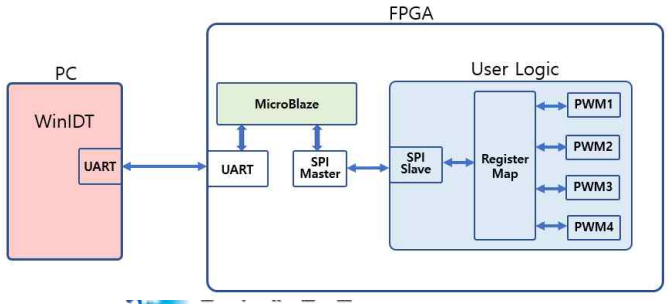
SoC 를 위한 *Peripheral* 설계

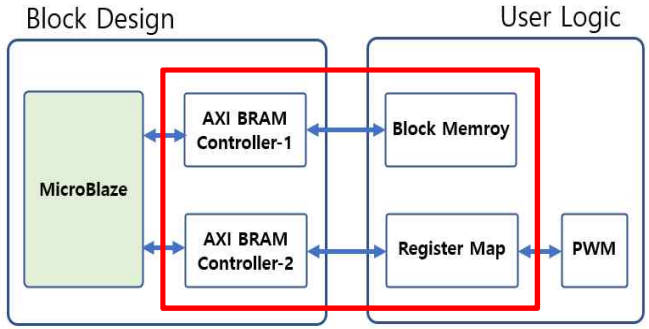
[*Serial Bus* ➔ *SPI* 구현]

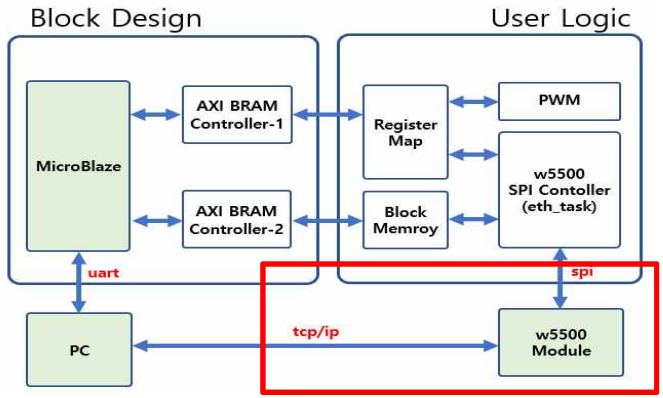
[*Reference*]

- *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 Controller Implementation

➔ [버튼 노이즈 제거 기능 구현]

➔ [*SPI Task* 구현]

SPI Controller Implementation

➢ *spi_controller_exam_0.xpr*

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

❖ ***SPI Master*와 *SPI Slave* 간의 통신을 구현**

- ***spi master* 에서 *slave*의 *User register* *(0x10 ~ 0x13, user_reg1 ~ user_reg4)*에 데이터를 *write* 하고 *read* 하는 것을 구현**

❖ ***Implementation Sequence***

- **구현 목표 : *BTN0*를 누를 때마다 *SPI Master* 에서 *0x10* 번지에 *0x00* 부터 *0xff* 까지 데이터를 *Write* 하고, *Write* 한 값을 읽어서 그 값을 *LED*에 표시**

➔ ***BTN0*를 한번 누르면, *Spi Master* 에서 *Spi Slave* 로 *0x10* 번지에 *0x00*을 *Write*** 하고, ***0x10* 번지를 *Read* 해서 그 값을 *LED*에 표시**

➔ ***BTN0*를 두번째 누르면, 0x10 번지에 0x01을 Write** 하고, **0x10 번지를 읽어서 그 값을 LED에 표시**

➔ **버튼을 누를 때마다 *write* 값을 증가 ➔ *read* 후에 그 값을 LED 에 표시**

➔ 정상적으로 동작한 다면 **버튼을 누를 때마다 0x00 ~ 0xff 까지 순차적으로 LED 에 표시**

① ***BTN1*을 누르면, *0x11* 번지에 *Write* 하고 Read 해서 그 값을 LED에 표시**

② ***BTN2*을 누르면, *0x12* 번지에 *Write* 하고 Read 해서 그 값을 LED에 표시**

③ ***BTN3*을 누르면, *0x13* 번지에 *Write* 하고 Read 해서 그 값을 LED에 표시**

④ ***LED*에 표시되는 값은, [*LD7 : bit7], [LD6 : bit6], [LD5 : bit5], [LD4 : bit4], [LD3 : bit3], [LD2 : bit2], [LD1 : bit1], [LD0 : bit0]***

SPI Controller Implementation

➔ **[버튼 노이즈 제거 기능 구현]**

➔ **[*SPI Task* 구현]**

SPI Controller Implementation

➢ *spi_controller_exam_0.xpr*

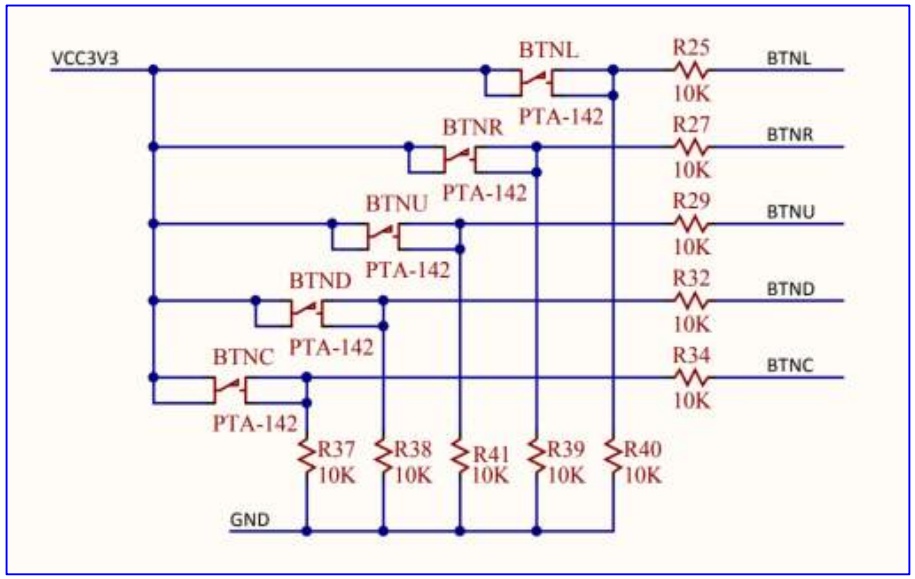
➢ *SPI (Serial Peripheral Interface Bus) Controller Implementation*

❖ **버튼 노이즈(*Button Noise*) 제거**

- Basys3 보드에 장착된 ***Button(btnC, btnU, btnL, btnR, btnD)*** 은 ***Tact Switch***
- ***Tact Switch*** 는 버튼을 누를 때 짧은 순간에 여러 번 접촉했다 떨어졌다를 반복하는 노이즈 현상
  ➔ 버튼을 한번만 눌러도 마치 여러 번 버튼을 누른 것과 같은 현상(오류)이 발생
- ***Button Noise*** 오류를 방지하는 것을 코드로 구현

❖ 버튼 회로

- ***Basys3*** 보드에 장착된 ***Button(btnC, btnU, btnL, btnR, btnD)*** 은 Tact Switch의 회로도
- [버튼 눌림 : ***High(1)***] ➔➔ [버튼 누르지 않으면 : ***Low(0)***]



SPI Controller Implementation

➔[버튼 노이즈 제거 기능 구현]

1. Code Implementation

➔[*SPI Task* 구현]

