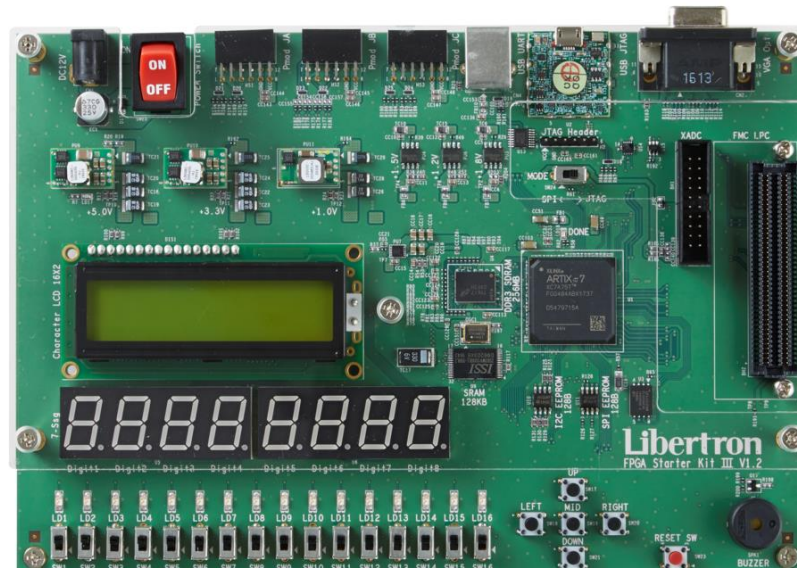


FSK III Block Memory Tutorial

Authors : 기술지원팀 김민석 팀장



Libertron Co., Ltd

본 설명서를 (주)리버트론의 허락 없이 복제하는 행위는 금지되어 있습니다.

1. 개요

- 이 문서에서는 FPGA 내의 Block Memory 의 동작을 이해하기 위해, Block Memory 의 개념을 익히고 Vivado Tool 을 통한 Simulation 을 통해 동작을 검증한다.
- Memory 를 구성하는 방법과 Memory 연결 핀에 대한 입력 조건을 확인하여 차후 프로젝트에 적용함을 목표로 한다.

2. 상세 설명

2.1 준비 사항 및 테스트 환경

2.1.1 준비사항

- FPGA 내의 BRAM (Block Memory) 에 대한 내용이므로 FSK III 보드는 필요 없음
- 본 자료를 통해서 차 후 User Project 에 적용 가능 함

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 의 Target Device 인 XC7A75T-1FFG484 Device 를 기반으로, IP Catalog 에서 생성한 BRAM 블록과 그 IP를 연결하기 위한 Wrapper 디자인으로 구성 되었다.
- 메모리에 연결된 입력 조건들을 테스트 벤치 (Testbench) 에 작성하여 Simulation 으로 동작 확인을 진행하며, Memory 를 구성하는 방법과 Memory 연결 핀에 대한 입력 조건이 어떻게 되는지 확인하여, 추후 자신이 사용하고자 하는 프로젝트 적용 함을 목표로 한다..

2.2.2 FPGA BRAM 동작 이해

- 본 자료에서의 FPGA BRAM 디자인은 FPGA Dedicate 되어 있는 Single Port

BRAM을 생성하여 HDL 로 상위 디자인과 연결 시켰다.

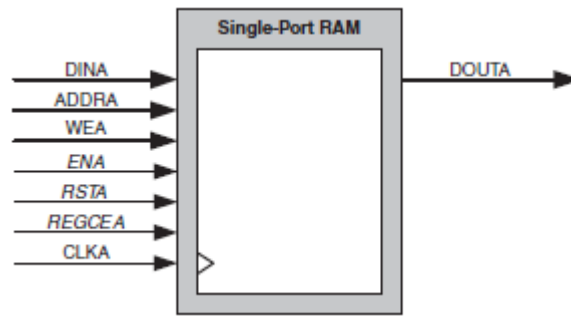
또한 Single Port BRAM 의 동작을 확인하기 위해 Testbench를 작성하여 Single Port BRAM 이 어떻게 동작하는지 확인한다.

- BRAM 의 이해를 위해 Xilinx 가 제공하는 pg058-blk-mem-gen.pdf 문서를 다운로드 하여 참조 하도록 한다.

- Xilinx FPGA 내의 BRAM 은 사용자의 목적에 따라 다양하게 구성하여 사용이 가능하다. Xilinx Vivado SW 환경에서 IP Catalog → Basic Elements → Memory Elements → Block Memory Generator (or Distribute Memory Generator : LUT를 활용하는 메모리) 를 실행하면 Memory 를 구성하는 창이 나타나며, 하기와 같은 Memory Type 을 선택해 구성이 가능하다.

FPGA 내의 Block Memory Type	
Single Port Memory	하나의 a port로 clk, addr, data를 받아 저장하여 출력 시키는 Memory
Two Port Memory	두 개의 컨트롤 신호 clk a/b, addr a/b 로, 하나의 a port data를 받아 저장하여 하나의 출력 port로 출력시키는 Memory
True Dual Port Memory	두 개의 컨트롤 신호 clk a/b, addr a/b 및 data a/b를 받아 저장하여 각각의 출력 a/b port로 출력 시키는 Memory
FIFO (First Input First Output)	처음 들어온 데이터가 처음으로 출력되는 Memory
Single Port ROM	Coe 파일 형태로 미리 데이터를 저장하여 하나의 a port로 컨트롤 및 출력하여 사용하는 Memory
Two Port ROM	Coe 파일 형태로 미리 데이터를 저장하여 두개의 a/b port로 각각 컨트롤 및 출력하여 사용하는 Memory

- pg058-blk-mem-gen.pdf 자료의 p41 에서 하기 그림과 같은 Single Port Memory의 인터페이스 내용을 확인할 수 있다.



