              clock = 0;
7.  #10000      reset = 1;
8.  end

9.  always  #5  clock = ~clock;
```

❖ 2 ~ 9 : reset, clock 생성

Simulation Sequency

(1) 0x10 번지에 0x45를 write

(2) 0x11 번지에 0x89를 write

(3) 0x10 번지 read, 0x45 인지 확인

(4) 0x11 번지 read, 0x89 인지 확인

```
10. reg         [3:0]       cnt1;
11. always @(posedge clock or negedge reset)
12. begin
13.             if(!reset)      cnt1 <= 4'b0;
14.             else            cnt1 <= cnt1 + 1'b1;
15. end

16. reg         [7:0]       cnt2;
17. always @(posedge clock or negedge reset)
18. begin
19.             if(!reset)      cnt2 <= 8'b0;
20.             else            cnt2 <= (cnt1==4'd15) ? cnt2+1'b1 : cnt2;
21. end
```

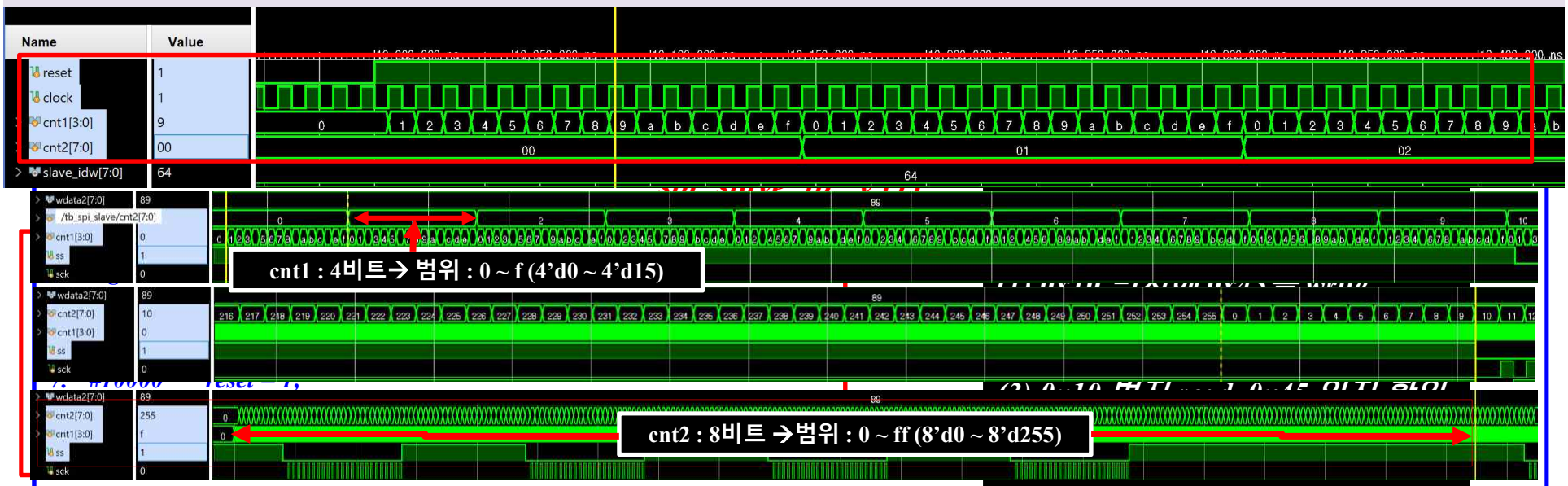
❖ 10 ~ 21 : ss, sck, mosi 신호를 만들기 위해 사용하는 counter ➔ cnt1, cnt2를 생성

➔ cnt1 : 4비트 ➔ 범위 : 0 ~ f (4'd0 ~ 4'd15)

➔ cnt2 : 8비트 ➔범위 : 0 ~ ff (8'd0 ~ 8'd255)