SPI Controller Implementation

➢ *spi_controller_exam_0.xpr*

➢ *SPI slave Implementation* ➔ Code Implementation ➔ 버튼의 입력을 10ms 간격으로 Check 해서 이전 값과 현재 값을 이용해서 버튼이 눌러진 순간을 검출

btn_in(1)

```
1.  `timescale 1ns / 1ps

2.  module btn_in(
3.              reset , clock , btn_in , btn_out
4.  );
5.  input       reset , clock , btn_in  ;
6.  output      btn_out ;

7.  parameter max_cnt = 20'd1_000_000;  // 10ms
8.  // parameter  max_cnt = 20'd1000;       // for simlation, 1000 : 10us, 100: 1us, 10 : 0.1us

9.  reg         [19:0]          cnt;
10. always @(posedge clock or negedge reset)
11. begin
12.             if(~reset)      cnt <= 20'b0;
13.             else            cnt <= (cnt==max_cnt) ? 20'b0 : cnt+1'b1;
14. end

15. reg                         btn_1d, btn_2d;
16. always @(posedge clock or negedge reset)
17. begin
18.             if(~reset)      begin
19.                             btn_1d <= 1'b0;
20.                             btn_2d <= 1'b0;
21.             end
22.             else            begin
23.                             btn_1d <= btn_in;
24.                             btn_2d <= btn_1d;
25.             end
26. end
```

❖ 2 ~ 6 : Port 정의

➔ btn 입력값 받아서, 노이즈를 제거하고 버튼이 눌러진 순간에 1-clock 펄스 출력

❖ 7 ~ 8 : 10ms 을 위한 카운터 최대 값[100Mhz clock, 100만번(=< 2^{20}) ➔ 10ms(100Hz, 1/100초)]

➔ *simulation* 에서 사용하는 값: 1000(=10us) ➔ 10ms 는 Simulation에서 매우 긴 시간

❖ 9 ~ 14 : 10ms 카운터를 생성

❖ 15 ~ 26 : 버튼 입력이 외부에서 들어오기 때문에, 안정적인 사용을 위하여 2번 delay 후 사용

SPI Controller Implementation

➢ *spi_controller_exam_0.xpr*

➢ *SPI Controller Implementation* ➔ Code Implementation ➔ 버튼의 입력을 10ms 간격으로 Check 해서 이전 값과 현재 값을 이용해서 버튼이 눌러진 순간을 검출

btn_in(2)

```
27. reg                 btn1, btn2;
28. always @(posedge clock or negedge reset)
29. begin
30.             if(~reset)      begin
31.                             btn1 <= 1'b0;
32.                             btn2 <= 1'b0;
33.             end
34.             else            begin
35.                             btn1 <= (cnt==max_cnt) ? btn_2d : btn1;
36.                             btn2 <= (cnt==max_cnt) ? btn1   : btn2;
37.             end
38. end
```

❖ 27 ~ 38 : 10ms 마다 버튼 입력 값을 검출해서 btn1(현재 버튼 값), btn2(10ms 이전에 검출된 값)에 저장

```
39. reg                 btn_out;
40. always @(posedge clock or negedge reset)
41. begin
42.             if(~reset)      btn_out <= 1'b0;
43.             else            btn_out <= ((cnt==max_cnt) & btn1 &~btn2) ? 1'b1 : 1'b0;
44. end

45. endmodule
```

❖ 39 ~ 45 : 10ms 마다 버튼이 눌러진 상태를 검출해서 btn_out 으로 출력을 생성

// btn1 ➔ btn_in 의 2 clock delay
// btn2 ➔ btn1 의 1 clock delay

SPI Controller Implementation

➔ **[버튼 노이즈 제거 기능 구현]**

1. *Code Implementation*
2. ***Test Bench***

➔ **[*SPI Task* 구현]**

SPI Controller Implementation

➢ spi_controller_exam_0.xpr

➢ SPI Controller Implementation ➔ Test Bench : **spi_controller_exam_tb . v** ➔ **Simulation Reault Check**

btn_in_tb.v

```
1.  `timescale 1ns / 1ps

2.  module btn_in_tb();

3.  reg         reset, clock;

4.  initial     begin
5.              reset = 0;
6.              clock = 0;
7.  #10000      reset = 1;
8.  end
9.
10. always  #5  clock = ~clock;   // 100 Mhz(10ns)

11. reg         [15:0]          cnt;
12. always @(posedge clock or negedge reset)
13. begin
14.             if(~reset)      cnt <= 16'b0;
15.             else            cnt <= cnt+1'b1;
16. end

17. reg                         btn;
18. always @(posedge clock or negedge reset)
19. begin
20. if(~reset) btn <= 1'b0;
21. else        btn <= (cnt==16'd100 ) ? 1'b1 : (cnt==16'd1200) ? 1'b0 : // 1100
22.             (cnt==16'd1300) ? 1'b1 : (cnt==16'd1350) ? 1'b0 : // 50
23.             (cnt==16'd1400) ? 1'b1 : (cnt==16'd1480) ? 1'b0 : // 80

24.             (cnt==16'd4200) ? 1'b1 : (cnt==16'd5400) ? 1'b0 : // 1200
25.             (cnt==16'd5600) ? 1'b1 : (cnt==16'd5700) ? 1'b0 : // 100
26.              (cnt==16'd5800) ? 1'b1 : (cnt==16'd5900) ? 1'b0 : // 100
27.             btn ;
28. end
```

```
29. wire        btn_out ;
30. btn_in      btn_in(
31.             .reset   (reset),
32.             .clock   (clock),
33.             .btn_in  (btn ),
34.             .btn_out (btn_out )
35. );
36.
37. endmodule
```

❖ 17 ~ 28 : btn 을 생성

➔ *btn* 입력값을 *10us (1000 clock)* 마다 *check* 하기 때문에, 버튼으로 인식하기 위해서는*1000 clock* 이상의 입력이 발생해야 함