- 메모리의 기본적인 동작은 저장 하는 기능이다. 동작을 하기 위해, 상기그림과 같은 신호가 필요하며, 그에 대한 동작설명과 타이밍은 하기 그림에서 확인 할 수 있다. (하기 내용은 pg058-blk-mem-gen.pdf 파일 p46에 있는 Write First Mode 에 상세히 표기되어 있음.)

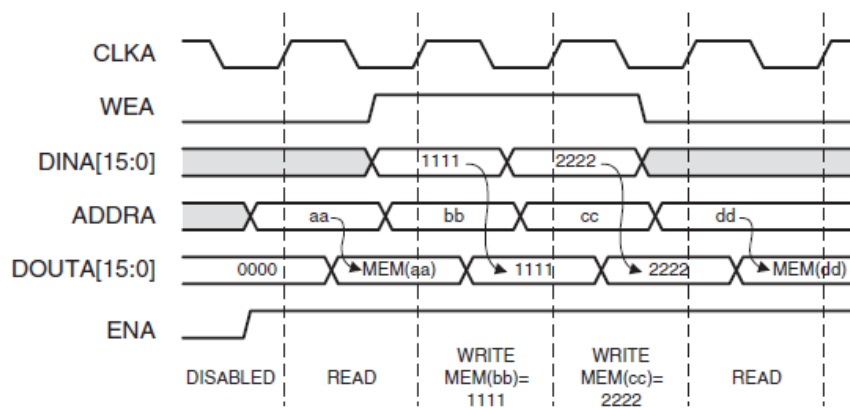
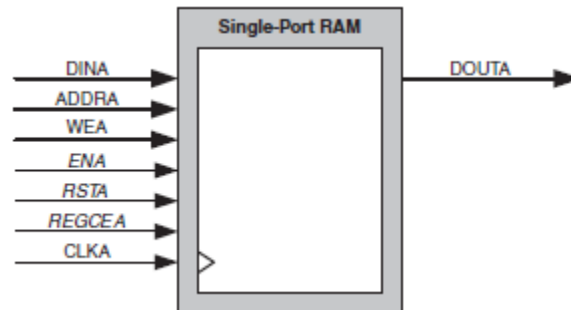


Figure 3-9: Write First Mode Example

BRAM 동작 Signal	
CLKA	Memory Write 동작에 사용되는 Clock
ADDRA	Memory 에 저장할 어드레스
RSTA	Active High (Low 일 때 Memory 에 저장 가능)
ENA	Active High (High 일 때 Memory 에 저장/출력 가능)
WEA	Active High (High 일 때 Memory 에 데이터 출력)
DINA	Memory 에 저장하기 위한 Data
DOUTA	Memory 에서 출력되는 Data

2.2.3 디자인 동작 블록도

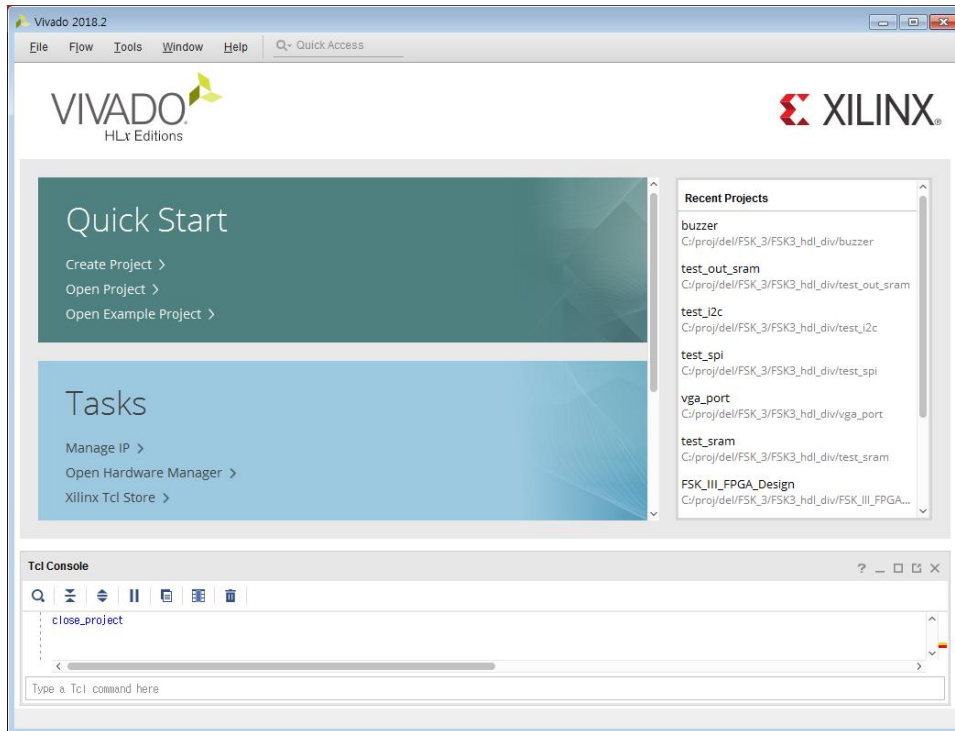


- 본 디자인 작업은 IP Catalog 에서 생성한 BRAM을 그대로 이용하고 Wrapper만 연결하여 사용하므로, 상기 그림이 그 자체로 블록도가 된다.