```
22. wire        [7:0]       slave_idw = 8'h64;
23. wire        [7:0]       slave_idr = 8'h65;
24. wire        [7:0]       waddr1    = 8'h10; // 8'h10= 8'b0001_0000
25. wire        [7:0]       raddr1    = 8'h10;
26. wire        [7:0]       wdata1    = 8'h45; // 8'h45= 8'b0100_0101
27. wire        [7:0]       waddr2    = 8'h11; // 8'h11= 8'b0001_0001
28. wire        [7:0]       raddr2    = 8'h11;
29. wire        [7:0]       wdata2    = 8'h89; // 8'h89= 8'b1000_1001
```

❖ 17 ~ 22 : 시퀀스에 사용할 신호 생성



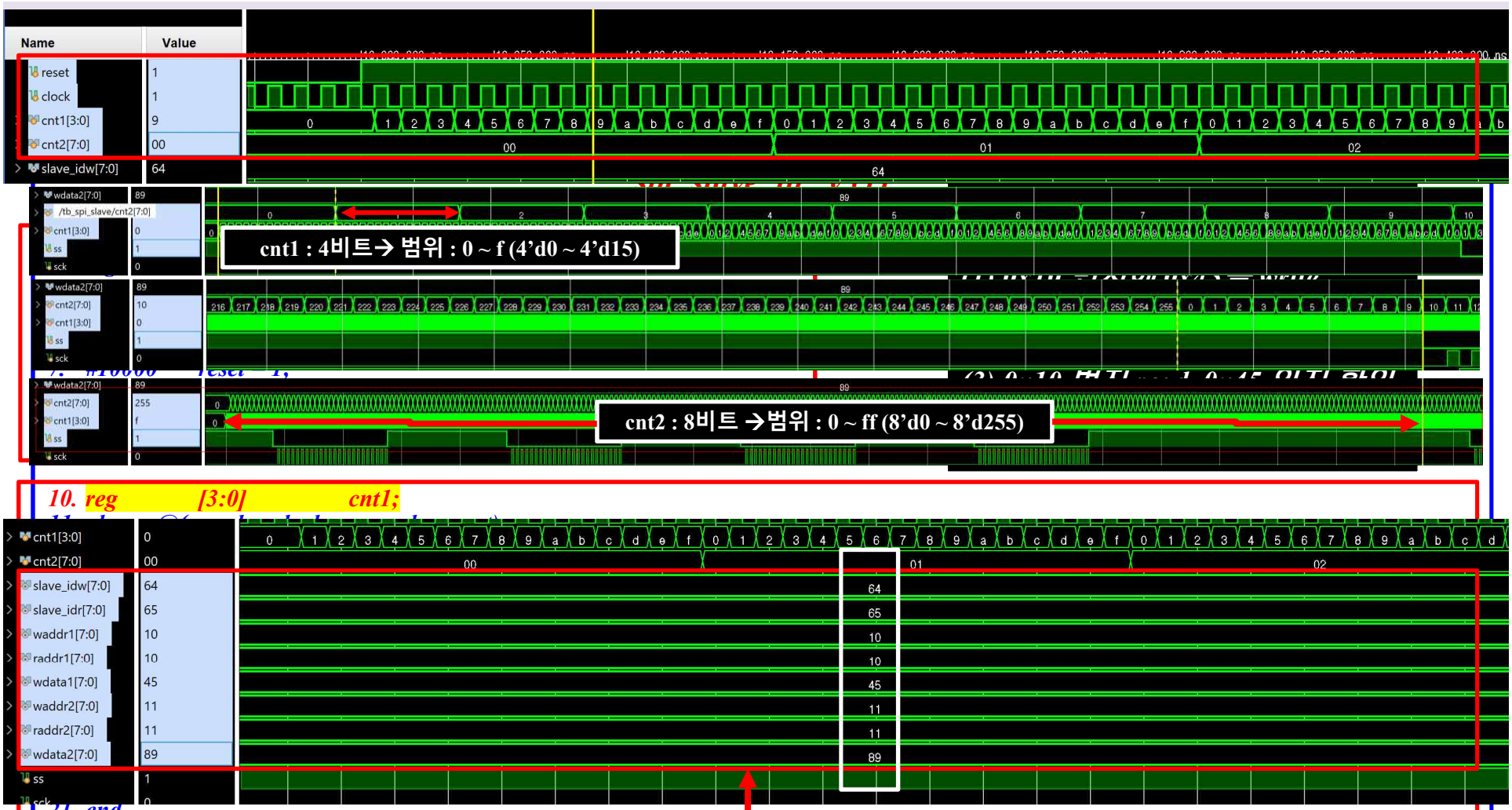

```
10. reg         [3:0]       cnt1;
11. always @(posedge clock or negedge reset)
12. begin
13.             if(!reset)      cnt1 <= 4'b0;
14.             else            cnt1 <= cnt1 + 1'b1;
15. end

16. reg         [7:0]       cnt2;
17. always @(posedge clock or negedge reset)
18. begin
19.             if(!reset)      cnt2 <= 8'b0;
20.             else            cnt2 <= (cnt1==4'd15) ? cnt2+1'b1 : cnt2;
21. end
```

- ❖ 10 ~ 21 : ss, sck, mosi 신호를 만들기 위해 사용하는 counter → cnt1, cnt2를 생성
- → cnt1 : 4비트 → 범위 : 0 ~ f (4'd0 ~ 4'd15)
- → cnt2 : 8비트 →범위 : 0 ~ ff (8'd0 ~ 8'd255)

```
22. wire        [7:0]       slave_idw = 8'h64;
23. wire        [7:0]       slave_idr = 8'h65;
24. wire        [7:0]       waddr1    = 8'h10;
25. wire        [7:0]       raddr1    = 8'h10;
26. wire        [7:0]       wdata1    = 8'h45;
27. wire        [7:0]       waddr2    = 8'h11;
28. wire        [7:0]       raddr2    = 8'h11;
29. wire        [7:0]       wdata2    = 8'h89;
```

- ❖ 17 ~ 22 : 시퀀스에 사용할 신호 생성

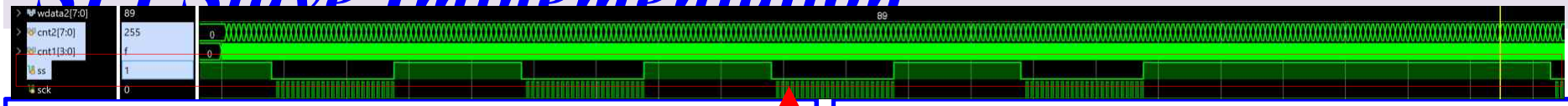



