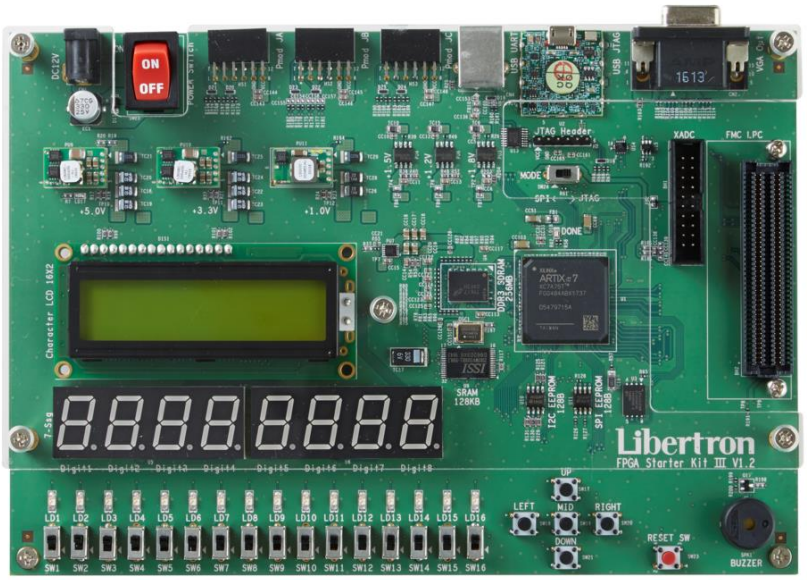
FPGA Starter Kit III

SRAM Tutorial (v1.2B)

Authors : 기술지원팀 김민석 팀장



Libertron Co., Ltd

1. 개요

- 본 문서에서는 SRAM 의 동작을 이해하고, SRAM 의 동작 확인을 위한 로직 블록을 설계 및 다운로드 하는 법을 기술 한다.

2. 상세 설명

2.1 준비 사항 및 테스트 환경

2.1.1 준비사항

- FPGA Starter Kit III V1.2B
- Power Adapter
- USB Type B Cable (FPGA 다운로드 용)

2.1.2. 테스트 환경

- Windows 10 / Vivado 2018.2 (상/하위 버전 관계 없음)

2.2 프로젝트 세부 설명

2.2.1 디자인 동작 방향

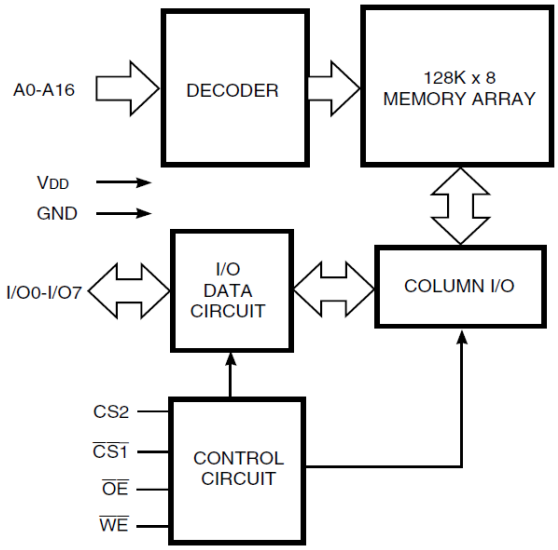
- 본 자료는 FSK III 에 장착되어 있는 128K x 8bit, 128KB 용량의 SRAM (Async) 을 기반으로 제작 되었다.

- FPGA 외부에 장착되어 있는 SRAM 특성상, Vivado 에서 Simulation 진행을 할 수 없어, FSM (Finite State Machine) 을 이용하여 디자인 하였다.
  SRAM (Async) 동작이 완료되면 FSK III 에 있는 LED 가 켜지도록 디자인 되었으며, Dip Switch 로 이를 확인할 수 있다.

2.2.2 SRAM 동작 이해

- 본 SRAM 의 당사에서 제공하는 CD Data 의 Schematic.pdf , 13p에서 확인할 수 있으며, 데이터 시트는 검색 사이트에 62WV1288ALL-258449.pdf 파일을 다운받아 참고 한다

- 하기와 같은 동작 블록도를 자료를 통해 확인할 수 있다.



PIN DESCRIPTIONS

A0-A16	Address Inputs
$\overline{CS1}$	Chip Enable 1 Input
CS2	Chip Enable 2 Input
$\overline{OE}$	Output Enable Input
$\overline{WE}$	Write Enable Input
I/O0-I/O7	Input/Output
NC	No Connection
VDD	Power
GND	Ground

TRUTH TABLE

Mode	$\overline{WE}$	$\overline{CS1}$	CS2	$\overline{OE}$	I/O Operation
Not Selected	X	H	X	X	High-Z
(Power-down)	X	X	L	X	High-Z
Output Disabled	H	L	H	H	High-Z
Read	H	L	H	L	DOUT
Write	L	L	H	X	DIN

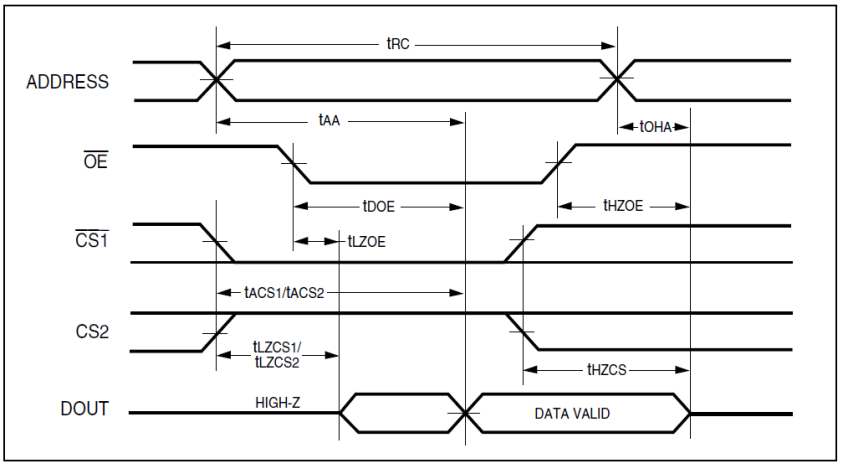
- 상기 그림을 보면 SRAM Memory 의 각 핀에 대한 동작 정보가 나와 있다.

Write - WE : Low , OE : X (don't care), CS1 : Low, CS2 : High 신호

Read - WE : High, OE : Low , CS1 : Low, CS2 : High

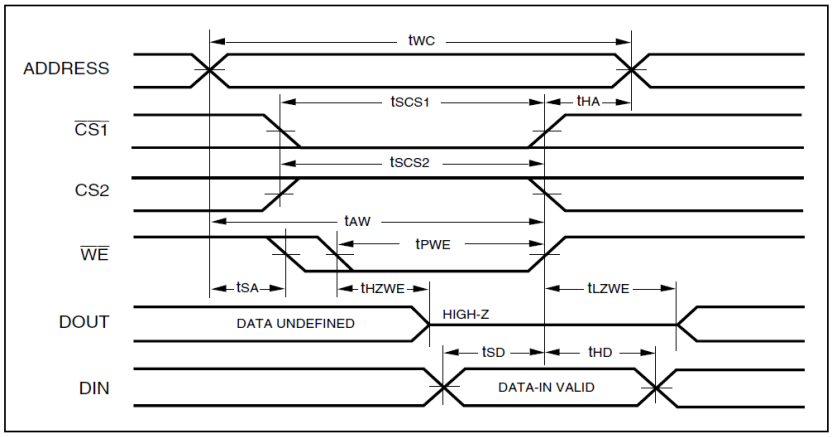
- Read / Write 의 동작 타이밍도는 하기의 그림과 같이 확인 할 수 있다.