2.2.4 Vivado 디자인 구성

1) Vivado New Project 실행

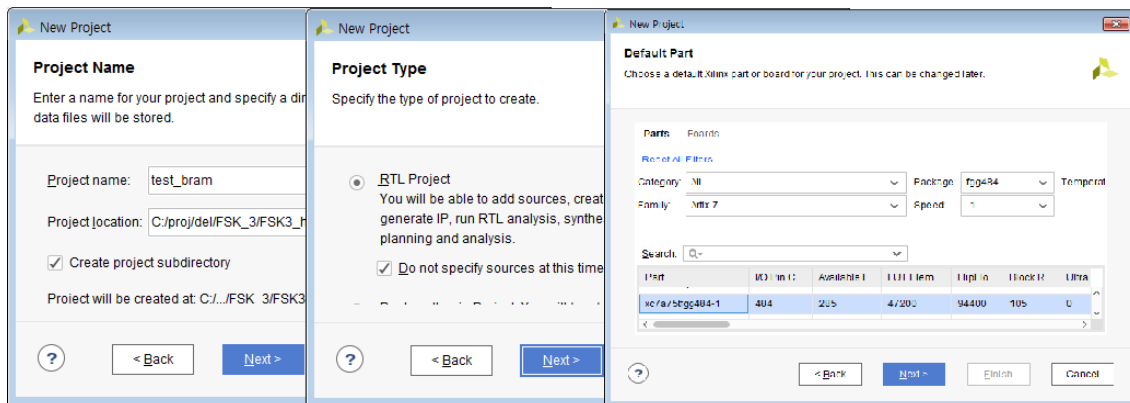
* Project 작업 경로 및 폴더에 특수문자 및 한글의 인식이 안되므로, 영어만 사용할 것



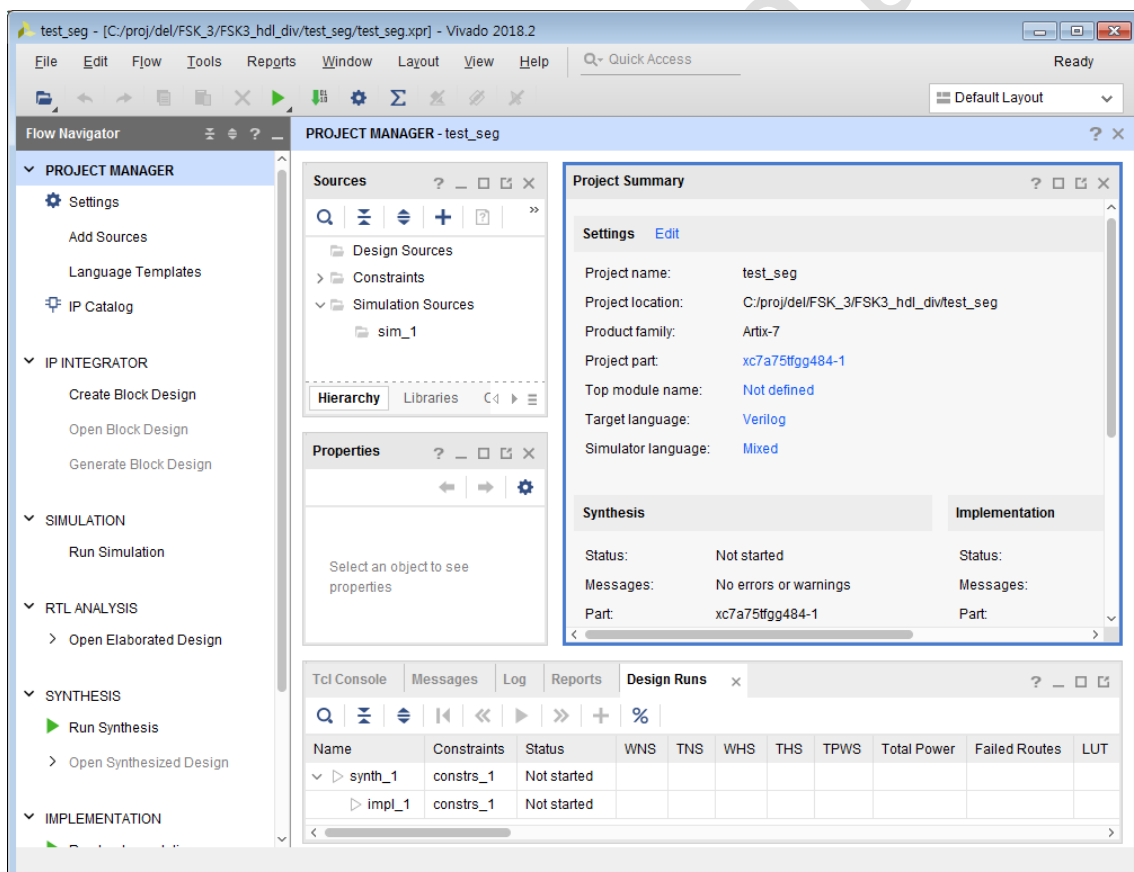
2) 상기 그림에서 다음과 같이 진행 한다.