```
10. reg         [3:0]       cnt1;
21. end
22. wire        [7:0]       slave_idw = 8'h64;
23. wire        [7:0]       slave_idr = 8'h65;
24. wire        [7:0]       waddr1    = 8'h10; // 8'h10= 8'b0001_0000
25. wire        [7:0]       raddr1    = 8'h10;
26. wire        [7:0]       wdata1    = 8'h45; // 8'h45= 8'b0100_0101
27. wire        [7:0]       waddr2    = 8'h11; // 8'h11= 8'b0001_0001
28. wire        [7:0]       raddr2    = 8'h11;
29. wire        [7:0]       wdata2    = 8'h89; // 8'h89= 8'b1000_1001
```

❖ 17 ~ 22 : 시퀀스에 사용할 신호 생성

SPI Slave Implementation

➢ *spi_slave_exam_0.xpr*



spi_slave_tb . v (2)

```
30. reg                     ss;
31. always @(posedge clock or negedge reset)
32. begin
33.             if(!reset)      ss <= 1'b1;
34.             else  ss <=     ((cnt2==8'd10 ) && (cnt1==4'd0)) ? 1'b0 :
35.                             ((cnt2==8'd34 ) && (cnt1==4'd8)) ? 1'b1 :
36.                             ((cnt2==8'd60 ) && (cnt1==4'd0)) ? 1'b0 :
37.                             ((cnt2==8'd84 ) && (cnt1==4'd8)) ? 1'b1 :
38.                             ((cnt2==8'd110) && (cnt1==4'd0)) ? 1'b0 :
39.                             ((cnt2==8'd134) && (cnt1==4'd8)) ? 1'b1 :
40.                             ((cnt2==8'd160) && (cnt1==4'd0)) ? 1'b0 :
41.                             ((cnt2==8'd184) && (cnt1==4'd8)) ? 1'b1 : ss ;
42. end
```

❖ 30 ~ 42 : ss 신호 생성

```
43. reg                     sck;
44. always @(posedge clock or negedge reset)
45. begin
46. if(!reset)      sck <= 1'b0;
47. else sck <= (cnt2==8'd11) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
48. (cnt2==8'd12) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
49. (cnt2==8'd13) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
50. (cnt2==8'd14) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
51. (cnt2==8'd15) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
52. (cnt2==8'd16) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
53. (cnt2==8'd17) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
54. (cnt2==8'd18) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
55. (cnt2==8'd19) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
56. (cnt2==8'd20) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
57. (cnt2==8'd21) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
58. (cnt2==8'd22) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
59. (cnt2==8'd23) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
60. (cnt2==8'd24) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
61. (cnt2==8'd25) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
62. (cnt2==8'd26) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
63. (cnt2==8'd27) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
64. (cnt2==8'd28) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
65. (cnt2==8'd29) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
66. (cnt2==8'd30) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
67. (cnt2==8'd31) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
68. (cnt2==8'd32) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
69. (cnt2==8'd33) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
70. (cnt2==8'd34) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
```

❖ 43 ~ 143 : sck 신호 생성

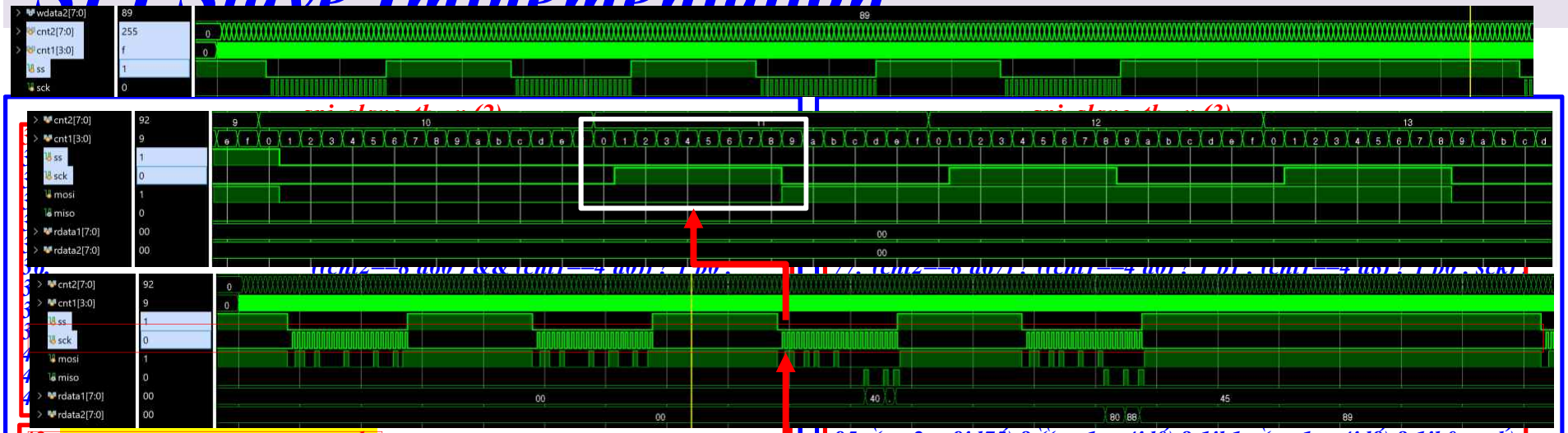
spi_slave_tb . v (3)