READ CYCLE NO. 2(1,3) ($\overline{CS1}$, CS2, $\overline{OE}$ Controlled)

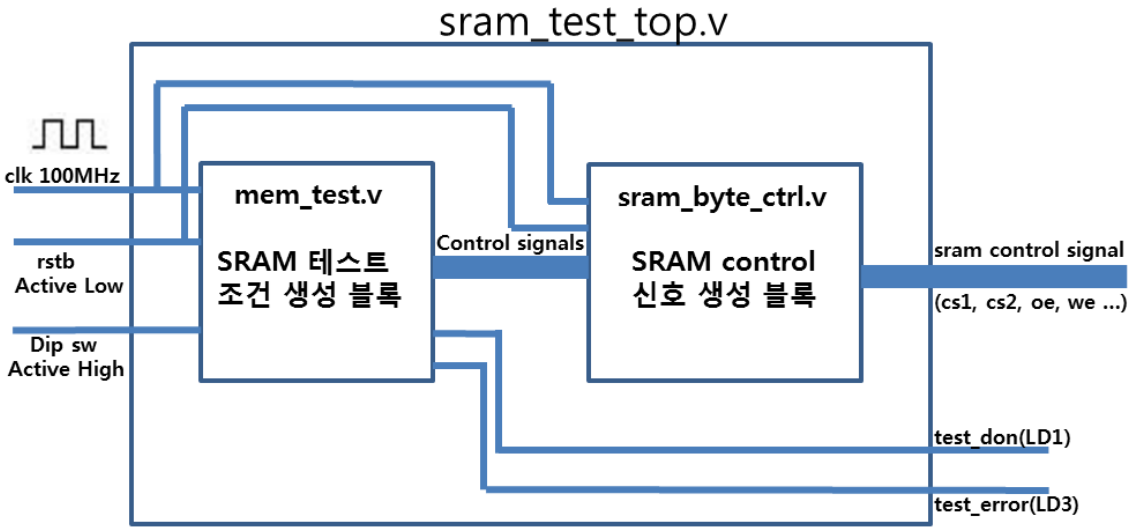


Notes:
1. $\overline{WE}$ is HIGH for a Read Cycle.
2. The device is continuously selected. $\overline{OE}$, $\overline{CS1}$= V_{IL}, CS2=$\overline{WE}$=V_{IH}.
3. Address is valid prior to or coincident with $\overline{CS1}$ LOW and CS2 HIGH transition.

WRITE CYCLE NO. 1 ($\overline{CS1}$/CS2 Controlled, $\overline{OE}$ = HIGH or LOW)



2.2.3 디자인 동작 블록도



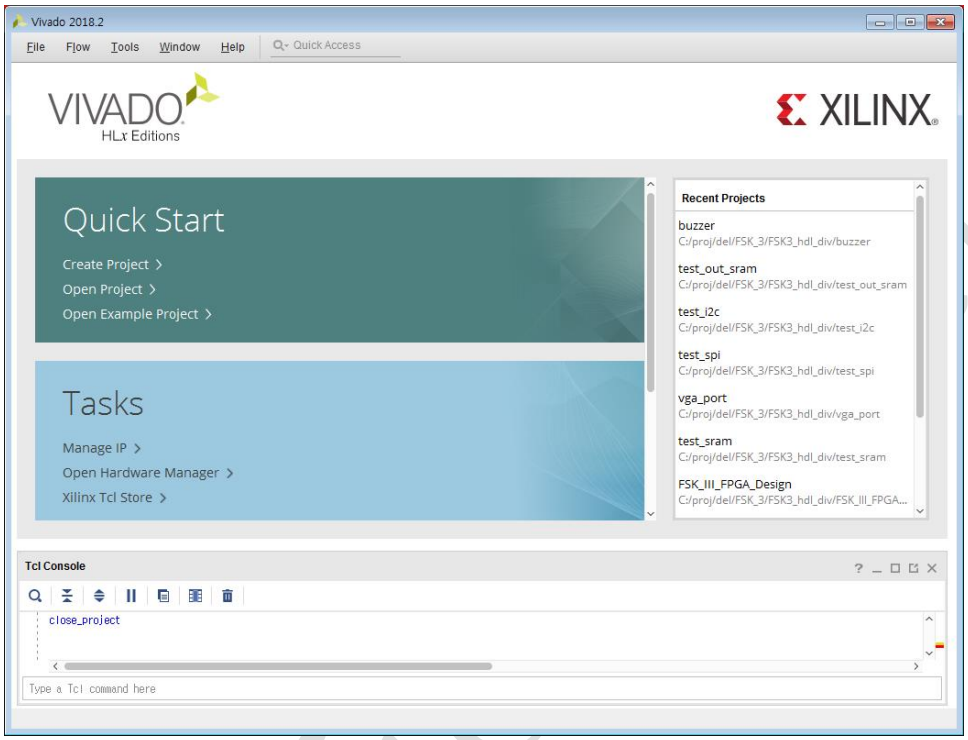
- 상기 블록도와 같이 FSK III 에서 들어오는 100MHz 의 Clock 을 기반으로, 로직이 동작하며 mem_test.v 블록과 sram_byte_test.v 블록으로 나눠진다.

Design Block	Function
mem_test.v	기본적으로 SRAM 테스트 조건 (IDLE, Write, Read 조건) 등의 Test 환경을 생성하여 Dip SW 에서 Test 시작입력을 받으면 sram_byte_ctrl 블록에 SRAM addr 및 data를 전달하고 data를 다시 받아, SRAM 에 정상적으로 저장되었는지 확인한다. 동작 확인 결과에 따라, LED (LD1) 또는 LED (LD3)가 켜지게 된다.
sram_byte_ctrl.v	기본적으로 SRAM Memory 를 컨트롤하는 Signal을 생성하며, SRAM Memory 에 Write/Read 한 정보를 mem_test 에 전달하여 mem_test 블록에서 SRAM Memory 에 정상적으로 데이터가 Write 되었는지 판단하게 한다.

2.2.4 Vivado 디자인 구성

1) Vivado New Project 실행

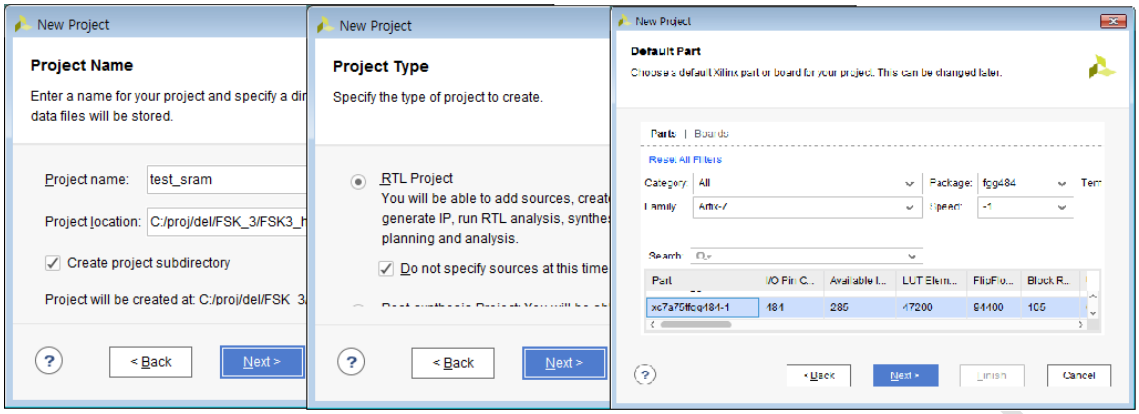
✳ Project 작업 경로 및 폴더에 특수문자 및 한글의 인식이 안되므로, 영어만 사용할 것



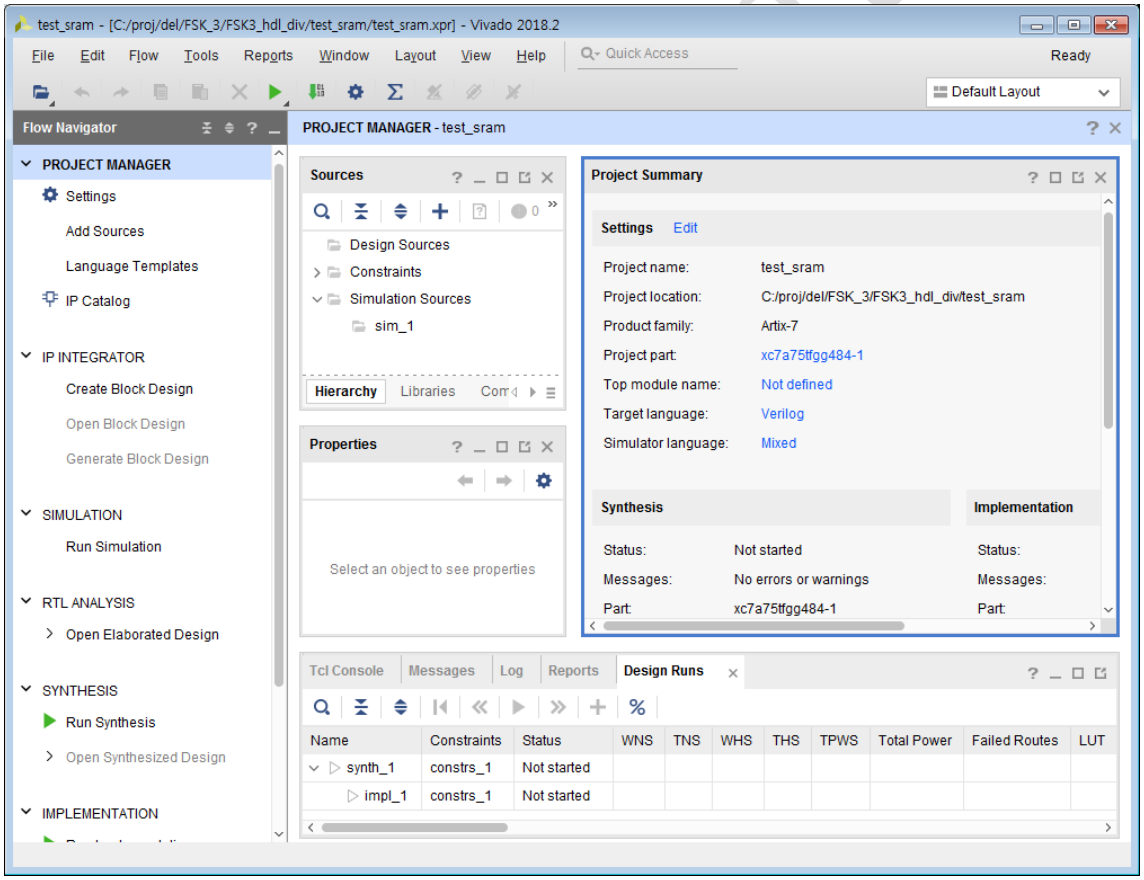
2) 상기 그림에서 다음과 같이 진행 한다.