❖ 버튼을 2번 누른 것으로 가정

➔ 첫번째는 *1100 clock* 이고 이어서 2번의 *glitch (50-clock, 80-clock)*이 발생

➔ 두번째는 *1200 clock* 이고 이어서 2번의 *glitch (100-clock, 100-clock)*이 발생

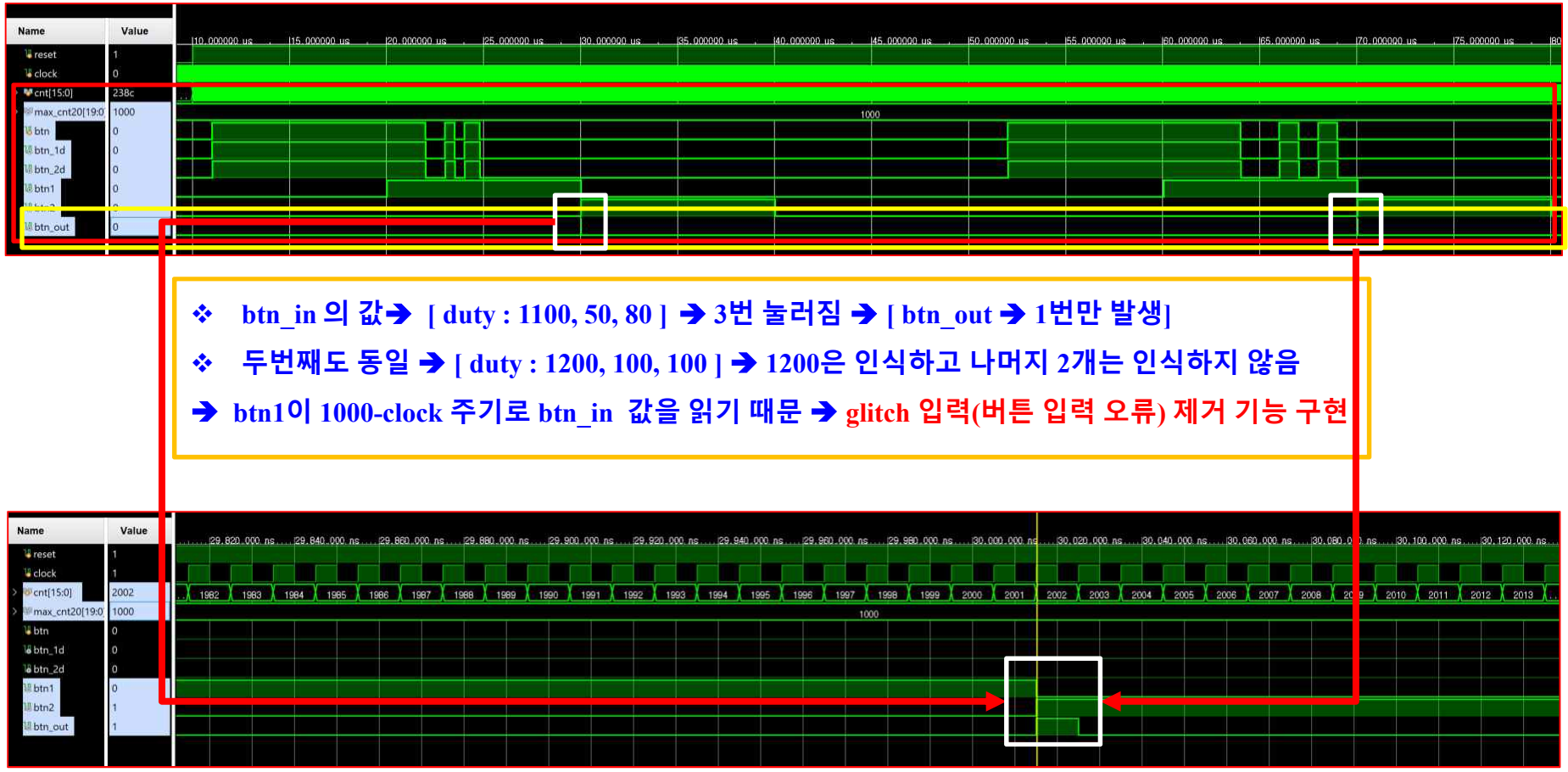
➔ 정상적으로 동작시, *btn_out*은 2번의 *pulse*가 발생

SPI Controller Implementation

➢ *spi_controller_exam_0.xpr*

➢ *SPI Controller Implementation* ➔ *Code Implementation* ➔ *Test Bench :* ***spi_controller_exam_tb . v ()***

- ***Simulation Reault Check ()***



❖ btn_in 의 값➔ [duty : 1100, 50, 80] ➔ 3번 눌러짐 ➔ [btn_out ➔ 1번만 발생]

❖ 두번째도 동일 ➔ [duty : 1200, 100, 100] ➔ 1200은 인식하고 나머지 2개는 인식하지 않음

➔ btn1이 1000-clock 주기로 btn_in 값을 읽기 때문 ➔ glitch 입력(버튼 입력 오류) 제거 기능 구현

SPI Controller Implementation

➔ [버튼 노이즈 제거 기능 구현]

➔ [*SPI Task* 구현]

SPI Controller Implementation

➔ [버튼 노이즈 제거 기능 구현]

➔ [*SPI Task* 구현]

➔ *Port* 정의 & *Code Implementation*

SPI Controller Implementation

➢ *spi_slave_exam_0.xpr*

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

signal	in/out	size	description
reset	***input***	[0]	*main reset, active low*
clock	***input***	[1]	*main clock, 100Mhz*
btn	***input***	[3:0]	*btn* 입력
led	***output***	[7:0]	*led* 출력

- *btn* : 4개의 버튼 입력
- *led* : 결과를 led에 출력

SPI Controller Implementation

➢ *spi_controller_exam_0.xpr*

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

spi_task.v(1)

```
1.  `timescale 1ns / 1ps

2.  module spi_task(
3.              reset , clock , btn  , led
4.  );
5.  input       reset , clock  ;
6.  input       [3:0]          btn    ;
7.  output      [7:0]          led    ;

8.  wire        [3:0]          btn2 ;
9.  btn_in      btn_in_u1 (
10.             .reset  (reset  ), .clock   (clock  ), .btn_in  (btn[0] ), .btn_out  (btn2[0] )
11. );
12. btn_in      btn_in_u2 (
13.             .reset  (reset  ), .clock   (clock  ), .btn_in  (btn[1] ), .btn_out  (btn2[1] )
14. );
15. btn_in      btn_in_u3 (
16.             .reset  (reset  ), .clock   (clock  ), .btn_in  (btn[2] ), .btn_out  (btn2[2] )
17. );
18. btn_in      btn_in_u4 (
19.             .reset  (reset  ), .clock   (clock  ), .btn_in  (btn[3] ), .btn_out  (btn2[3] )
20. );
21. reg         [1:0]          btn_id; // button 4개
22. always @(posedge clock or negedge reset)
23. begin
24.             if(~reset)     btn_id <= 2'b0;
25.             else           btn_id <=   btn2[0] ? 2'b00 :
26.                                        btn2[1] ? 2'b01 :
27.                                        btn2[2] ? 2'b10 :
28.                                        btn2[3] ? 2'b11 : btn_id;
29. end
```

❖ 2 ~ 7 : Port 정의

❖ 8 ~ 20 : btn_in 모듈을 이용하여 노이즈가 제거된 btn2 값을 생성

❖ 21 ~ 29 : 4개의 버튼 중에서 눌러진 버튼의 index

SPI Controller Implementation

➢ spi_controller_exam_0.xpr

➢ SPI Task Implementation ➔ Code Implementation ➔ wdata0 ~ wdata3: spi master ➔ spi slave 로 write 하는 data 값

spi_task.v()

```
30. reg         [7:0]       wdata0;
31. always @(posedge clock or negedge reset)
32. begin
33.             if(~reset)      wdata0 <= 8'h00;
34.             else            wdata0 <= btn2[0] ? wdata0+1'b1 : wdata0 ;
35. end
```

❖ 30 ~ 35 : btn2[0] 가 눌러졌을 때, wdata0 값을 1씩 증가

➔ wdata0 ~ wdata3 은 spi master 에서 spi slave 로 write 하는 data 값

```
36. reg         [7:0]       wdata1;
37. always @(posedge clock or negedge reset)
38. begin
39.             if(~reset)      wdata1 <= 8'h00;
40.             else            wdata1 <= btn2[1] ? wdata1+1'b1 : wdata1 ;
41. end
```

❖ 36 ~ 41 : btn2[1]이 눌러졌을 때, wdata1 값을 1씩 증가

```
42. reg         [7:0]       wdata2;
43. always @(posedge clock or negedge reset)
44. begin
45.             if(~reset)      wdata2 <= 8'h00;
46.             else            wdata2 <= btn2[2] ? wdata2+1'b1 : wdata2 ;
47. end
```

❖ 42 ~ 47 : btn2[2] 가 눌러졌을 때, wdata2 값을 1씩 증가

```
48. reg         [7:0]       wdata3;
49. always @(posedge clock or negedge reset)
50. begin
51.             if(~reset)      wdata3 <= 8'h00;
52.             else            wdata3 <= btn2[3] ? wdata3+1'b1 : wdata3 ;
53. end
```