```
71. (cnt2==8'd61) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
72. (cnt2==8'd62) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
73. (cnt2==8'd63) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
74. (cnt2==8'd64) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
75. (cnt2==8'd65) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
76. (cnt2==8'd66) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
77. (cnt2==8'd67) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
78. (cnt2==8'd68) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
79. (cnt2==8'd69) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
80. (cnt2==8'd70) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
81. (cnt2==8'd71) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
82. (cnt2==8'd72) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
83. (cnt2==8'd73) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
84. (cnt2==8'd74) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
85. (cnt2==8'd75) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
86. (cnt2==8'd76) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
87. (cnt2==8'd77) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
88. (cnt2==8'd78) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
89. (cnt2==8'd79) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
90. (cnt2==8'd80) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
91. (cnt2==8'd81) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
92. (cnt2==8'd82) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
93. (cnt2==8'd83) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
94. (cnt2==8'd84) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :

95. (cnt2==8'd111) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
96. (cnt2==8'd112) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
97. (cnt2==8'd113) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
98. (cnt2==8'd114) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
99. (cnt2==8'd115) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
100.(cnt2==8'd116) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
101.(cnt2==8'd117) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
102.(cnt2==8'd118) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
103.(cnt2==8'd119) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
104.(cnt2==8'd120) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
105.(cnt2==8'd121) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
106.(cnt2==8'd122) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
107.(cnt2==8'd123) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
108.(cnt2==8'd124) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
109.(cnt2==8'd125) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
110.(cnt2==8'd126) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
111.(cnt2==8'd127) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
112.(cnt2==8'd128) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
```

SPI Slave Implementation

➢ *spi_slave_exam_0.xpr*



❖ 43 ~ 143 : sck 신호 생성

```
43. reg                         sck;
44. always @(posedge clock or negedge reset)
45. begin
46. if(!reset)      sck <= 1'b0;
47. else sck <= (cnt2==8'd11) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 :
    sck) :
48. (cnt2==8'd12) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
49. (cnt2==8'd13) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
50. (cnt2==8'd14) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
51. (cnt2==8'd15) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
52. (cnt2==8'd16) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
53. (cnt2==8'd17) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
54. (cnt2==8'd18) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
55. (cnt2==8'd19) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
56. (cnt2==8'd20) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
57. (cnt2==8'd21) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
58. (cnt2==8'd22) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
59. (cnt2==8'd23) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
60. (cnt2==8'd24) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
61. (cnt2==8'd25) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
62. (cnt2==8'd26) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
63. (cnt2==8'd27) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
64. (cnt2==8'd28) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
65. (cnt2==8'd29) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
66. (cnt2==8'd30) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
67. (cnt2==8'd31) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
68. (cnt2==8'd32) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
69. (cnt2==8'd33) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
70. (cnt2==8'd34) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
```

```
85. (cnt2==8'd75) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
86. (cnt2==8'd76) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
87. (cnt2==8'd77) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
88. (cnt2==8'd78) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
89. (cnt2==8'd79) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
90. (cnt2==8'd80) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
91. (cnt2==8'd81) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
92. (cnt2==8'd82) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
93. (cnt2==8'd83) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
94. (cnt2==8'd84) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
```

```
95. (cnt2==8'd111) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
96. (cnt2==8'd112) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
97. (cnt2==8'd113) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
98. (cnt2==8'd114) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
99. (cnt2==8'd115) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
100.(cnt2==8'd116) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
101.(cnt2==8'd117) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
102.(cnt2==8'd118) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
103.(cnt2==8'd119) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
104.(cnt2==8'd120) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
105.(cnt2==8'd121) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
106.(cnt2==8'd122) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
107.(cnt2==8'd123) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
108.(cnt2==8'd124) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
109.(cnt2==8'd125) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
110.(cnt2==8'd126) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
111.(cnt2==8'd127) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
112.(cnt2==8'd128) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
```

SPI Slave Implementation

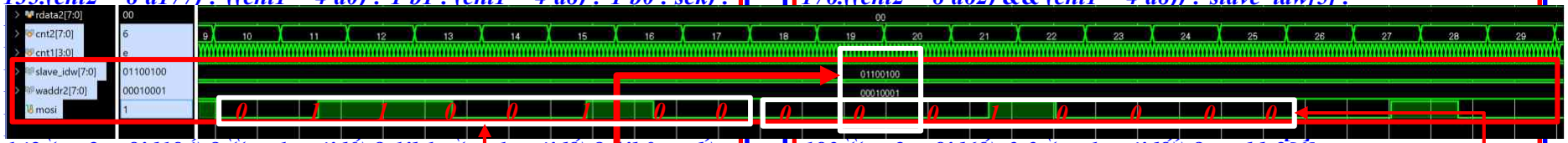
➢ spi_slave_exam_0.xpr

➢ SPI Slave Implementation ➔ Verilog Code Implementation ➔ Test Bench : **spi_slave_tb . v (4)(5)**

spi_slave_tb . v (4)