Create Project ➔ Create a New Vivado Project 에서 Next 클릭
➔ Project Name 란에 "test_sram" 입력 후 Next 클릭
➔ RTL Project 선택 후 Next 클릭
➔ Default Part 란에서 하기와 같이 선택 (하기 그림 참조)

Family	Artix-7
Package	Fgg484
Speed	-1
Full Part Name	XC7A75TFGG484-1

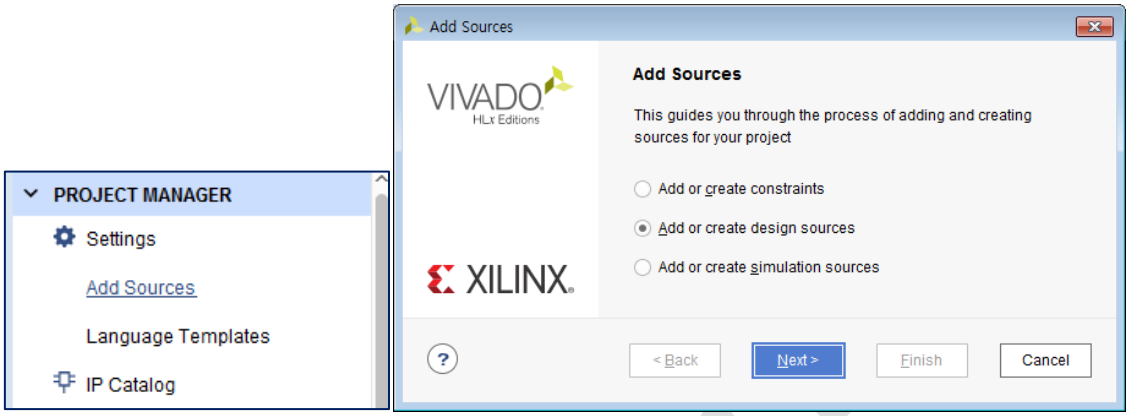


3) 상기 작업 후 New Project Summary 창이 나타나면 Finish 클릭
작업이 완료 되면 하기와 같이 Vivado 초기 프로젝트 화면이 나타난다.

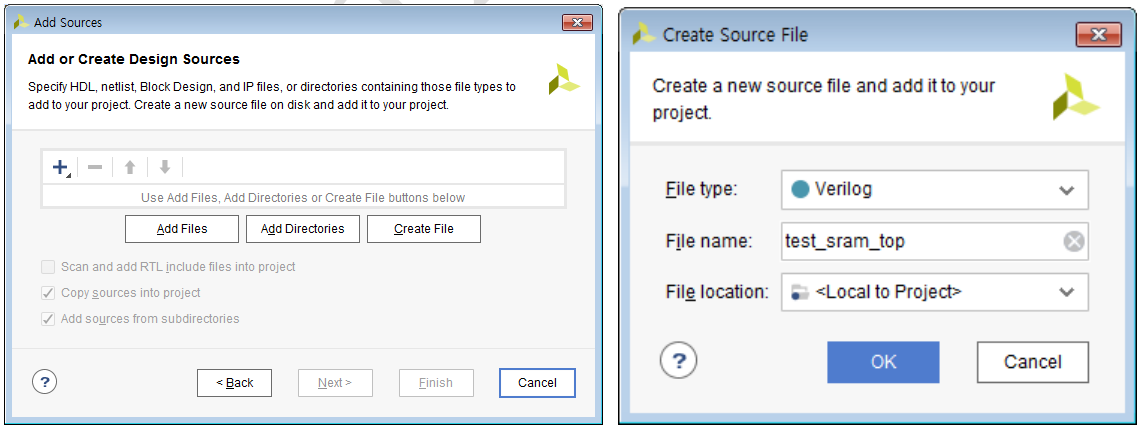


2.2.5 Vivado New Design Create

1) 하기의 그림과 같이 Project Manager 창에 있는 Add Source 를 클릭
➔ Add or create design sources 선택 및 Next 클릭



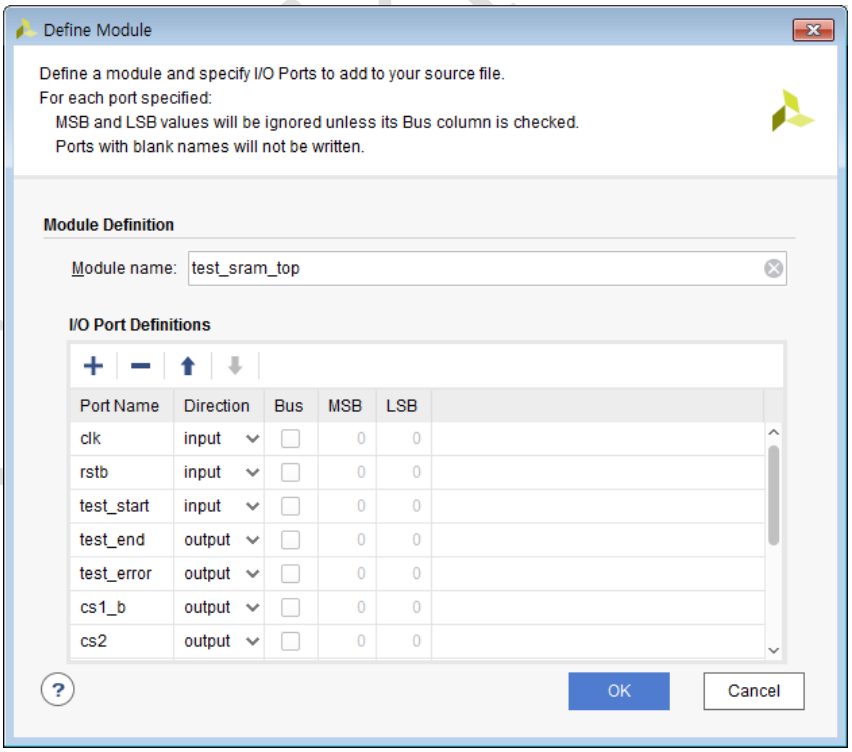
2) Add Sources 창에서 Create File ➔
File Name 란에 "test_sram_top" 입력 후 Ok 버튼 클릭 및 Finish 클릭
(User 가 임의의 영어로 이름을 지정해도 됨)