- Create Project → Create a New Vivado Project 에서 Next 클릭
- Project Name 란에 "test_bram" 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

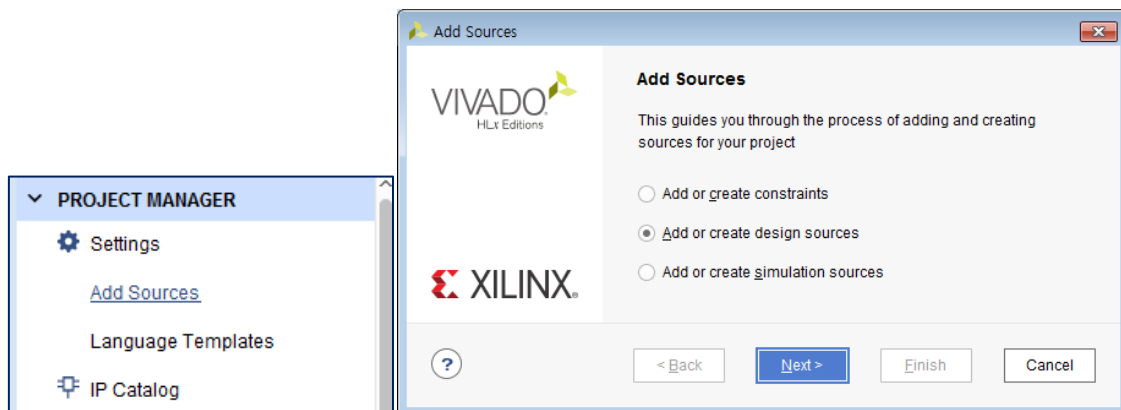


- 3) 상기 작업 후 New Project Summary 창이 나타나면 Finish 클릭
작업이 완료 되면 하기와 같이 Vivado 초기 프로젝트 화면이 나타난다.

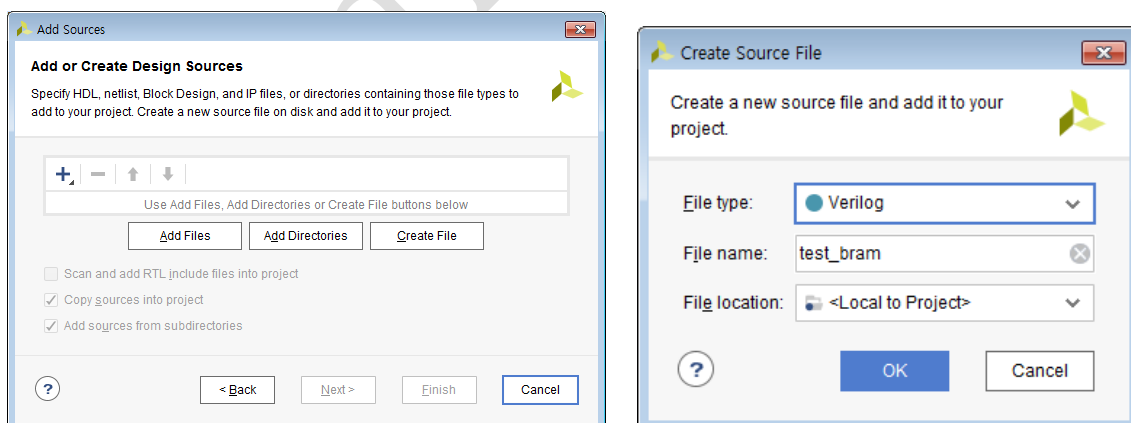


2.2.5 Vivado New Design Create

- 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_bram 입력 후 Ok 버튼 클릭 및 Finish 클릭
(User 가 임의의 영어로 이름을 지정해도 됨)



- 3) Define Module 창에서 하기와 같이 입력하고 OK 클릭

Port Name	Direction	Bus	MSB	LSB
clk	Input			
rstb	Input			
ena	Input			
wea	Input			
addra	Input	✓	8	0
dina	Input	✓	15	0
douta	output	✓	15	0
rsta_busy	output			

Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked
Ports with blank names will not be written.