```
113.(cnt2==8'd129) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
114.(cnt2==8'd130) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
115.(cnt2==8'd131) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
116.(cnt2==8'd132) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
117.(cnt2==8'd133) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
118.(cnt2==8'd134) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :

119.(cnt2==8'd161) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
120.(cnt2==8'd162) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
121.(cnt2==8'd163) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
122.(cnt2==8'd164) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
123.(cnt2==8'd165) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
124.(cnt2==8'd166) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
125.(cnt2==8'd167) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
126.(cnt2==8'd168) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
127.(cnt2==8'd169) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
128.(cnt2==8'd170) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
129.(cnt2==8'd171) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
130.(cnt2==8'd172) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
131.(cnt2==8'd173) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
132.(cnt2==8'd174) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
133.(cnt2==8'd175) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
134.(cnt2==8'd176) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
135.(cnt2==8'd177) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
```




```
142.(cnt2==8'd184) ? ((cnt1==4'd0) ? 1'b1 : (cnt1==4'd8) ? 1'b0 : sck) :
    sck ;
143.end

144.reg                    mosi;
145.always @(posedge clock or negedge reset)
146.begin
147.if(!reset)   mosi <= 1'b1;
148.else  mosi <= ((cnt2==8'd10) && (cnt1==4'd0)) ? slave_idw[7] :
149.              ((cnt2==8'd11) && (cnt1==4'd8)) ? slave_idw[6] :
150.              ((cnt2==8'd12) && (cnt1==4'd8)) ? slave_idw[5] :
151.              ((cnt2==8'd13) && (cnt1==4'd8)) ? slave_idw[4] :
152.              ((cnt2==8'd14) && (cnt1==4'd8)) ? slave_idw[3] :
```

❖ 144 ~ 249 : mosi 신호 생성

spi_slave_tb . v (5)

```
153.((cnt2==8'd15) && (cnt1==4'd8)) ? slave_idw[2] :
154.((cnt2==8'd16) && (cnt1==4'd8)) ? slave_idw[1] :
155.((cnt2==8'd17) && (cnt1==4'd8)) ? slave_idw[0] :
156.((cnt2==8'd18) && (cnt1==4'd8)) ? waddr1[7]    :
157.((cnt2==8'd19) && (cnt1==4'd8)) ? waddr1[6]    :
158.((cnt2==8'd20) && (cnt1==4'd8)) ? waddr1[5]    :
159.((cnt2==8'd21) && (cnt1==4'd8)) ? waddr1[4]    :
160.((cnt2==8'd22) && (cnt1==4'd8)) ? waddr1[3]    :
161.((cnt2==8'd23) && (cnt1==4'd8)) ? waddr1[2]    :
162.((cnt2==8'd24) && (cnt1==4'd8)) ? waddr1[1]    :
163.((cnt2==8'd25) && (cnt1==4'd8)) ? waddr1[0]    :
164.((cnt2==8'd26) && (cnt1==4'd8)) ? wdata1[7]    :
165.((cnt2==8'd27) && (cnt1==4'd8)) ? wdata1[6]    :
166.((cnt2==8'd28) && (cnt1==4'd8)) ? wdata1[5]    :
167.((cnt2==8'd29) && (cnt1==4'd8)) ? wdata1[4]    :
168.((cnt2==8'd30) && (cnt1==4'd8)) ? wdata1[3]    :
169.((cnt2==8'd31) && (cnt1==4'd8)) ? wdata1[2]    :
170.((cnt2==8'd32) && (cnt1==4'd8)) ? wdata1[1]    :
171.((cnt2==8'd33) && (cnt1==4'd8)) ? wdata1[0]    :
172.((cnt2==8'd35) && (cnt1==4'd0)) ? 1'b1         :
173

174.((cnt2==8'd60) && (cnt1==4'd0)) ? slave_idw[7] :
175.((cnt2==8'd61) && (cnt1==4'd8)) ? slave_idw[6] :
176.((cnt2==8'd62) && (cnt1==4'd8)) ? slave_idw[5] :
```