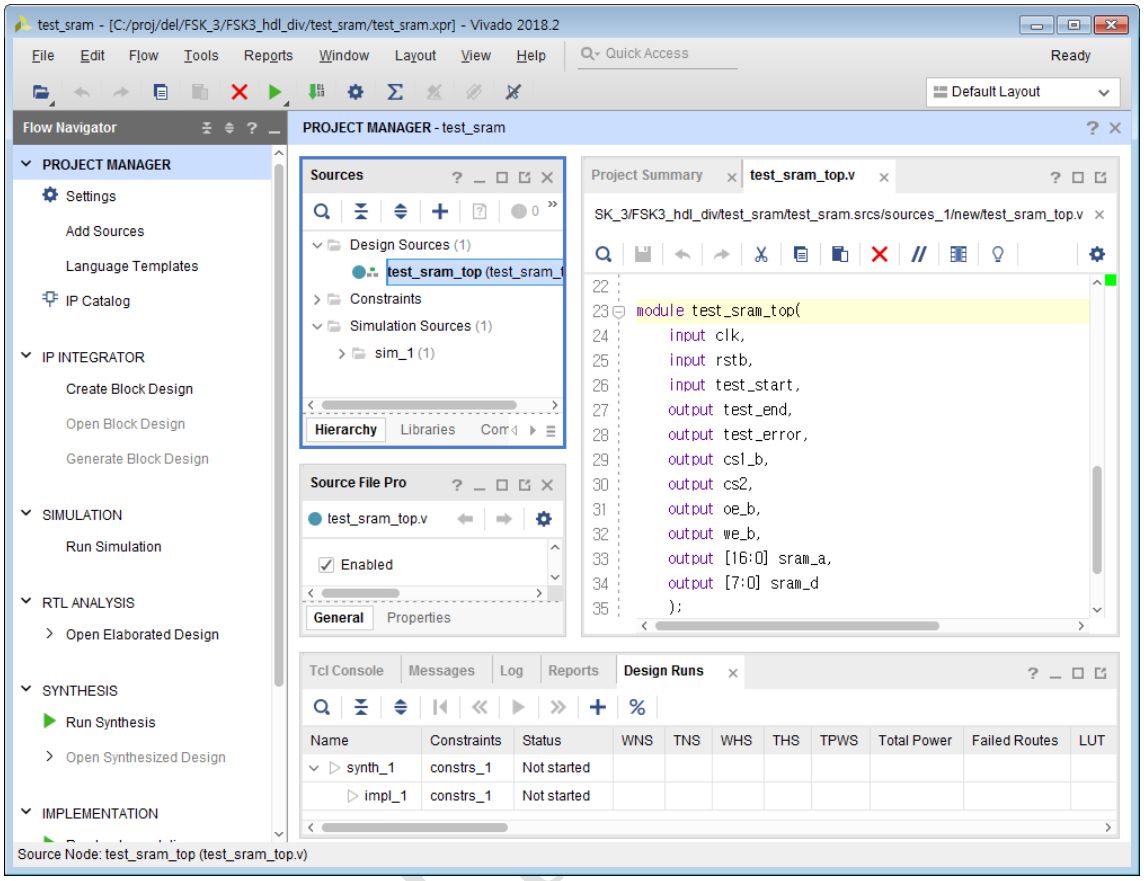
3) **Define Module 창에서 하기와 같이 입력하고 OK 클릭**

✳ **test_sram_top.v 디자인**

Port Name	Direction	Bus	MSB	LSB
clk	Input			
rstb	Input			
test_start	Input			
test_end	Output			
test_error	Output			
cs1_b	Output			
cs2	Output			
oe_b	Output			
we_b	Output			
sram_a	Output	✓	16	0
sram_d	Output	✓	7	0



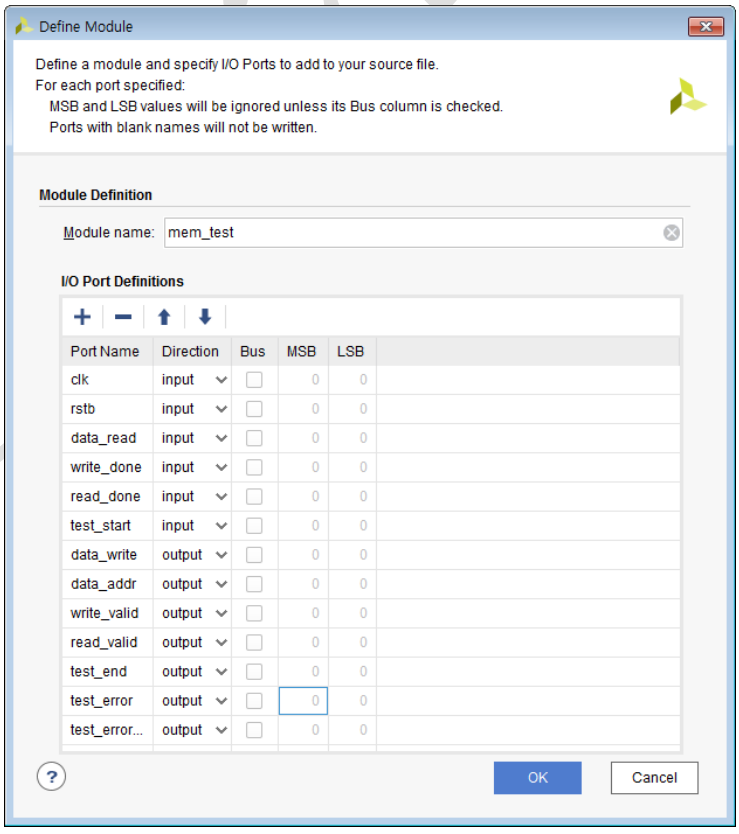
✻ 작업이 완료 되면 하기와 같이 Vivado 창이 나타난다.



✻ 상기와 같은 방식으로, 하기에 나와 있는 mem_test.v , sram_byte_ctrl.v 디자인도 Add Sources 하여 디자인 작업을 완성하자.

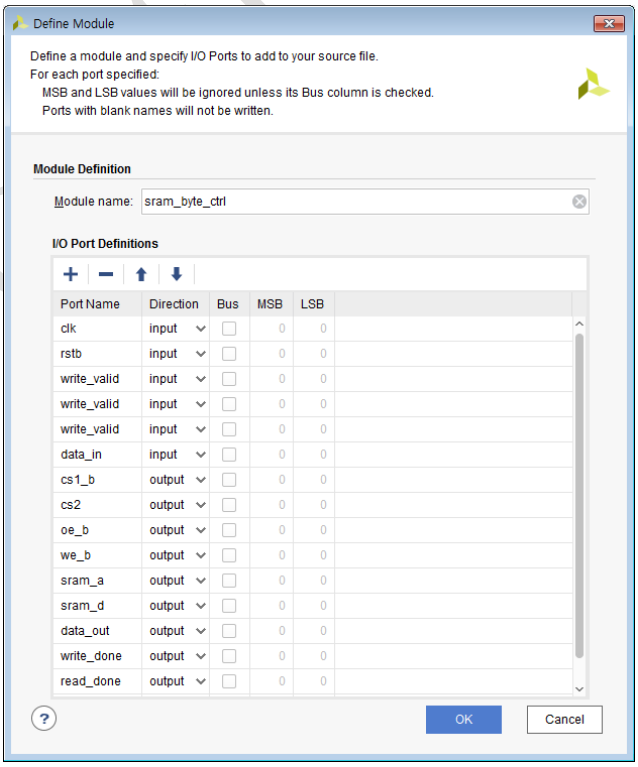
✳ **mem_test.v 디자인**

Port Name	Direction	Bus	MSB	LSB
clk	Input			
rstb	Input			
data_read	Input			
write_done	Input			
read_done	Input			
test_start	Input			
data_write	output			
data_addr	output			
write_valid	output			
read_valid	output			
test_end	output			
test_error	output			



✳ **sram_byte_ctrl.v 디자인**

Port Name	Direction	Bus	MSB	LSB
clk	Input			
rstb	Input			
write_valid	Input			
read_valid	Input			
addr_in	Input			
data_in	Input			
cs1_b	output			
cs2	output			
oe_b	output			
we_b	output			
sram_a	output			
sram_d	output			
write_done	output			
read_done	output			



2.2.6 Viavdo 디자인 입력 작업

- 본 자료에서는 SRAM Memory를 사용하기 위해 본 디자인 (test_sram_top.v) 과 서브 디자인 2개 (mem_test.v , sram_byte_ctrl.v) 로 구성되어 있다.

- 상기 디자인을 순서대로 나열했으며, 순서대로 디자인 설명을 진행 한다.