Module Definition

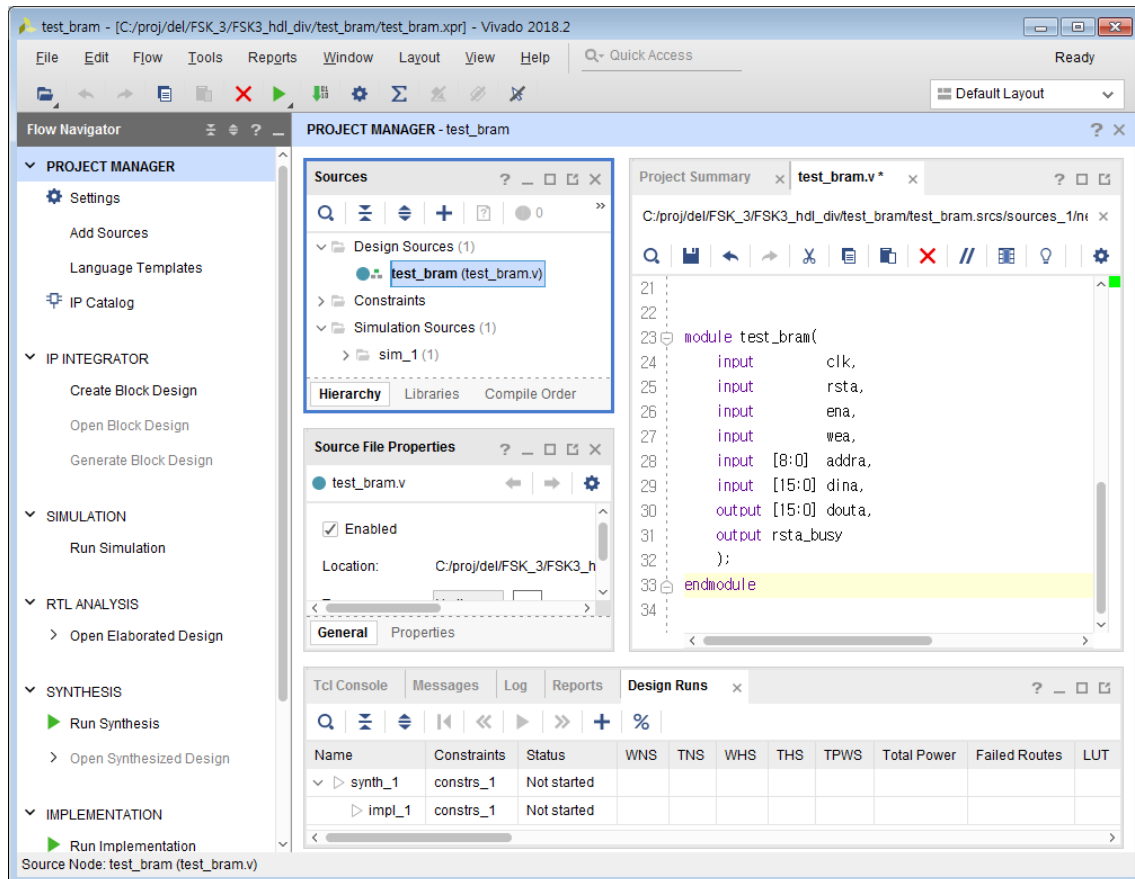
Module name: test_bram

I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
clk	input	<input type="checkbox"/>	0	0
rstb	input	<input type="checkbox"/>	0	0
ena	input	<input type="checkbox"/>	0	0
wea	input	<input type="checkbox"/>	0	0
addra	input	<input checked="" type="checkbox"/>	8	0
dina	input	<input checked="" type="checkbox"/>	15	0
douta	outp...	<input checked="" type="checkbox"/>	15	0
rsta_busy	outp...	<input type="checkbox"/>	0	0

? OK Cancel

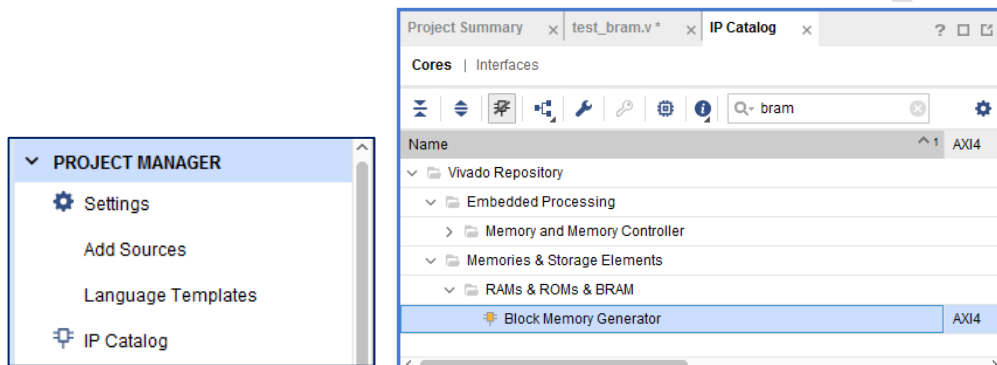
* 작업이 완료 되면 하기와 같이 Vivado 창이 나타난다.



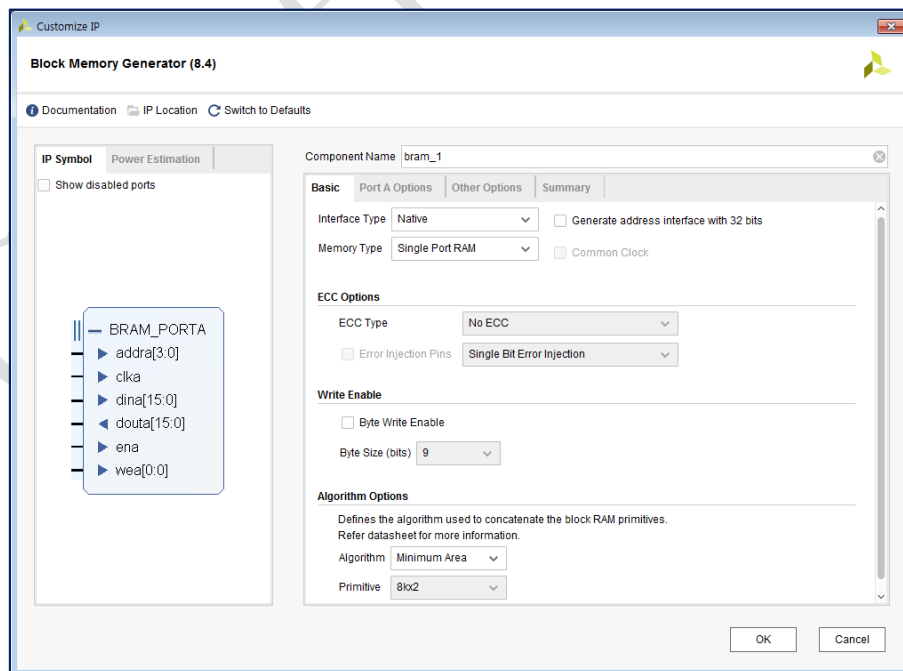
2.2.6 IP Catalog 에서 BRAM 생성 작업

- 1) 하기의 좌측 그림과 같이 Vivado 의 PROJECT MANAGER 탭에서 IP Catalog 를 클릭한다.