```
183.((cnt2==8'd69) && (cnt1==4'd8)) ? waddr2[6]    :
184.((cnt2==8'd70) && (cnt1==4'd8)) ? waddr2[5]    :
185.((cnt2==8'd71) && (cnt1==4'd8)) ? waddr2[4]    :
186.((cnt2==8'd72) && (cnt1==4'd8)) ? waddr2[3]    :
```

SPI Slave Implementation

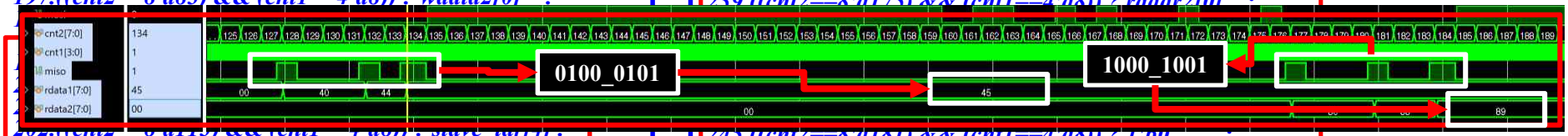
➢ spi_slave_exam_0.xpr

➢ SPI Slave Implementation ➔ Verilog Code Implementation ➔ Test Bench : **spi_slave_tb . v (6)(7)**

spi_slave_tb . v (6)

```
196.((cnt2==8'd82) && (cnt1==4'd8)) ? wdata2[1]    :
197.((cnt2==8'd83) && (cnt1==4'd8)) ? wdata2[0]    :
203.((cnt2==8'd114) && (cnt1==4'd8)) ? slave_idr[3] :
204.((cnt2==8'd115) && (cnt1==4'd8)) ? slave_idr[2] :
205.((cnt2==8'd116) && (cnt1==4'd8)) ? slave_idr[1] :
206.((cnt2==8'd117) && (cnt1==4'd8)) ? slave_idr[0] :
207.((cnt2==8'd118) && (cnt1==4'd8)) ? raddr1[7]    :
208.((cnt2==8'd119) && (cnt1==4'd8)) ? raddr1[6]    :
209.((cnt2==8'd120) && (cnt1==4'd8)) ? raddr1[5]    :
210.((cnt2==8'd121) && (cnt1==4'd8)) ? raddr1[4]    :
211.((cnt2==8'd122) && (cnt1==4'd8)) ? raddr1[3]    :
212.((cnt2==8'd123) && (cnt1==4'd8)) ? raddr1[2]    :
213.((cnt2==8'd124) && (cnt1==4'd8)) ? raddr1[1]    :
214.((cnt2==8'd125) && (cnt1==4'd8)) ? raddr1[0]    :
215.((cnt2==8'd126) && (cnt1==4'd8)) ? 1'b0         :
216.((cnt2==8'd127) && (cnt1==4'd8)) ? 1'b0         :
217.((cnt2==8'd128) && (cnt1==4'd8)) ? 1'b0         :
218.((cnt2==8'd129) && (cnt1==4'd8)) ? 1'b0         :
219.((cnt2==8'd130) && (cnt1==4'd8)) ? 1'b0         :
220.((cnt2==8'd131) && (cnt1==4'd8)) ? 1'b0         :
221.((cnt2==8'd132) && (cnt1==4'd8)) ? 1'b0         :
222.((cnt2==8'd133) && (cnt1==4'd8)) ? 1'b0         :
223.((cnt2==8'd135) && (cnt1==4'd0)) ? 1'b1         :

224.((cnt2==8'd160) && (cnt1==4'd0)) ? slave_idr[7] :
225.((cnt2==8'd161) && (cnt1==4'd8)) ? slave_idr[6] :
226.((cnt2==8'd162) && (cnt1==4'd8)) ? slave_idr[5] :
227.((cnt2==8'd163) && (cnt1==4'd8)) ? slave_idr[4] :
228.((cnt2==8'd164) && (cnt1==4'd8)) ? slave_idr[3] :
229.((cnt2==8'd165) && (cnt1==4'd8)) ? slave_idr[2] :
230.((cnt2==8'd166) && (cnt1==4'd8)) ? slave_idr[1] :
231.((cnt2==8'd167) && (cnt1==4'd8)) ? slave_idr[0] :
232.((cnt2==8'd168) && (cnt1==4'd8)) ? raddr2[7]    :
233.((cnt2==8'd169) && (cnt1==4'd8)) ? raddr2[6]    :
234.((cnt2==8'd170) && (cnt1==4'd8)) ? raddr2[5]    :
235.((cnt2==8'd171) && (cnt1==4'd8)) ? raddr2[4]    :
236.((cnt2==8'd172) && (cnt1==4'd8)) ? raddr2[3]    :
```



spi_slave_tb . v (7)

```
237.((cnt2==8'd173) && (cnt1==4'd8)) ? raddr2[2]    :
238.((cnt2==8'd174) && (cnt1==4'd8)) ? raddr2[1]    :
239.((cnt2==8'd175) && (cnt1==4'd8)) ? raddr2[0]    :
245.((cnt2==8'd181) && (cnt1==4'd8)) ? 1'b0         :
246.((cnt2==8'd182) && (cnt1==4'd8)) ? 1'b0         :
247.((cnt2==8'd183) && (cnt1==4'd8)) ? 1'b0         :
248.((cnt2==8'd185) && (cnt1==4'd0)) ? 1'b1         : mosi ;
249.end