A. test_sram_top.v 디자인 입력

✱ 하기의 내용과 같이 디자인을 입력한다.

```
module sram_test_top
(/*************************************************
** Input Signal Define                          **
*************************************************/
input   wire            clk             ,
input   wire            rstb            ,
input   wire            test_start      ,
/*************************************************
** Output Signal Define                         **
*************************************************/
output wire             test_end        ,
output wire             test_error      ,
output wire             cs1_b           ,
output wire             cs2             ,
output wire             oe_b            ,
output wire             we_b            ,
```

```
output wire [16:0]        sram_a            ,
inout   wire [07:0]       sram_d      );
/*************************************************
** Wire Definition                                **
*************************************************/
// Memory Test Block Signal
wire   [07:0]    data_read    ;
wire   [07:0]    data_write   ;
wire   [16:0]    data_addr    ;
wire             write_valid ;
wire             read_valid   ;
wire             write_done   ;
wire             read_done    ;
/*************************************************
** Memory Test Block
*************************************************/
mem_test
#(.DATA_WIDTH      (              8),
 .ADDR_WIDTH       (             17),
 .DATA_NUM         (     128*1024-1)  )
u_mem_test
( .clk             (clk            ),
 .rstb             (rstb           ),
```

```
 .data_read          (data_read        ),
 .write_done         (write_done       ),
 .read_done          (read_done        ),
 .test_start         (test_start       ),

.data_write         (data_write       ),
 .data_addr          (data_addr        ),
 .write_valid        (write_valid      ),
 .read_valid         (read_valid       ),
 .test_end           (test_end         ),
 .test_error         (test_error       )     );
/***************************************************
** SRAM Control Block
***************************************************/
sram_byte_ctrl
u_sram_byte_ctrl
( .clk               (clk              ),
 .rstb               (rstb             ),
 .write_valid        (write_valid      ),
 .read_valid         (read_valid       ),
 .addr_in            (data_addr        ),
 .data_in            (data_write       ),
.cs1_b               (cs1_b            ),
```

```
 .cs2              (cs2            ),
 .oe_b             (oe_b           ),
 .we_b             (we_b           ),
 .sram_a           (sram_a         ),
 .sram_d           (sram_d         ),
 .data_out         (data_read      ),
 .write_done       (write_done     ),
 .read_done        (read_done      )   );
endmodule
```

∗ 상기 디자인은 SRAM 를 동작시키기 위한 test_sram_top.v 디자인으로, mem_test.v , sram_byte_ctrl.v 을 연결하여 전체 디자인을 기능에 맞게 연결하는 역할을 한 디자인 블록이다.

2.2.7 test_sram_top.v : Design 요약 설명 (코드 내 주석)

```
/****************************************************
********Input Signal Define ************************
****************************************************/
/****************************************************
********Output Signal Define ***********************
****************************************************/
```

▶ **Input, Output Define**

- test_sram_top.v 디자인의 입출력을 선언한 부분

```
/****************************************************
** Wire Definition                                 **
****************************************************/
```

▶ **Wire Definition**

- Memory 를 테스트하기 위한 시그널을 정의하는 로직

```
/****************************************************
** Memory Test Block
****************************************************/
```

▶ **Memory Test Block**

- mem_test.v 디자인을 Top 디자인과 연결하는 로직

```
/****************************************************
** SRAM Control Block
****************************************************/
```

▶ **SRAM Control Block**

- sram_byte_ctrl.v 디자인을 Top 디자인과 연결하는 로직

B. mem_test.v 디자인 입력

✱ 하기의 내용과 같이 디자인을 입력한다.

```
module mem_test
#(
 parameter DATA_WIDTH          = 8           ,
 parameter ADDR_WIDTH          = 16          ,
 parameter DATA_NUM            = 128
)
(
```

```
/*************************************************
** Input Signal Define                          **
*************************************************/
input   wire                    clk             ,
input   wire                    rstb            ,
input   wire [DATA_WIDTH-1:0]   data_read       ,
input   wire                    write_done      ,
input   wire                    read_done       ,
input   wire                    test_start      ,
/*************************************************
** Output Signal Define                         **
*************************************************/
output wire [DATA_WIDTH-1:0]   data_write      ,
output wire [ADDR_WIDTH-1:0]   data_addr       ,
output reg                     write_valid     ,
output reg                     read_valid      ,
output reg                     test_end        ,
output reg                     test_error      );
reg   [ADDR_WIDTH-1:0]   test_error_cnt;
/*************************************************
** Parameter Definition                         **
*************************************************/
localparam IDLE             = 6'b000001;
```

```
localparam WRITE          = 6'b000010;
localparam INIT_READ      = 6'b000100;
localparam READ           = 6'b001000;
localparam TEST_DONE      = 6'b010000;
localparam RESTART_TEST = 6'b100000;
/*****************************************************
** Wire Definition                                   **
*****************************************************/
wire                      test_start_r    ;
/*****************************************************
** Reg Definition                                    **
*****************************************************/
reg    [05:0]             cs, ns          ;
reg    [01:0]             test_start_d    ;
reg    [ADDR_WIDTH-1:0]   data_cnt        ;
reg    [ADDR_WIDTH-1:0]   data_cnt_buf    ;
reg    [DATA_WIDTH-1:0]   data_read_buf   ;
reg                       write_done_buf ;
reg                       read_done_buf   ;
/*****************************************************************
-- Test Start Edge Detect
*****************************************************************/
always @(posedge clk or negedge rstb) begin
```

```
    if (!rstb) begin
        test_start_d    <= 2'd0;
    end
    else begin
        test_start_d    <= {test_start_d[0], test_start};
    end
end

assign test_start_r   = (test_start_d == 2'b01);

/*****************************************************************
-- Done Signal Buffering
*****************************************************************/
always @(posedge clk or negedge rstb) begin
    if (!rstb) begin
        write_done_buf <= 1'b0;
        read_done_buf  <= 1'b0;
    end
    else begin
        write_done_buf <= write_done;
        read_done_buf  <= read_done;
    end
end
```

```
/*****************************************************************
-- Memory Test COM FSM
*****************************************************************/
always @( * ) begin
    case ( cs )
        IDLE     : if (test_start_r) begin
                       ns = WRITE;
                   end
                   else begin
                       ns = IDLE;
                    end
/*************************************************************/
        WRITE: if ((data_cnt == DATA_NUM-1) && write_done_buf) begin
                    ns = INIT_READ;
                  end
                  else begin
                       ns = WRITE;
                    end
/*************************************************************/
        INIT_READ     : ns = READ;
/*************************************************************/
        READ : if ((data_cnt == DATA_NUM-1) && read_done_buf) begin
                       ns = TEST_DONE;
```

```
                end
                else begin
                        ns = READ;
                end

/*************************************************************/
        TEST_DONE : if (test_start_r) begin
                         ns = RESTART_TEST;
                        end
                        else begin
                        ns = TEST_DONE;
                        end
/*************************************************************/
        RESTART_TEST: ns = WRITE;
/*************************************************************/
        default   : ns = IDLE;
    endcase
end
/*****************************************************************
// Memory Test SEQ FSM
*****************************************************************/
always @(posedge clk or negedge rstb) begin
    if (!rstb) begin
```

```
        cs                  <= IDLE;
        data_cnt            <= 'd0;
        data_cnt_buf        <= 'd0;
        data_read_buf       <= 'd0;
        test_error          <= 1'b0;
        test_error_cnt <= 'd0;
    end
    else begin
        cs <= ns;
/*************************************************************/
    case ( ns )
     IDLE : begin test_end <= 1'b0; write_valid <= 1'b0; read_valid <= 1'b0; end
    WRITE: begin test_end <= 1'b0; write_valid <= 1'b1; read_valid <= 1'b0; end
INIT_READ : begin test_end <= 1'b0; write_valid <= 1'b0; read_valid <= 1'b0;
end
     READ : begin test_end <= 1'b0; write_valid <= 1'b0; read_valid <= 1'b1; end
   TEST_DONE : begin test_end <= 1'b1; write_valid <= 1'b0; read_valid <=
1'b0; end
    default : begin test_end <= 1'b0; write_valid <= 1'b0; read_valid <= 1'b0; end
     endcase
/*************************************************************/
       if ((ns == INIT_READ) || (ns == IDLE) || (ns == RESTART_TEST))
       begin
            data_cnt      <= 'd0;
```

```
        end
        else if (write_done_buf || read_done_buf) begin
            data_cnt      <= data_cnt + 'd1;
        end
        data_cnt_buf    <= data_cnt;
        data_read_buf   <= data_read;

        // Error Check
        if ((cs == INIT_READ) || (cs == RESTART_TEST)) begin
            test_error      <= 1'b0;
            test_error_cnt <= 'd0;
        end
        else if ((cs == READ) && read_done_buf) begin
            if (data_cnt_buf[DATA_WIDTH-1:0] != data_read_buf) begin
                test_error      <= 1'b1;
                test_error_cnt <= test_error_cnt + 1;
            end
        end
    end
end
/*************************************************************/
assign data_write = data_cnt[DATA_WIDTH-1:0];
assign data_addr  = data_cnt;
```

```
endmodule
```

2.2.7 mem_test.v : Design 요약 설명 (코드 내 주석)

```
/****************************************************
******** Input Signal Define ************************
****************************************************/
/****************************************************
******** Output Signal Define ***********************
****************************************************/
```

▶ **Input, Output Signal Define**

- mem_test.v 디자인의 입출력을 정의한 로직

```
/****************************************************
******** Parameter Definition ***********************
****************************************************/
```

▶ **Parameter Definition**

- SRAM Memory 를 테스트하기 위한 IDLE, WRITE, INIT_READ, READ, TEST_DONE, RESTART_TEST 상태 등을 Parameter value 로 정의한 로직 이다.
차후 여기에 정의된 value 를 기반으로 FSM (Finite State Machine)을 활용하여 SRAM을 테스트하는 로직이 동작한다.

```
/*************************************************

******** Wire Definition*******************************

*************************************************/
/*************************************************

******** Reg Definition********************************

*************************************************/
```