❖ 48 ~ 53 : btn2[3]이 눌러졌을 때, wdata3 값을 1 씩 증가

SPI Controller Implementation

➔ [버튼 노이즈 제거 기능 구현]

➔ [*SPI Task* 구현]

➔ *Port* 정의 & *Code Implementation*

➔ *State* 정의

SPI Controller Implementation

➢ *spi_controller_exam_0.xpr*

➢ *SPI slave Implementation* ➔ Code Implementation ➔State 정의 ➔ **SM에서 사용할 State 정의**

➔ ***SM***은 ***IDLE, WREADY, WRITE, RREADY, READ*** 로 구성

state	*transition*	*description*
IDLE	버튼이 입력되면 WREADY 상태로 전환	idle state
WREADY	3-clock 후에 WRITE 상태로 전환	write 전에 *addr, data, start_w* 신호 생성
WRITE	write 가 완료되면 RREADY 상태로 전환	*spi write*
RREADY	3-clock 후에 READ 상태로 전환	read 전에 *addr, data, start_r* 신호 생성
READ	read 가 완료되면 IDLE 상태로 전환	*spi read*

❖ ***IDLE*** : 초기 상태 ➔ 버튼을 누르면 ***IDLE*** 상태에서 ***WREADY*** 상태로 천이

❖ ***WREADY*** : ***spi master*** 에서 ***spi slave***로 ***spi write*** 동작 ➔ ***spi master***의 입력을 생성 : ***addr, data***

- *start_w* 신호를 생성 ➔ spi master는 *spi slave*에 데이터를 전송(*spi write*) : *addr, data, start_w* 신호 생성후, *WRITE* 상태로 천이

❖ ***WRITE*** : *spi master*에서 *spi* 통신이 완료될 때까지 기다림 ➔ *spi* 통신(*spi write*)이 완료되면, *spi read* 동작하기 위해서 *RREADY* 상태로 전환

❖ ***RREADY*** : *spi master*에서 *spi slave* 로부터 데이터를 읽기 위한 *spi read*를 구현 ➔ *addr, data, start_r* 신호 생성한 후 *READ* 상태로 전환

❖ ***READ*** : *spi read*가 완료될 때까지 기다렸다가, 완료되면 *IDLE* 상태로 전환

❖ 버튼을 한번씩 누를 때마다, ***IDLE ➔ WREADY ➔ WRITE ➔ RREADY ➔ READ ➔ IDLE*** 상태 전환

SPI Controller Implementation

➢ *spi_controller_exam_0.xpr*

➢ *SPI Task Implementation* ➔ *Code Implementation* ➔ *parameter, SM, State flag* 정의

spi_task.v()

```
54. // ----------------------------------------
55. parameter           SLAVE_IDW = 8'h64;
56. parameter           SLAVE_IDR = 8'h65;
57. parameter           REG_ADDR1 = 8'h10;
58. parameter           REG_ADDR2 = 8'h11;
59. parameter           REG_ADDR3 = 8'h12;
60. parameter           REG_ADDR4 = 8'h13;
61. parameter           SPI_FREQ  = 10'd16;
```

❖ 54 ~ 61 : 내부에서 사용하는 parameter를 정의

```
62. // ----------------------------------------
63. // 1) define state
64. reg       [2:0]     m_state ;
65. parameter           M_IDLE   = 3'd0 ;
66. parameter           M_WREADY = 3'd1 ;
67. parameter           M_WRITE  = 3'd2 ;
68. parameter           M_RREADY = 3'd3 ;
69. parameter           M_READ   = 3'd4 ;
```

state	transition	description
IDLE	버튼이 입력되면 WREADY 상태로 전환	idle state
WREADY	3-clock 후에 WRITE 상태로 전환	write 전에 addr, data, start_w 신호 생성
WRITE	write 가 완료되면 RREADY 상태로 전환	spi write
RREADY	3-clock 후에 READ 상태로 전환	read 전에 addr, data, start_r 신호 생성
READ	read 가 완료되면 IDLE 상태로 전환	spi read

❖ 62 ~ 69 : SM 정의

```
70. // ----------------------------------------
71. // 2) state flag
72. wire        s_idle   = (m_state == M_IDLE  ) ? 1'b1 : 1'b0 ;
73. wire        s_wready = (m_state == M_WREADY) ? 1'b1 : 1'b0 ;
74. wire        s_write  = (m_state == M_WRITE ) ? 1'b1 : 1'b0 ;
75. wire        s_rready = (m_state == M_RREADY) ? 1'b1 : 1'b0 ;
76. wire        s_read   = (m_state == M_READ  ) ? 1'b1 : 1'b0 ;
```

❖ 70 ~ 76 : state flag

SPI Controller Implementation

➔ [버튼 노이즈 제거 기능 구현]

➔ [*SPI Task* 구현]

➔ *Port* 정의 & *Code Implementation*

➔ *State* 정의

➔ *Code Implementation*

SPI Controller Implementation

➢ *spi_controller_exam_0.xpr*

➢ *SPI Task Implementation* ➔ *Code Implementation*

spi_task.v()

```
77. // 3) code implementation
78. reg         [1:0]       wready_cnt;
79. always @(posedge clock or negedge reset)
80. begin
81.             if(~reset)      wready_cnt <= 2'b0;
82.             else            wready_cnt <= ~s_wready ? 2'b0 : wready_cnt+1'b1;
83. end
```

❖ 77 ~ 83 : wready state에서 사용하기 위한 counter를 생성

state	transition	description
WREADY	3-clock 후에 WRITE 상태로 전환	write 전에 addr, data, start_w 신호 생성

```
84. wire        [7:0]       reg_addr = (btn_id==2'd0) ? REG_ADDR1 :
85.                         (btn_id==2'd1) ? REG_ADDR2 :
86.                         (btn_id==2'd2) ? REG_ADDR3 : REG_ADDR4 ;
87. reg         [7:0]       addr;
88. always @(posedge clock or negedge reset)
89. begin
90.             if(~reset)      addr <= 8'b0;
91.             else            addr <= (s_wready & (wready_cnt==2'b11)) ? reg_addr : addr ;
92. end
```

```
93. wire        [7:0]       reg_wdata = (btn_id==2'd0) ? wdata0 :
94.                         (btn_id==2'd1) ? wdata1 :
95.                         (btn_id==2'd2) ? wdata2 : wdata3 ;
96. reg         [7:0]       wdata;
97. always @(posedge clock or negedge reset)
98. begin
99.             if(~reset)      wdata <= 8'b0;
100.            else            wdata <= (s_wready & (wready_cnt==2'b11)) ? reg_wdata : wdata ;
101.end
```

```
102.
103.reg                      start_w;
104.always @(posedge clock or negedge reset)
105.begin
106.            if(~reset)      start_w <= 1'b0;
107.            else            start_w <= (s_wready & (wready_cnt==2'b11)) ? 1'b1 : 1'b0 ;
108.end
```