250.wire                miso;
251.reg      [7:0]      rdata1 ;
252.always @(posedge clock or negedge reset)
253.begin
254.if(!reset)    rdata1 <= 8'b0;
255.else          begin
256.              rdata1[7] <= ((cnt2==8'd127) && (cnt1==4'd0)) ? miso : rdata1[7];
257.              rdata1[6] <= ((cnt2==8'd128) && (cnt1==4'd0)) ? miso : rdata1[6];
258.              rdata1[5] <= ((cnt2==8'd129) && (cnt1==4'd0)) ? miso : rdata1[5];
259.              rdata1[4] <= ((cnt2==8'd130) && (cnt1==4'd0)) ? miso : rdata1[4];
260.              rdata1[3] <= ((cnt2==8'd131) && (cnt1==4'd0)) ? miso : rdata1[3];
261.              rdata1[2] <= ((cnt2==8'd132) && (cnt1==4'd0)) ? miso : rdata1[2];
262.              rdata1[1] <= ((cnt2==8'd133) && (cnt1==4'd0)) ? miso : rdata1[1];
263.              rdata1[0] <= ((cnt2==8'd134) && (cnt1==4'd0)) ? miso : rdata1[0];
264.              end
265.end

266.reg      [7:0]      rdata2 ;
267.always @(posedge clock or negedge reset)
268.begin
269.if(!reset)    rdata2 <= 8'b0;
270.else          begin
271.              rdata2[7] <= ((cnt2==8'd177) && (cnt1==4'd0)) ? miso : rdata2[7];
272.              rdata2[6] <= ((cnt2==8'd178) && (cnt1==4'd0)) ? miso : rdata2[6];
273.              rdata2[5] <= ((cnt2==8'd179) && (cnt1==4'd0)) ? miso : rdata2[5];
274.              rdata2[4] <= ((cnt2==8'd180) && (cnt1==4'd0)) ? miso : rdata2[4];
275.              rdata2[3] <= ((cnt2==8'd181) && (cnt1==4'd0)) ? miso : rdata2[3];
276.              rdata2[2] <= ((cnt2==8'd182) && (cnt1==4'd0)) ? miso : rdata2[2];
277.              rdata2[1] <= ((cnt2==8'd183) && (cnt1==4'd0)) ? miso : rdata2[1];
278.              rdata2[0] <= ((cnt2==8'd184) && (cnt1==4'd0)) ? miso : rdata2[0];
279.              end
280.end
```

❖ 250 ~ 265 : 0x10 번지 read 한 값을 rdata1 에 저장 ➔ 이 값이 0x45 되어야 함

❖ 266 ~ 280 : 0x11 번지 read 한 값을 rdata2 에 저장 ➔ 이 값이 0x89 되어야 함

SPI Slave Implementation

➢ *spi_slave_exam_0.xpr*

➢ *SPI Slave Implementation* ➔ *Verilog Code Implementation* ➔ *Test Bench :* ***spi_slave_tb . v (2)(3)***

spi_slave_tb . v (3)

```
281.        spi_slave   spi_slave(
282.        .reset (reset),
283.        .clock (clock),
284.        .ss (ss ),
285.        .sck (sck),
286.        .mosi (mosi),
287.        .miso (miso)
288.);
289.