▶ **Wire, Reg Definition**

- SRAM Memory 를 Test 하기 위한 wire 및 reg 를 선언한 부분

```
/*****************************************************************

-- Test Start Edge Detect

*****************************************************************/
```

▶ **Test Start Edge Detect**

- Dip SW (test_start) 로 들어오는 SRAM Test 시작 신호를 Clock에 맞게 받아 들여 인식하는 로직 이다.

```
/*****************************************************************

-- Done Signal Buffering

*****************************************************************/
```

▶ **Done Signal Buffering**

- sram_byte_ctrl 에서 SRAM read_done, write_done 신호가 생성되어 출력되는

것을 받아, Clock 에 맞게 저장하는 기능의 로직이다.

```
/*****************************************************************
-- Memory Test COM FSM
*****************************************************************/
```

▶ Memory Test COM FSM

- FSM 로직 블록으로, cs 변화값이 "Parameter Definition" 부에서 정의된 값과 일치 되면 그 정의된 동작을 하도록 하는 로직

```
/*****************************************************************
// Memory Test SEQ FSM
*****************************************************************/
```

▶ Memory Test SEQ FSM

- ns 값의 변화에 따라, "Parameter Definition" 에서 정의된 IDLE ~ TEST_DONE 상태일 때에 각 상태를 표시하는 신호의 값을 정의한다. SRAM Memory 에 저장할 데이터를 생성하고, 그 데이터가 SRAM 에서 read 나 write 가 제대로 됐는지 판단하여 신호를 출력 시키는 로직이다.

C. sram_byte_ctrl.v 디자인 입력

✱ 하기의 내용과 같이 디자인을 입력한다.

```
module sram_byte_ctrl
(
/*************************************************
** Input Signal Define                          **
*************************************************/
input   wire            clk         ,
input   wire            rstb        ,
input   wire            write_valid ,
input   wire            read_valid  ,
input   wire [16:0]     addr_in     ,
input   wire [07:0]     data_in     ,
/*************************************************
** Output Signal Define                         **
*************************************************/
output reg              cs1_b       ,
output reg              cs2         ,
output reg              oe_b        ,
output reg              we_b        ,
output wire [16:0]      sram_a      ,
inout   wire [07:0]     sram_d      ,
output reg   [07:0]     data_out    ,
```

```
output wire                 write_done   ,
output wire                 read_done       );



/*************************************************
** Pasrameter Definition                        **
*************************************************/
localparam IDLE               = 10'b0000000001;
localparam WRITE_OE           = 10'b0000000010;
localparam WRITE_WE_CS        = 10'b0000000100;
localparam WRITE_DATA         = 10'b0000001000;
localparam WRITE_WE_END = 10'b0000010000;
localparam READ_WE_CS         = 10'b0000100000;
localparam READ_OE            = 10'b0001000000;
localparam READ_OE_END        = 10'b0010000000;
localparam READ_DATA          = 10'b0100000000;
localparam READ_DATA_SP = 10'b1000000000;
/*************************************************
** Wire Definition                              **
*************************************************/
wire              wait_cmp      ;


/*************************************************
```

```
** Reg Definition                                        **
*************************************************/
reg    [09:0]    cs, ns        ;
reg    [03:0]    wait_cnt      ;
reg    [03:0]    wait_cnt_cmp;
reg              dir_control ;
/*****************************************************************
-- SRAM Sequence COM FSM
*****************************************************************/
always @( * ) begin
   case ( cs )
      IDLE          : if (write_valid) begin
                        ns = WRITE_OE;
                      end
                      else if (read_valid) begin
                        ns = READ_WE_CS;
                      end
                      else begin
                        ns = IDLE;
                      end

/*************************************************************/
      WRITE_OE      : if (wait_cmp) begin
```

```
                    ns = WRITE_WE_CS;
                end
                    else begin
                        ns = WRITE_OE;
                    end
/*************************************************************/
        WRITE_WE_CS : if (wait_cmp) begin
                    ns = WRITE_DATA;
                end
                    else begin
                        ns = WRITE_WE_CS;
                    end
/*************************************************************/
        WRITE_DATA   : if (wait_cmp) begin
                    ns = WRITE_WE_END;
                end
                    else begin
                        ns = WRITE_DATA;
                    end
/*************************************************************/
        WRITE_WE_END: if (wait_cmp) begin
                    ns = IDLE;
                end
                    else begin
```

```
                        ns = WRITE_WE_END;

                    end
/*************************************************************/

        READ_WE_CS   : if (wait_cmp) begin

                        ns = READ_OE;

                    end

                    else begin

                        ns = READ_WE_CS;

                    end
/*************************************************************/

        READ_OE      : if (wait_cmp) begin

                        ns = READ_OE_END;

                    end

                    else begin

                        ns = READ_OE;

                    end
/*************************************************************/

        READ_OE_END : if (wait_cmp) begin

                        ns = READ_DATA;

                    end

                    else begin

                        ns = READ_OE_END;

                    end
/*************************************************************/
```

```
        READ_DATA     : if (wait_cmp) begin
                            ns = READ_DATA_SP;
                        end
                        else begin
                            ns = READ_DATA;
                        end
/*************************************************************/
        READ_DATA_SP: if (wait_cmp) begin
                            ns = IDLE;
                        end
                        else begin
                            ns = READ_DATA_SP;
                        end
/*************************************************************/
        default       : ns = IDLE;
    endcase
end
/*****************************************************************
// SRAM Sequence SEQ FSM
*****************************************************************/
always @(posedge clk or negedge rstb) begin
    if (!rstb) begin
        cs              <= IDLE;
        cs1_b           <= 1'b1;
```

```
        cs2             <= 1'b0;
        oe_b            <= 1'b1;
        we_b            <= 1'b1;
        dir_control     <= 1'b0;
        data_out        <= 8'd0;
        wait_cnt        <= 3'd0;
    end
    else begin
        cs <= ns;
/*************************************************************/
    case ( ns )
     IDLE : begin cs1_b <= 1'b1; cs2 <= 1'b0; oe_b <= 1'b1;
            we_b <= 1'b1; dir_control <= 1'b0; data_out <= data_out; end
     WRITE_OE : begin cs1_b <= 1'b1; cs2 <= 1'b0; oe_b <= 1'b1;
            we_b <= 1'b1; dir_control <= 1'b0; data_out <= data_out; end
     WRITE_WE_CS : begin cs1_b <= 1'b0; cs2 <= 1'b1; oe_b <= 1'b1;
            we_b <= 1'b0; dir_control <= 1'b0; data_out <= data_out; end
     WRITE_DATA  : begin cs1_b <= 1'b0; cs2 <= 1'b1; oe_b <= 1'b1;
            we_b <= 1'b0; dir_control <= 1'b1; data_out <= data_out; end
     WRITE_WE_END: begin cs1_b <= 1'b1; cs2 <= 1'b0; oe_b <= 1'b1;
            we_b <= 1'b1; dir_control <= 1'b0; data_out <= data_out; end
     READ_WE_CS : begin cs1_b <= 1'b0; cs2 <= 1'b1; oe_b <= 1'b1;
            we_b <= 1'b1; dir_control <= 1'b0; data_out <= data_out; end
     READ_OE    : begin cs1_b <= 1'b0; cs2 <= 1'b1; oe_b <= 1'b0;
            we_b <= 1'b1; dir_control <= 1'b0; data_out <= data_out; end
    READ_OE_END : begin cs1_b <= 1'b1; cs2 <= 1'b0; oe_b <= 1'b1;
```

```
                we_b <= 1'b1; dir_control <= 1'b0; data_out <= data_out; end

    READ_DATA   : begin cs1_b <= 1'b1; cs2 <= 1'b0; oe_b <= 1'b1;
                  we_b <= 1'b1; dir_control <= 1'b0; data_out <= sram_d;   end

    READ_DATA_SP: begin cs1_b <= 1'b1; cs2 <= 1'b0; oe_b <= 1'b1;
                  we_b <= 1'b1; dir_control <= 1'b0; data_out <= sram_d;    end

    default     : begin cs1_b <= 1'b1; cs2 <= 1'b0; oe_b <= 1'b1;
                  we_b <= 1'b1; dir_control <= 1'b0; data_out <= data_out; end

    endcase

/*************************************************************/

      if ((cs != ns) || (ns == IDLE)) begin

          wait_cnt      <= 3'd0;

        end

        else begin

          wait_cnt      <= wait_cnt + 1;

        end

    end

end

/*************************************************************/

assign sram_a       = addr_in;

assign sram_d       = (dir_control == 1'b0) ? 8'hZZ   : data_in;

assign write_done = ((cs == WRITE_WE_END) && (ns == IDLE) &&
(wait_cmp == 1'b1));

assign read_done  = ((cs == READ_DATA_SP) && (ns == IDLE) &&
(wait_cmp == 1'b1));

/*************************************************************
```

```
-- SRAM Wait Time
//////////////////////////////////////////////////////////////////////////////
always @(posedge clk or negedge rstb) begin
    if (!rstb) begin
        wait_cnt_cmp    <= 3'd0;
    end
    else begin
        case ( ns )
            IDLE            : wait_cnt_cmp <= 3'd0;
            WRITE_OE        : wait_cnt_cmp <= 3'd1;
            WRITE_WE_CS     : wait_cnt_cmp <= 3'd2;
            WRITE_DATA      : wait_cnt_cmp <= 3'd3;
            WRITE_WE_END    : wait_cnt_cmp <= 3'd1;
            READ_WE_CS      : wait_cnt_cmp <= 3'd1;
            READ_OE         : wait_cnt_cmp <= 3'd4;
            READ_OE_END     : wait_cnt_cmp <= 3'd1;
            READ_DATA       : wait_cnt_cmp <= 3'd1;
            READ_DATA_SP    : wait_cnt_cmp <= 3'd1;
            default         : wait_cnt_cmp <= 3'd0;
        endcase
    end
end
/////////////////////////////////////////////////////////////////////////////
```

```
assign wait_cmp     = (wait_cnt == (wait_cnt_cmp - 1));

endmodule
```