❖ 84 ~ 108 : spi write 를 위한 addr, wdata, start_w 를 생성

➔ 눌려진 버튼에 따라서 addr, wdata 값을 설정

➔ addr, wdata, start_w 는 wready_cnt 값이 3 일 때 생성

➔ addr, waddr 는 자신의 값을 유지하고 있고, start_w 값은 1-clock pulse 로 생성

SPI Controller Implementation

➢ spi_controller_exam_0.xpr

➢ SPI Task Implementation ➔ Code Implementation

spi_task.v()

```
109.reg         [3:0]       write_cnt;
110.always @(posedge clock or negedge reset)
111.begin
112.            if(~reset)  write_cnt <= 4'b0;
113.            else        write_cnt <= ~s_write ? 4'b0 :
114.                        (write_cnt==4'd15) ? 4'd15 : write_cnt+1'b1;
115.end

116.reg         [1:0]       rready_cnt;
117.always @(posedge clock or negedge reset)
118.begin
119.            if(~reset)  rready_cnt <= 2'b0;
120.            else        rready_cnt <= ~s_rready ? 2'b0 : rready_cnt+1'b1;
121.end

122.reg                     start_r;
123.always @(posedge clock or negedge reset)
124.begin
125.            if(~reset)  start_r <= 1'b0;
126.            else        start_r <= (s_rready & (rready_cnt==2'b11)) ? 1'b1 : 1'b0 ;
127.end

128.reg         [3:0]       read_cnt;
129.always @(posedge clock or negedge reset)
130.begin
131.            if(~reset)  read_cnt <= 4'b0;
132.            else        read_cnt <= ~s_read ? 4'b0 :
133.                        (read_cnt==4'd15) ? 4'd15 : read_cnt+1'b1;
134.end
```

❖ rready 상태에서 spi read 를 위한 addr 을 생성해야 함. 그러나 spi write, spi read 의 address가 동일한 값을 사용하기 때문에 생략

❖ 109 ~ 115 : write state 에서 사용하는 counter 생성

❖ 116 ~ 121 : rready state 에서 사용하는 counter 생성

❖ 122 ~ 127 : spi read 를 위한 start_r 신호 생성

❖ 128 ~ 134 : read state 에서 사용하는 counter 생성

state	transition	description
WRITE	write 가 완료되면 RREADY 상태로 전환	spi write
RREADY	3-clock 후에 READ 상태로 전환	read 전에 addr, data, start_r 신호 생성
READ	read 가 완료되면 IDLE 상태로 전환	spi read

SPI Controller Implementation

➢ *spi_controller_exam_0.xpr*

➢ *SPI Task Implementation* ➔ *Code Implementation*

spi_task.v()

```
135.wire      [7:0]       rdata  ;
136.wire                  ss     ;
137.wire                  sck    ;
138.wire                  mosi   ;
139.wire                  miso   ;
140.wire                  done_mst;
141.spi_master            spi_master_u1(
142.          .reset    (reset   ),
143.          .clock    (clock   ),
144.          .freq     (SPI_FREQ),
145.          .start_w  (start_w ),
146.          .start_r  (start_r ),
147.          .addr     (addr    ),
148.          .wdata    (wdata   ),
149.          .rdata    (rdata   ),
150.          .done     (done_mst),
151.          .ss       (ss      ),
152.          .sck      (sck     ),
153.          .mosi     (mosi    ),
154.          .miso     (miso    )
155.);
156.spi_slave  spi_slave_u1(
157.          .reset    (reset   ),
158.          .clock    (clock   ),
159.          .ss       (ss      ),
160.          .sck      (sck     ),
161.          .mosi     (mosi    ),
162.          .miso     (miso    )
163.);
```

❖ 77 ~ 83 : spi_master, spi_slave를 연결하기 위한 신호 생성

❖ 84 ~ 108 : spi master module 추가

❖ 84 ~ 108 : spi slave module 추가

SPI Controller Implementation

➢ spi_controller_exam_0.xpr

➢ SPI Task Implementation ➔ Code Implementation

spi_task.v()

```
164.reg         [7:0]           led;
165.always @(posedge clock or negedge reset)
166.begin
167.            if(~reset)      led <= 4'b0;
168.            else            led <= (s_read & (read_cnt==4'd15) & done_mst) ? rdata : led;
169.end
```

❖ 164 ~ 169 : spi read 데이터를 led 로 출력함. s_read 상태에서 spi read가 완료되면 (done_mst 가 active 되면) rdata 값을 led로 출력

```
170.// ----------------------------------------
171.// 4) state transition
172.always @(posedge clock or negedge reset)
173.begin
174.            if(~reset)      m_state <= 3'b0;
175.            else            m_state <= (s_idle   & (btn2 != 4'b0 ) ) ? M_WREADY :
176.                            (s_wready & (wready_cnt==2'b11) ) ? M_WRITE  :
177.                            (s_write  & (write_cnt==4'd15) & done_mst ) ? M_RREADY :
178.                            (s_rready & (rready_cnt==2'b11)) ? M_READ   :
179.                            (s_read   & (read_cnt==4'd15) & done_mst  ) ? M_IDLE   :
180.                            m_state ;
181.end

182.endmodule
```

❖ 170 ~ 182 : 상태 전이를 구현

➔버튼이 입력되면 [idle ➔ wready]로 ➔ [wready 에서 3-clock 후에 write] 로

➔ spi write 완료되면 rready 로 ➔ rready 에서 3-clock 후에 read 로

➔ spi read 완료시 : idle 상태로 전환

SPI Controller Implementation

➔ [버튼 노이즈 제거 기능 구현]

➔ [*SPI Task* 구현]

- ➔ *Port* 정의 & *Code Implementation*
- ➔ *State* 정의
- ➔ *Code Implementation*
- ➔ ***Test Bench***

SPI Controller Implementation

➢ *spi_controller_exam_0.xpr*

➢ *SPI Task Implementation* ➔ *Code Implementation* ➔ ***Test Bench*** *: spi_controller_exam_0.v ()*

```
1.  `timescale 1ns / 1ps
2.  module spi_task_tb();
3.  reg                         reset, clock;
4.  initial     begin
5.              reset = 0;
6.              clock = 0;
7.  #10000      reset = 1;
8.  end
9.  always  #5  clock = ~clock;                 // 100 Mhz
```

❖ 2 ~ 9 : reset, clock 생성

```
10. reg         [19:0]          cnt ;
11. always @(posedge clock or negedge reset)
12. begin
13.             if(~reset)      cnt <= 20'd0;
14.             else            cnt <= cnt+1'b1;
15. end
```

❖ 10 ~ 15 : 내부에서 사용하는 counter를 생성

```
16. reg         [3:0]           btn;
17. always @(posedge clock or negedge reset)
18. begin
19.             if(~reset)      btn <= 4'd0;
20.             else            btn <=      (cnt==20'd10000) ? 4'd1 : (cnt==20'd10100) ? 4'd0 :      // 4'd1 ➔ 0001
21.                                         (cnt==20'd20000) ? 4'd2 : (cnt==20'd20100) ? 4'd0 :      // 4'd2 ➔ 0010
22.                                         (cnt==20'd30000) ? 4'd4 : (cnt==20'd30100) ? 4'd0 :      // 4'd4 ➔ 0100
23.                                         (cnt==20'd40000) ? 4'd8 : (cnt==20'd40100) ? 4'd0 :      // 4'd8 ➔ 1000
24.                                         (cnt==20'd50000) ? 4'd1 : (cnt==20'd50100) ? 4'd0 :      // 4'd1 ➔ 0001
25.                                         (cnt==20'd60000) ? 4'd2 : (cnt==20'd60100) ? 4'd0 :      // 4'd2 ➔ 0010
26.                                         (cnt==20'd70000) ? 4'd4 : (cnt==20'd70100) ? 4'd0 :      // 4'd4 ➔ 0100
27.                                         (cnt==20'd80000) ? 4'd8 : (cnt==20'd80100) ? 4'd0 :      // 4'd8 ➔ 1000
28.                                         (cnt==20'd90000) ? 4'd1 : (cnt==20'd90100) ? 4'd0 : btn;
29. end
```