290.endmodule
```

❖ 281 ~ 288 : spi_slave module

SPI Slave Implementation

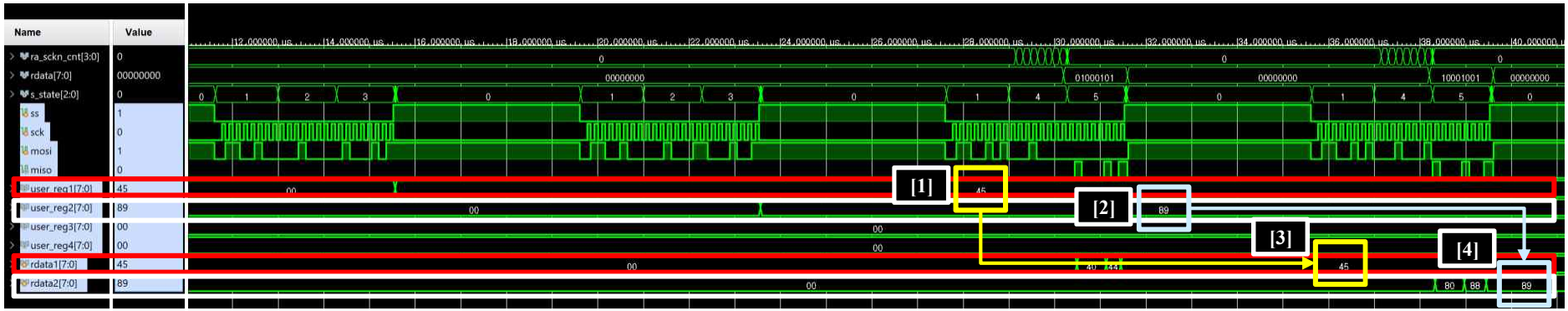
- ➔ Spec and Timing 정의
- ➔ Port 정의
- ➔ State 정의
- ➔ Code Implementation
- ➔ State Transition
- ➔ Test Bench
- ➔ **Simulation Result**

SPI Slave Implementation

➢ *spi_slave_exam_0.xpr*

➢ *SPI Slave Implementation* ➔ *Verilog Code Implementation* ➔ *Test Bench :* ***spi_slave_tb . v (8)***

- ***Simulation Result Check (1)***

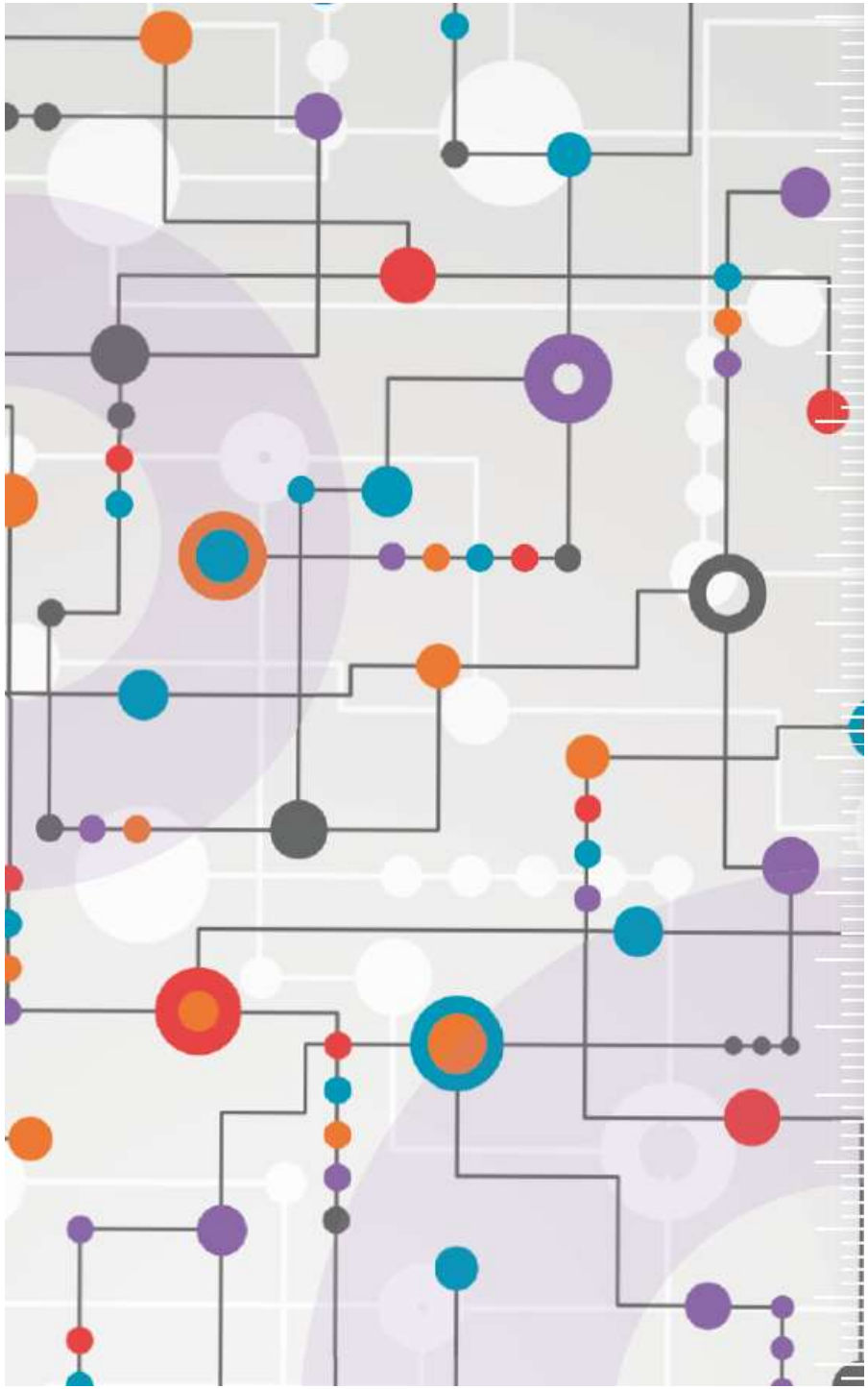


✓ [1] 첫번째 0x10 번지에 0x45 를 write ➔ 0x10 번지는 user_reg1 address 이고, user_reg1 값이 write 후에 0x45 로 변경

✓ [2] 두번째 0x11 번지에 0x89 를 write ➔ 0x11 번지는 user_reg2 address 이고, user_reg2 값이 write 후에 0x89 로 변경

✓ [3] 세번째 0x10 번지 [user_reg1]의 값을 read ➔ read 후에 rdata1 값이 0x45로 변경

✓ [4] 네번째 0x11 번지 [user_reg2]의 값을 read ➔ read 후에 rdata2 값이 0x89로 변경



수고하셨습니다.