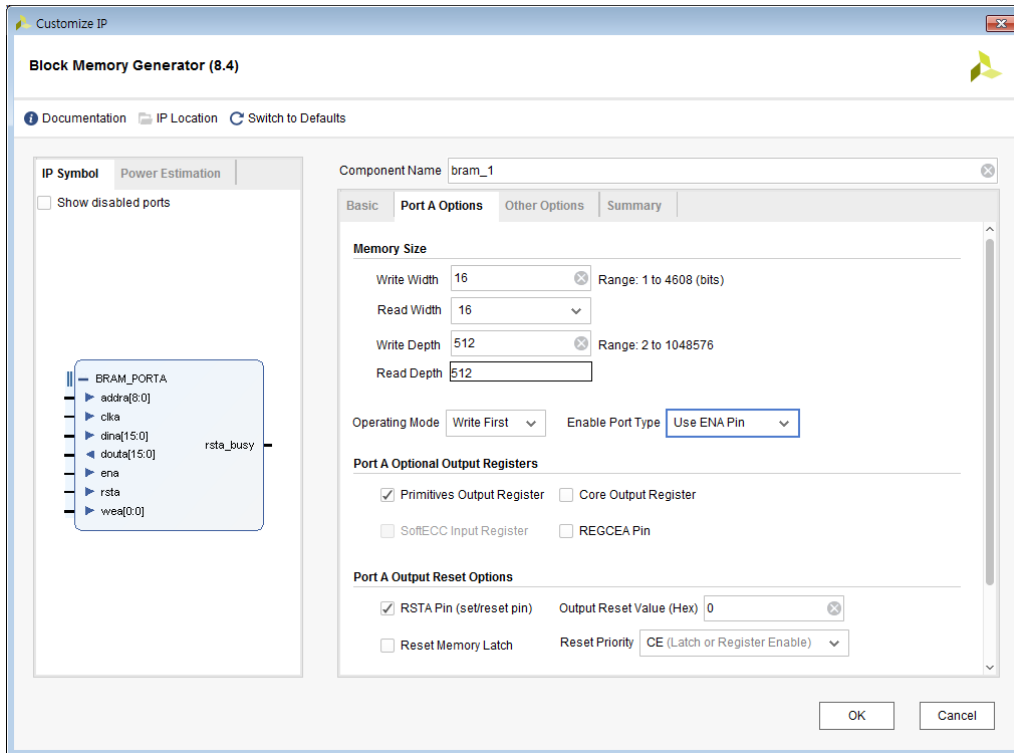
하기 우측그림과 같이 IP Catalog 창이 나타나면 검색 탭에서 bram 입력
→ Vivado Repository → Memories & Storage Elements
→ RAMs & ROMs & BRAM → Block Memory Generator 를 선택



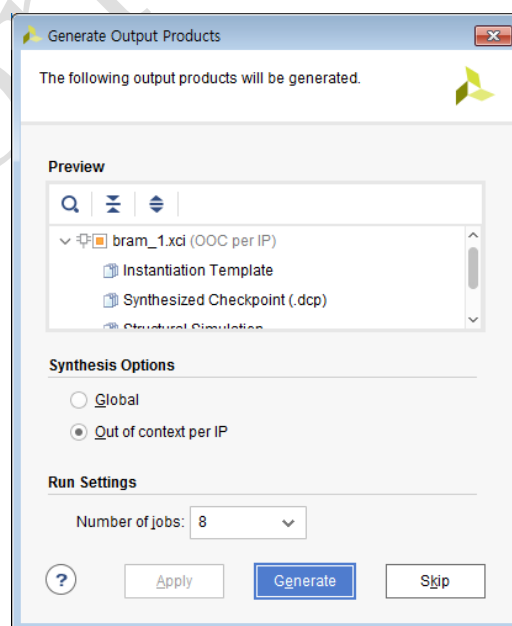
- 2) 하기 창이 나타나면, Memory Type 에서 Single Port RAM 을 선택하고 Port A Options 를 클릭한다.



- 3) Port A Options 창이 나타나면 하기의 그림과 같이 Write Depth 값을 512로 입력하고, RSTA Pin 란에 체크한다.
나머지는 Default 상태에서 OK 를 클릭