❖ 16 ~ 29 : btn0, btn1, btn2, btn3, btn0, btn1, btn2, btn3, btn0 의 순서로 btn 입력을 생성

```
30. wire        [7:0]           led;
31. spi_task  spi_task_u1(
32.             .reset (reset ), .clock (clock),  .btn(btn ),  .led (led )
33. );
34. endmodule
```

❖ 30 ~ 34 : spi_task 모듈

SPI Controller Implementation

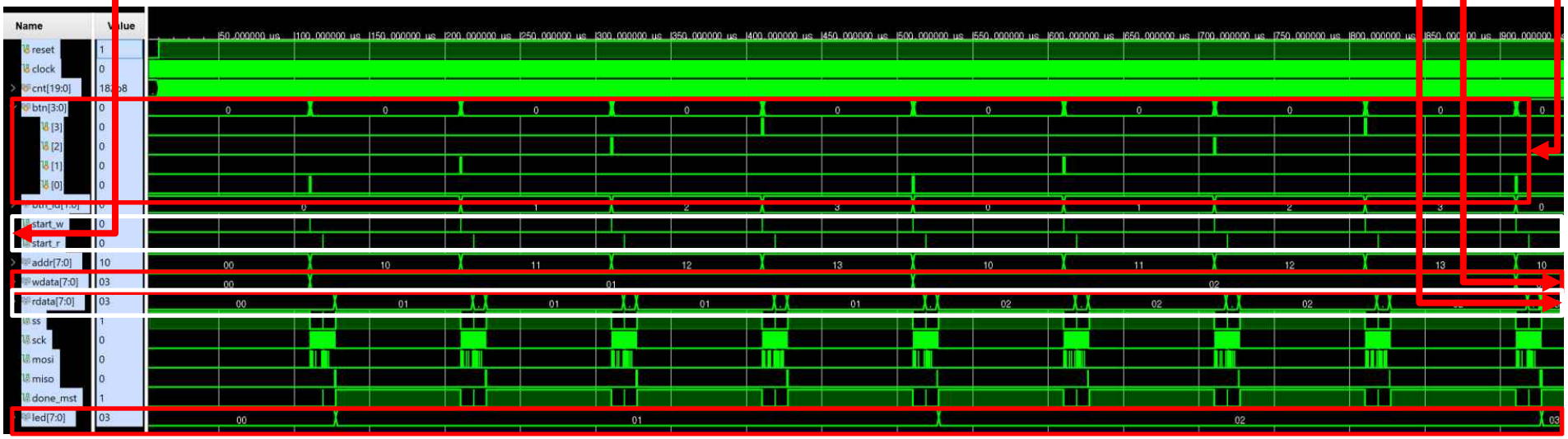
➢ *spi_controller_exam_0.xpr*

➢ *SPI Controller Implementation* ➔ *Code Implementation* ➔ *Test Bench :* ***spi_controller_exam_tb . v ()***

- ***Simulation Reault Check ()*** ➔ *btn_in.v* 파일을 수정 ➔ [max_cnt 의 값을 20'd10 로 수정]

```
1. //parameter                max_cnt = 20'd1_000_000;   // 10ms
2. parameter  max_cnt = 20'd10;              // for sim, 1000 : 10us, 100: 1us, 10 : 0.1us
```

- btn의 입력이 순차적으로 btn[0], btn[1], btn[2], btn[3], btn[0], btn[1], btn[2], btn[3], btn[0] 발생
- btn이 입력될 때 마다, addr, wdata, start_w, start_r 신호가 생성되어 spi write, spi read 가 진행
- wdata, rdata가 동일한 값이 되는 것 확인 ➔ 최종적으로 led 에 wdata와 동일한 데이터



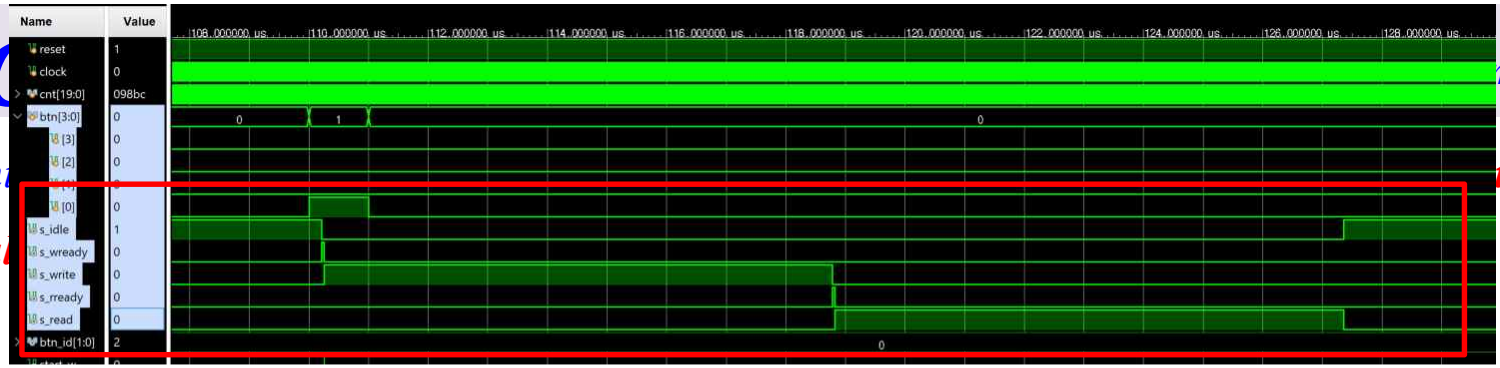
SPI C

m_0.xpr

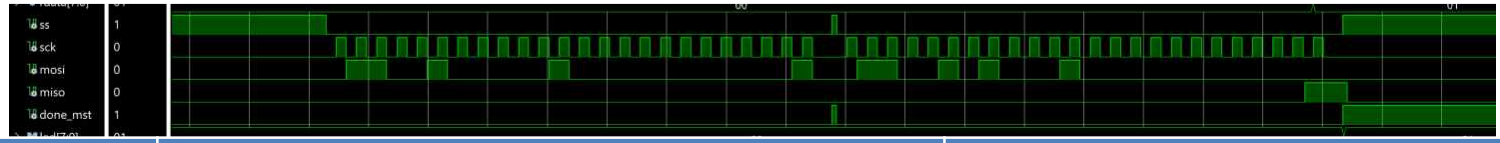
- *SPI Con*
  - ***Simul***

tb . v ()

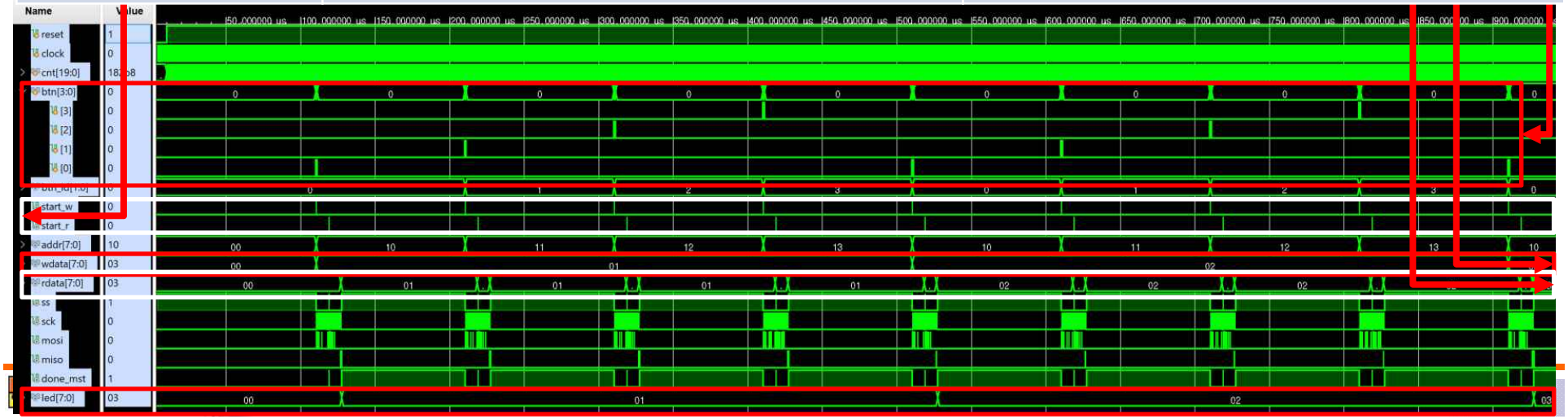
]