2.2.7 sram_byte_ctrl.v : Design 요약 설명 (코드 내 주석)

```
/****************************************************
** Input Signal Define*********************************
****************************************************/
/****************************************************
** Output Signal Define ******************************
****************************************************/
```

▶ **Input, Output Signal Define**

- sram_byte_ctrl.v 디자인의 입출력을 선언한 부분

```
/****************************************************
** Parameter Definition                            **
****************************************************/
```

▶ **Input, Output Signal Define**

- sram_byte_ctrl.v 디자인이 SRAM Memory 를 컨트롤하기 위한 블록으로, FSM (Finite State Machine) 으로 구성하여 동작시켰다. 여기서 FSM 의 IDLE, WRITE_OE, WRITE_WE_CS, WRITE_DATA, WRITE_WE_END, READ_WE_CS, READ_OE, READ_OE_END, READ_DATA, READ_DATA_SP 의 상태를 파라미터로 정의한 로직이다.
차후, 이 부분은 "SRAM Sequence COM FSM" 디자인 부분에서 FSM 의 상태 값으로 사용 된다.

```
/*************************************************
** Wire Definition ***********************************
*************************************************/
/*************************************************
** Reg Definition ************************************
*************************************************/
```

▶ Wire, Reg Definition

- SRAM을 동작하기 위한 시그널과 SRAM Memory 의 상태를 나타내기 위한 reg 가 정의되어 있는 부분

```
/*****************************************************************
-- SRAM Sequence COM FSM
*****************************************************************/
```

▶ SRAM Sequence COM FSM

- FSM 디자인으로, cs 값에 따라 IDLE 상태에서 데이터를 읽어오는 READ_DATA_SP 상태까지로 동작하게 하는 디자인이다.

```
/*****************************************************************
// SRAM Sequence SEQ FSM
*****************************************************************/
```

▶ **SRAM Sequence SEQ FSM**

- ns 값에 따라 IDLE 상태에서 READ_DATA_SP 상태까지 SRAM 을 컨트롤하기 위해 동작해야 하는 cs1_b, cs2, oe_b, we_b, dir_control, data_out 의 값을 정의하는 디자인이다.

```
/*****************************************************************
-- SRAM Wait Time
*****************************************************************/
```

▶ **SRAM Wait Time**

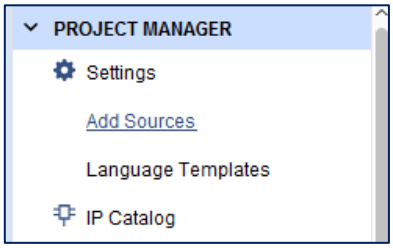
- SRAM 을 Test 하기 위해 입력 받는 Clock 은 100MHz 이다.
빠른 Clock 을 사용하므로 **2.2.2 SRAM 동작 이해** 부분의 타이밍도에 나와 있는 타이밍을 고려하여, 각 상태의 Wait Timing 을 정의한 블록 디자인 이다.

✳ SRAM 동작의 추가 구성 방법

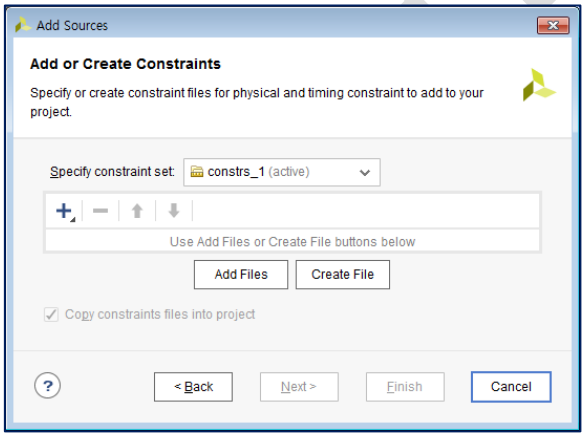
- 본 자료의 디자인은 SRAM 에 Write / Read 를 하여 확인하는 디자인이다. 여기에서 저장하는 데이터를 RGB 데이터형태에 맞춰 저장하고, 이것을 읽어 들여 RGB 포트로 출력하면 SRAM 을 이용하여 RGB 포트를 사용하는 TFT-LCD 의 RGB 컬러 영상 출력이 가능하다.

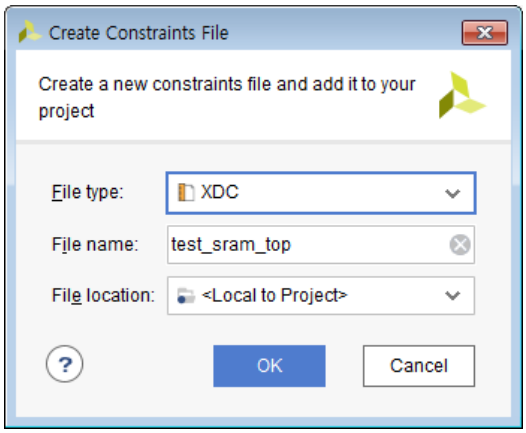
2.2.8 Vivado Design Pin 정보 (XDC) 입력 작업

1) 하기의 그림과 같이 PROJECT MANAGER 창에서 Add Sources 클릭
➔ Add or create constraints 선택 후 Next 클릭

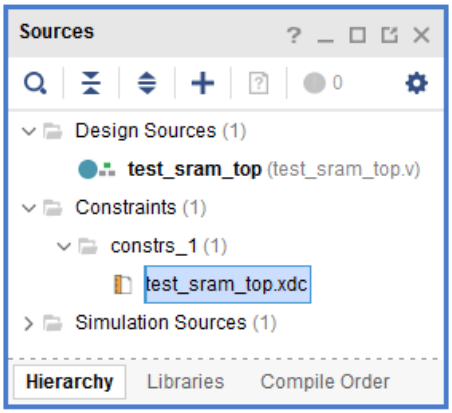


2) Add or Create Constraints 창에서 Create File 클릭
➔ File name 란에 "test_sram_top" 라고 입력한 후 OK 클릭
➔ 그리고 다시 Add or Create Constraints 창이 나타나면 Finish 클릭





3) 하기의 그림과 같이 Sources 창에서 test_sram_top.xdc 가 나타난 것을 확인할 수 있으며, test_sram_top.xdc 파일을 더블 클릭하면 Vivado 우측창에 XDC 를 편집할 수 있는 에디터 창이 나타난다.
이곳에 하기의 핀 정보를 입력 한다.