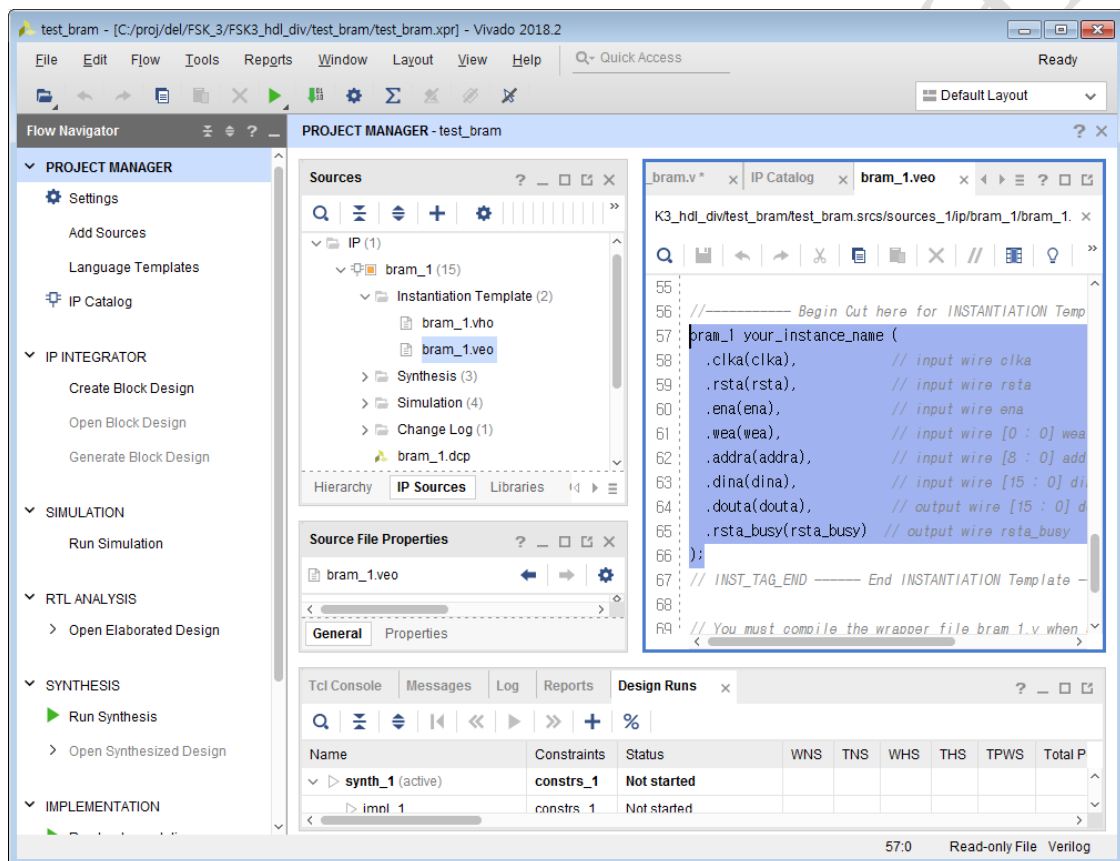
- 4) 하기 창이 나타나면 Default 상태에서 Generate 를 클릭한다.
지금까지의 작업이 완료되면 bram_1 의 메모리 블록이 구성된다.



2.2.7 Vivado 디자인 입력 작업

- 이전까지의 작업으로 bram_1 에 대한 Memory IP 가 구성 되었다.
이것을 이전에 구성한 test_bram.v 에 연결하여 사용 한다.
- 하기 그림과 같이 Sources 창에 있는 IP Sources 탭을 선택
→ IP → bram_1 → Instantiation Template → bram_1.veo 더블 클릭
→ 우측에 있는 디자인 에디터 창에 bram_1 IP에 대한 Instance가 나타난다.

하기의 오른쪽창에 드래그 하여 복사하고, test_bram.v 디자인에 붙여 넣는다.



→ 하기의 내용과 같이 디자인을 입력한다.

```
module test_bram(
    input          clk,
```

```
input      rsta,

input      ena,

input      wea,

input  [8:0]  addra,

input  [15:0] dina,

output [15:0] douta,

output rsta_busy );

/*****

** Instance Definition **

*****/

bram_1 u0 (

.clka      (clk),          // input wire clka

.rsta      (rsta),          // input wire rsta

.ena       (ena),          // input wire ena

.wea       (wea),          // input wire [0 : 0] wea

.addra     (addra),        // input wire [3 : 0] addra

.dina      (dina),          // input wire [7 : 0] dina

.douta     (douta),         // output wire [15 : 0] douta

.rsta_busy (rsta_busy      ) ); // output wire rsta_busy

endmodule
```

2.2.8 테스트 벤치 (Testbench) 디자인 작성

```
module tb_test_bram();

reg          clk;

reg          rsta;

reg          ena;

reg  [0 : 0] wea;

reg  [8 : 0] addra;

reg  [15 : 0] dina;

wire [15 : 0] douta;

wire          rsta_busy;

/*****

** Instance Definition                                **

*****/

test_bram u0 (

    .clk      (clk),

    .rsta     (rsta),

    .ena      (ena),

    .wea      (wea),

    .addra    (addra),

    .dina     (dina),

    .douta    (douta),

    .rsta_busy (rsta_busy) );

/*****
```

```

** Instance Definition                                     **

*****/

initial begin

    clk  = 0; rsta = 1; wea = 0; ena = 0; #100;

    rsta = 0; #1000000000;

end

always #10740 ena = ~ena;

always #10740 wea = ~wea;

always #10 clk = ~clk;

/*****/

** addra Address Generation                               **

*****/

always@(posedge clk) begin

    if (rsta) begin

        addra <= 0;

        end

    else begin

        if (addra == 'd512) begin

            addra <= 0;

            end

        else begin

            addra <= addra + 1;

            end

        end

    end

```



```

        end

    end

    /**
    ** dina Data Generation
    **
    */

    always@(posedge clk) begin

        if (rsta) begin

            dina <= 'h0001;

            end

            else begin

                if (dina == 'hFFFF) begin

                    dina <= 'h0001;

                    end

                    else begin

                        dina <= {dina[14:0], dina[0]};

                        end

                    end

                end

            end

        end

    endmodule

```

2.2.9 디자인 안내 (test_bram 디자인 주석 안내)

```
/*  
***** Instance Definition *****  
*/
```