✓ 버튼[0]가 눌러졌을 때의 SM의 변화 : s_idle ➔ s_wready ➔ s_write ➔ s_rready ➔ s_read -> s_idle 의 순으로 state가 변화



state	transition	description
IDLE	버튼이 입력되면 WREADY 상태로 전환	idle state
WREADY	3-clock 후에 WRITE 상태로 전환	write 전에 addr, data, start_w 신호 생성
WRITE	write 가 완료되면 RREADY 상태로 전환	spi write
RREADY	3-clock 후에 READ 상태로 전환	read 전에 addr, data, start_r 신호 생성
READ	read 가 완료되면 IDLE 상태로 전환	spi read



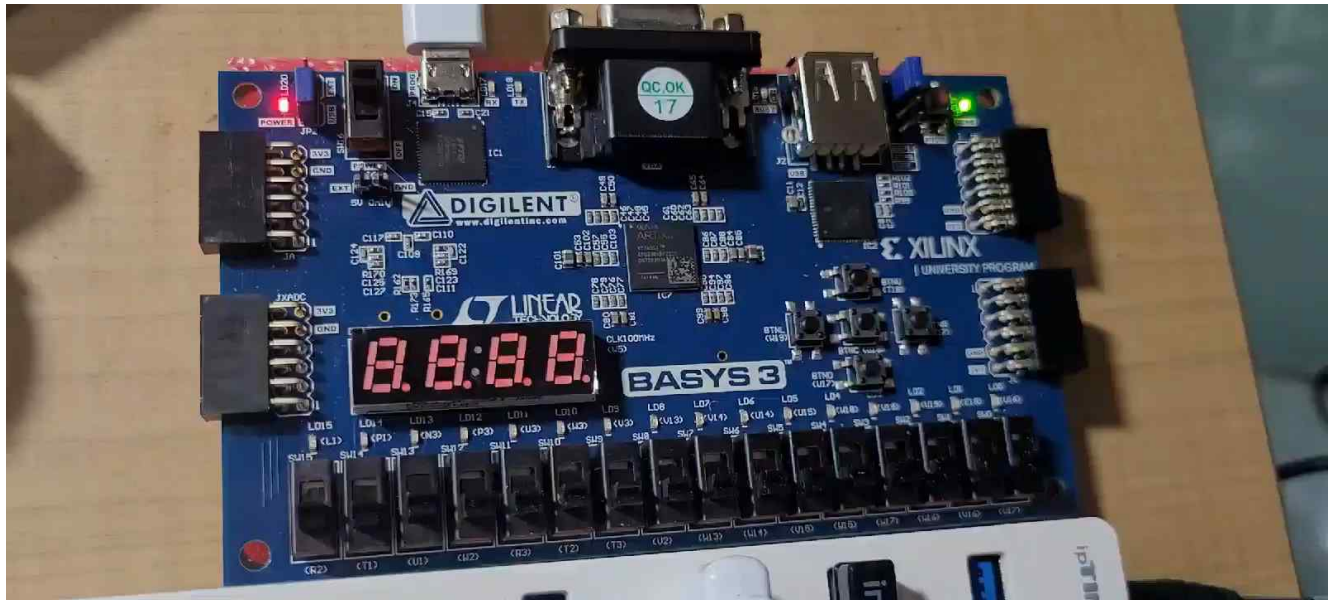
SPI Controller Implementation ➢ *spi_controller_exam_1.xpr*

➢ *SPI Controller Implementation* ➔ *Code Implementation* ➔ *Test Bench* ➔ *Bitstream* 생성 , 다운로드

- ***Basys3 Download Reault Check ()*** ➔ *btn_in.v* 파일을 수정 ➔ [max_cnt : 20'd1_000_000 로 수정]

```
1. parameter    max_cnt = 20'd1_000_000;  // 10ms
2. //parameter  max_cnt = 20'd10;          // for sim, 1000 : 10us, 100: 1us, 10 : 0.1us
```

- btn의 입력이 순차적으로 btn[0], btn[1], btn[2], btn[3], btn[0], btn[1], btn[2], btn[3], btn[0] 발생
- btn이 입력될 때 마다, addr, wdata, start_w, start_r 신호가 생성되어 spi write, spi read 가 진행
- wdata, rdata가 동일한 값이 되는 것 확인 ➔ 최종적으로 led 에 wdata와 동일한 데이터



SPI Controller Implementation

➔ [버튼 노이즈 제거 기능 구현]

➔ [*SPI Task* 구현]