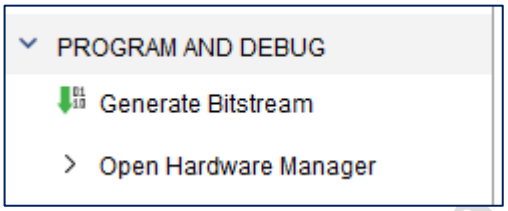

```
set_property -dict {PACKAGE_PIN R4 IOSTANDARD LVCMOS33} [get_ports clk]

set_property -dict {PACKAGE_PIN U7 IOSTANDARD LVCMOS33} [get_ports rstb]

set_property -dict {PACKAGE_PIN Y18 IOSTANDARD LVCMOS33} [get_ports test_end]

set_property -dict {PACKAGE_PIN AB18 IOSTANDARD LVCMOS33} [get_ports test_error]

set_property -dict {PACKAGE_PIN J4   IOSTANDARD LVCMOS15} [get_ports test_start]

set_property -dict {PACKAGE_PIN P15 IOSTANDARD LVCMOS33}   [get_ports oe_b]

set_property -dict {PACKAGE_PIN P14 IOSTANDARD LVCMOS33}   [get_ports we_b]

set_property -dict {PACKAGE_PIN T3 IOSTANDARD LVCMOS33}   [get_ports cs1_b]

set_property -dict {PACKAGE_PIN T1 IOSTANDARD LVCMOS33}   [get_ports cs2]

set_property -dict {PACKAGE_PIN V2 IOSTANDARD LVCMOS33}   [get_ports {sram_d[0]}]

set_property -dict {PACKAGE_PIN R3 IOSTANDARD LVCMOS33}   [get_ports {sram_d[1]}]

set_property -dict {PACKAGE_PIN R2 IOSTANDARD LVCMOS33}   [get_ports {sram_d[2]}]

set_property -dict {PACKAGE_PIN W2 IOSTANDARD LVCMOS33}   [get_ports {sram_d[3]}]
```

```
set_property -dict {PACKAGE_PIN Y2 IOSTANDARD LVCMOS33}   [get_ports {sram_d[4]}]
set_property -dict {PACKAGE_PIN W1 IOSTANDARD LVCMOS33}   [get_ports {sram_d[5]}]
set_property -dict {PACKAGE_PIN Y1 IOSTANDARD LVCMOS33}   [get_ports {sram_d[6]}]
set_property -dict {PACKAGE_PIN U3 IOSTANDARD LVCMOS33}   [get_ports {sram_d[7]}]
set_property -dict {PACKAGE_PIN V5 IOSTANDARD LVCMOS33}   [get_ports {sram_a[0]}]
set_property -dict {PACKAGE_PIN R6 IOSTANDARD LVCMOS33}   [get_ports {sram_a[1]}]
set_property -dict {PACKAGE_PIN T6 IOSTANDARD LVCMOS33}   [get_ports {sram_a[2]}]
set_property -dict {PACKAGE_PIN Y6 IOSTANDARD LVCMOS33}   [get_ports {sram_a[3]}]
set_property -dict {PACKAGE_PIN AA6 IOSTANDARD LVCMOS33} [get_ports {sram_a[4]}]
set_property -dict {PACKAGE_PIN V7 IOSTANDARD LVCMOS33}   [get_ports {sram_a[5]}]
set_property -dict {PACKAGE_PIN W7 IOSTANDARD LVCMOS33}   [get_ports {sram_a[6]}]
set_property -dict {PACKAGE_PIN AB7 IOSTANDARD LVCMOS33} [get_ports {sram_a[7]}]
set_property -dict {PACKAGE_PIN AB6 IOSTANDARD LVCMOS33} [get_ports {sram_a[8]}]
set_property -dict {PACKAGE_PIN V9 IOSTANDARD LVCMOS33}   [get_ports {sram_a[9]}]
set_property -dict {PACKAGE_PIN V8 IOSTANDARD LVCMOS33}   [get_ports {sram_a[10]}]
set_property -dict {PACKAGE_PIN AA8 IOSTANDARD LVCMOS33} [get_ports {sram_a[11]}]
set_property -dict {PACKAGE_PIN AB8 IOSTANDARD LVCMOS33} [get_ports {sram_a[12]}]
set_property -dict {PACKAGE_PIN Y8 IOSTANDARD LVCMOS33}   [get_ports {sram_a[13]}]
set_property -dict {PACKAGE_PIN Y7 IOSTANDARD LVCMOS33}   [get_ports {sram_a[14]}]
set_property -dict {PACKAGE_PIN W9 IOSTANDARD LVCMOS33}   [get_ports {sram_a[15]}]
set_property -dict {PACKAGE_PIN Y9 IOSTANDARD LVCMOS33}   [get_ports {sram_a[16]}]
```

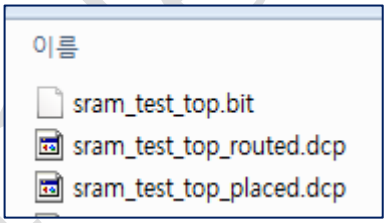
2.2.9 Vivado 디자인 다운로드 파일(bit) 생성

1) 지금까지의 작업으로 디자인 소스와 핀 정보(XDC) 파일 생성이 완료 되었으며, FSK III 에 Design Download 를 위해 bit 파일을 생성한다.

하기 그림과 같이 PROJECT MANAGER 창에 있는 Generate Bitstream 클릭 후 다른 창이 나타나면 OK를 눌러 진행한다.

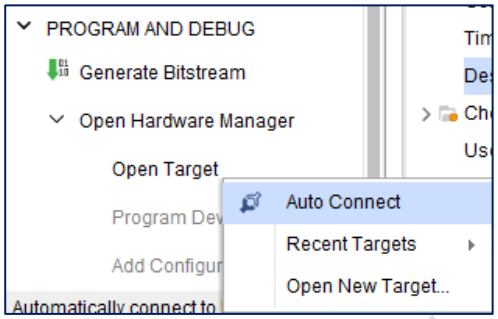


작업이 완료 되면 프로젝트 폴더에서 Bit 파일이 생성됨을 확인할 수 있다. (프로젝트 폴더 → test_sram.runs 폴더 → impl_1 폴더 → test_sram.bit)

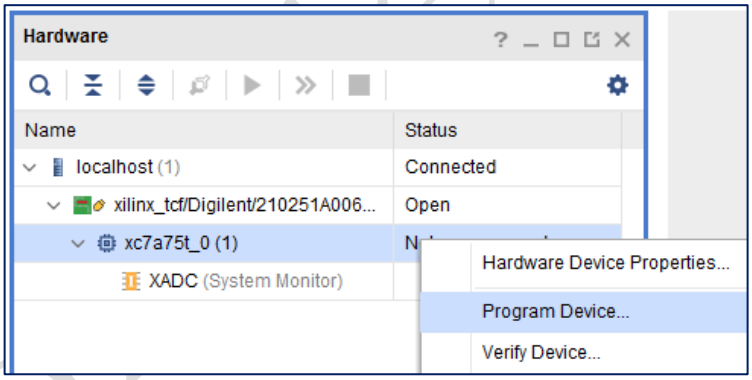


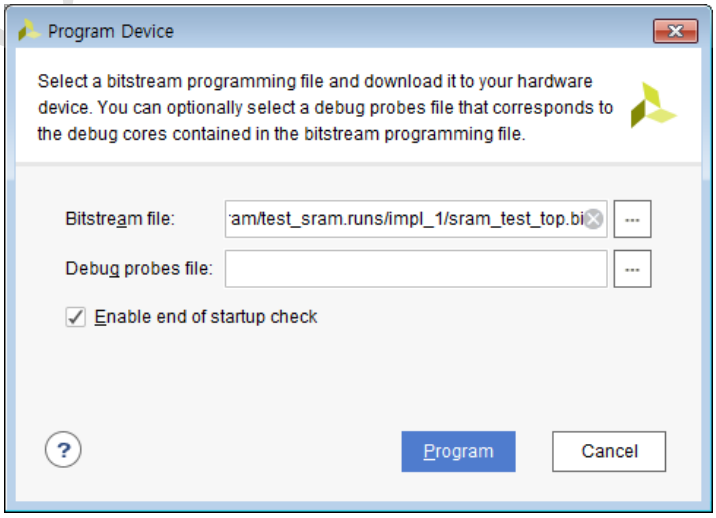
2.2.10 FSK III 에 Design (bit file) Download 진행

1) FSK III 보드를 JTAG Cable (Micro 5pin)을 통해 PC와 연결하고 전원을 켠다.
하기 그림과 같이 PROJECT MANAGER ➔ PROGRAM AND DEBUG
➔ Open Hardware Manager ➔ Open Target ➔ Auto Connect 를 선택



2) Hardware 탭에 xc7a75t_0(1) 이 나타나면 우 클릭 후 Program Device 선택
➔ Program Device 창에서 Program 을 클릭하여 FPGA (FSK III) 에
다운로드를 진행 한다.





- FPGA 에 bit 파일을 다운로드 한 후, Dip SW1 이 좌측 그림과 같이 OFF 되어 있으면 LD1 이 꺼져 있는 것을 확인할 수 있고, 우측 그림과 같이 Dip SW1 이 On 이면 LD1 이 켜지는 것을 확인할 수 있다.

감사합니다.

Revision History

Ver	Date	Revision
1.0	2020-05-11	Initial Document Release.