► Instance Definition

- Test_bram.v 디자인과 bram_1 디자인을 연결하는 구조적 모델링 디자인 부분

2.2.10 디자인 안내 (tb_test_bram 디자인 주석 안내)

```
/*  
***** Instance Definition *****  
*/
```

► Instance Definition

- tb_test_bram.v 디자인과 test_bram.v 디자인을 연결하는 구조적 모델링 디자인 부분

```
/*  
** Input initialization & Operation Definition *****  
*/
```

► Input initialization & Operation Definition

- 각 입력 신호의 초기화와 Clock 과 ena, wea 의 컨트롤 신호의 입력조건을 정의한 부분

```
/*  
** addra Address Generation  
**  
***/
```

► addra Address Generation

- BRAM 에 들어가는 Address 즉, addra 에 들어가는 address 를 counter 로직을 이용 및 생성해서 넣어주는 부분. 이 부분은 test_rbram.v 에 넣어서 연결시켜도 같은 동작을 한다. Addra 의 경우 16 bit 이나, 메모리 구성시 512 depth 로 구성하여 Testbench 에서도 512 까지 생성하고 초기화 되도록 하였다.

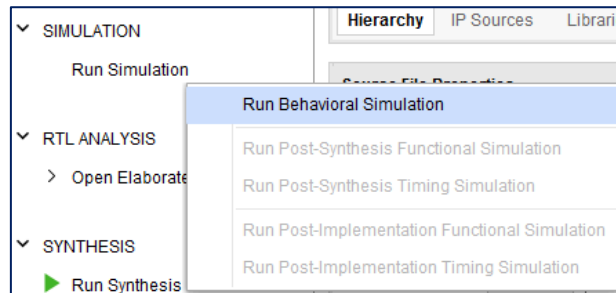
```
/*  
** dina Data Generation  
**  
***/
```



► dina Data Generation

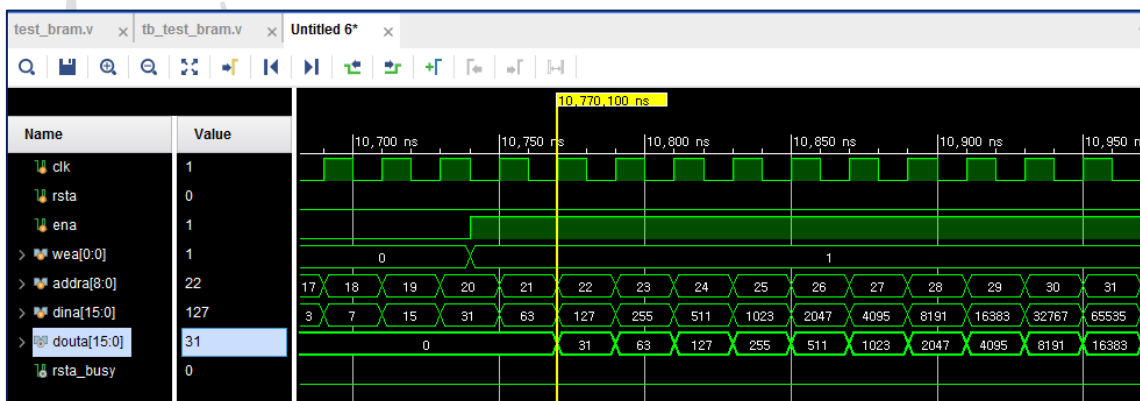
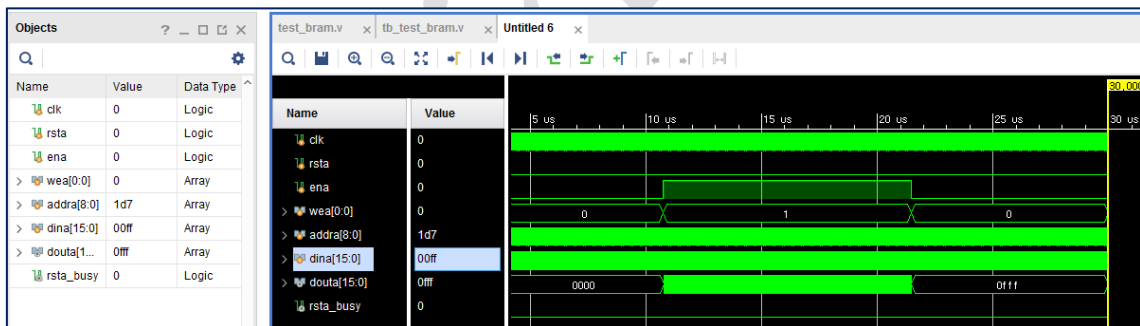
- dina 에 들어갈 데이터를 생성하는 부분. Address 를 생성하는 부분과 다르게 하기 위해, shift 로직을 이용하여 데이터가 생성되도록 하였다.

2.2.11 Simulation 결과 확인

- 1) 하기의 그림과 같이 PROJECT MANAGER 창에서 Simulation → Run Behavioral Simulation 클릭.



- 2) Viavdo 창 상단에 있는  30 us 의 그림과 같이 30us로 입력하고  버튼을 클릭하여 실행한다.
하기의 그림과 같이 dina 로 데이터가 들어가서 ena, wea 조건에 따라 douta 로 데이터가 출력되는 것을 확인할 수 있다.



감사합니다.

Revision History

Ver	Date	Revision
1.0	2020-04-13	Initial Document Release.