➔ Port 정의

➔ *XDC Implementation & Program* : 실습

SPI Controller Implementation

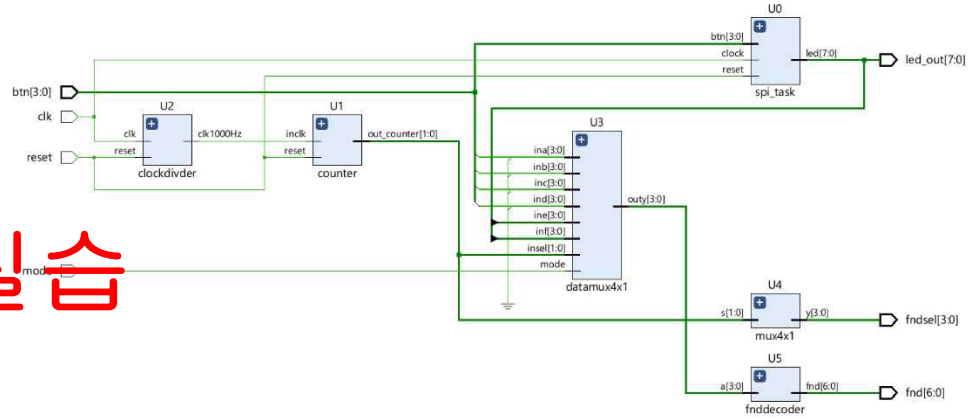
➢ *spi_controller_exam_1.xpr*

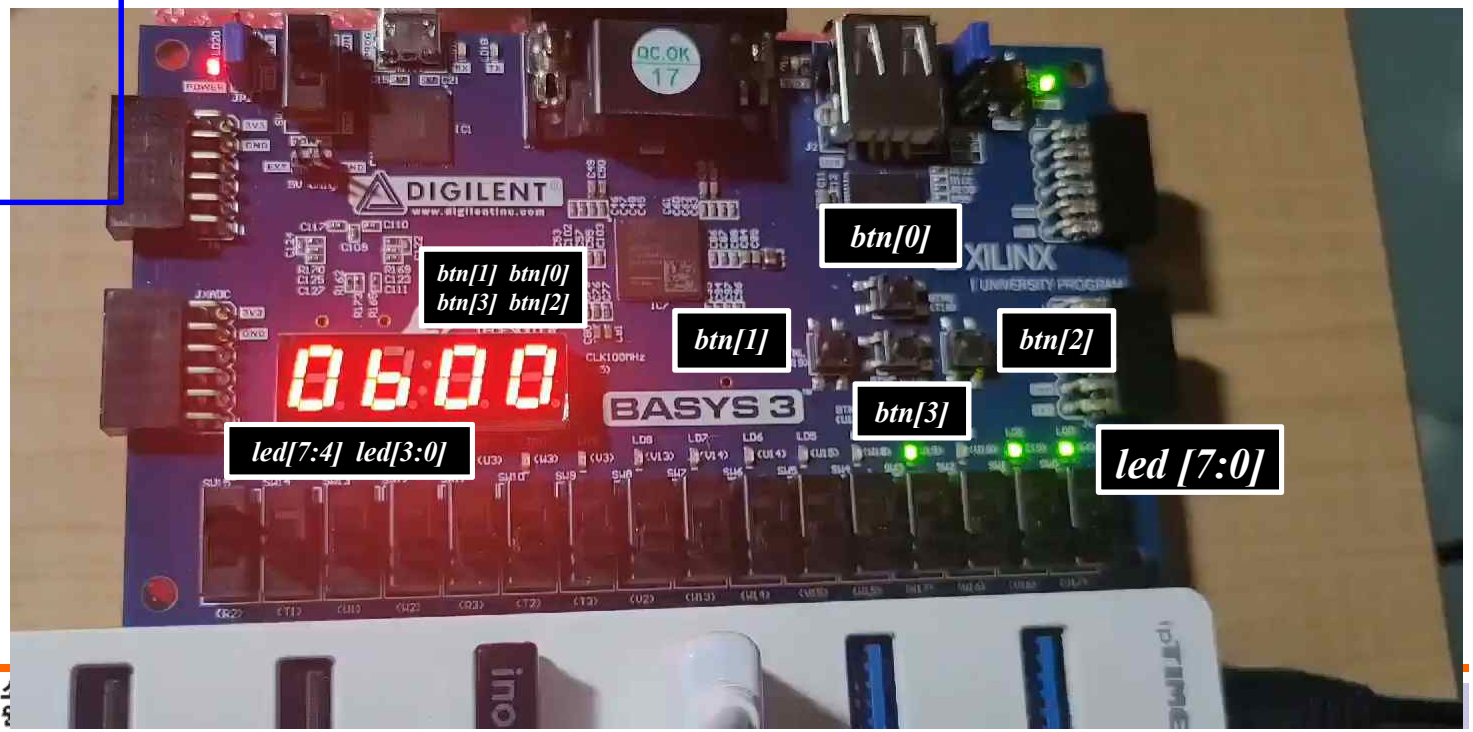
➢ *SPI Controller Implementation ➔ Code Implementation ➔ Test Bench ➔ Bitstream 생성 , 다운로드*

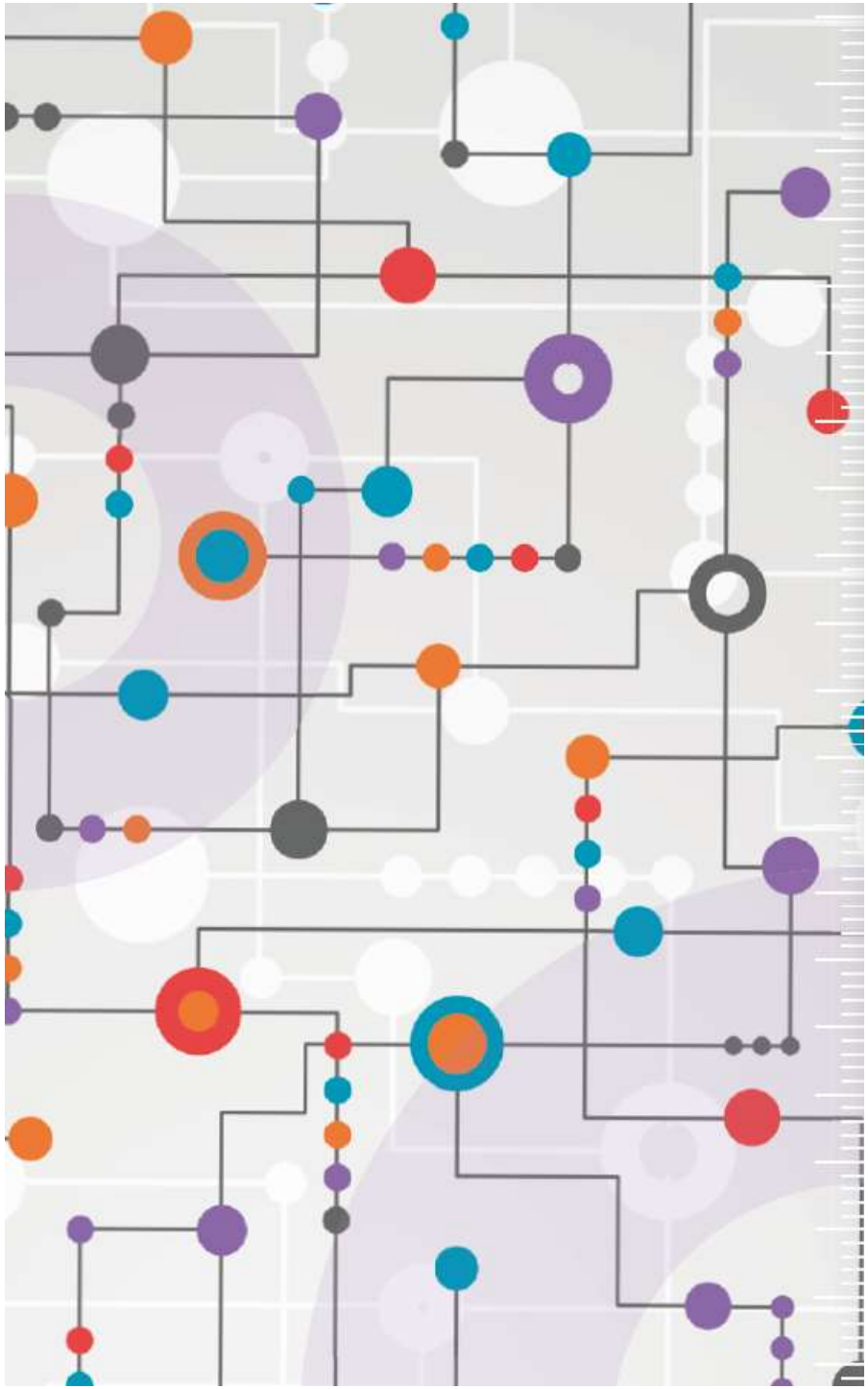
- ***Basys3 Download Reault Check () ➔ btn_in.v 파일을 수정 ➔ [max_cnt : 20'd1_000_000 로 수정]***

- *btn[0] ➔ BTNR (T18)*
- *btn[1] ➔ BTND (W19)*
- *btn[2] ➔ BTNU (T17)*
- *btn[3] ➔ BTNU (U17)*
- *reset ➔ SW15*
- *clk ➔ System Clock [100MHz]*
- *FND4 ➔ rdata 10 (led[7:4])*
- *FND3 ➔ rdata 0(led[7:4])*
- *FND2 ➔ Mode 0 : btn[1]*
- *FND1 ➔ Mode 0 : btn[0]*
- *FND2 ➔ Mode 1 : btn[3]*
- *FND1 ➔ Mode 1 : btn[2]*
- *LD 7 ~ LD 0 ➔ led_out [7:0]*

✓ 실습







수고하셨습니다.