

SoC 를 위한 *Peripheral* 설계

Reference : MicroBlaze.v15 [IHIL]

2024-06-20

 Kim.S.W

 대한상공회의소
인력개발원 @IHIL

digilent.com

 XILINX®

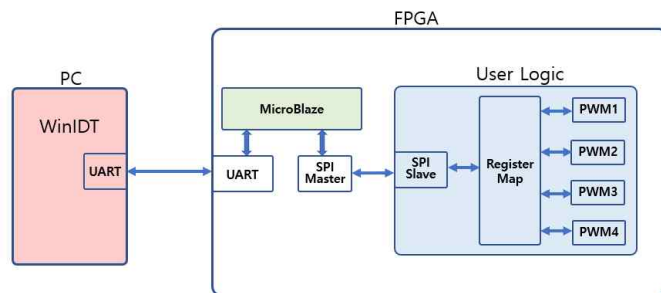
1/75

1/50

Table of Contents

➤ SoC를 위한 Peripheral 설계

1. Xilinx IP
2. Create and Package New IP
3. SPI
 - 1) SPI Master
 - 2) SPI Slave
 - 3) SPI Controller
4. UART
5. AMBA
6. MicroBlaze_Hello World
7. MicroBlaze_LED_Counter
8. MicroBlaze_Peripheral Implementation
9. MicroBlaze_User Logic Interface

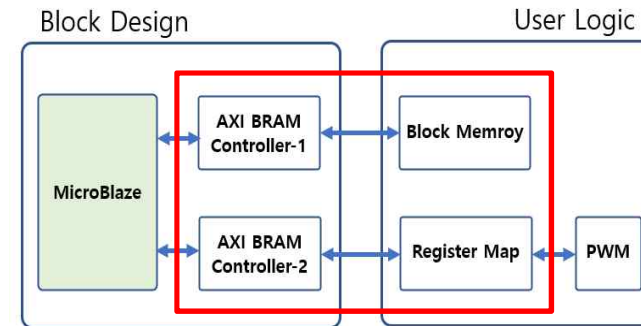


10. SPI_Master_IP(MicroBlaze_User Logic Interface)

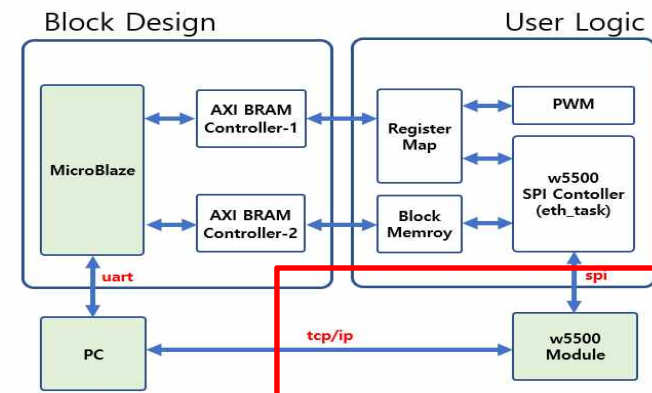
11. TCP_IP Implementation Using W5500

12. MicroBlaze_Block Memory Interface-1

13. MicroBlaze_Block Memory Interface-2



14. w5500 Interface Implementation

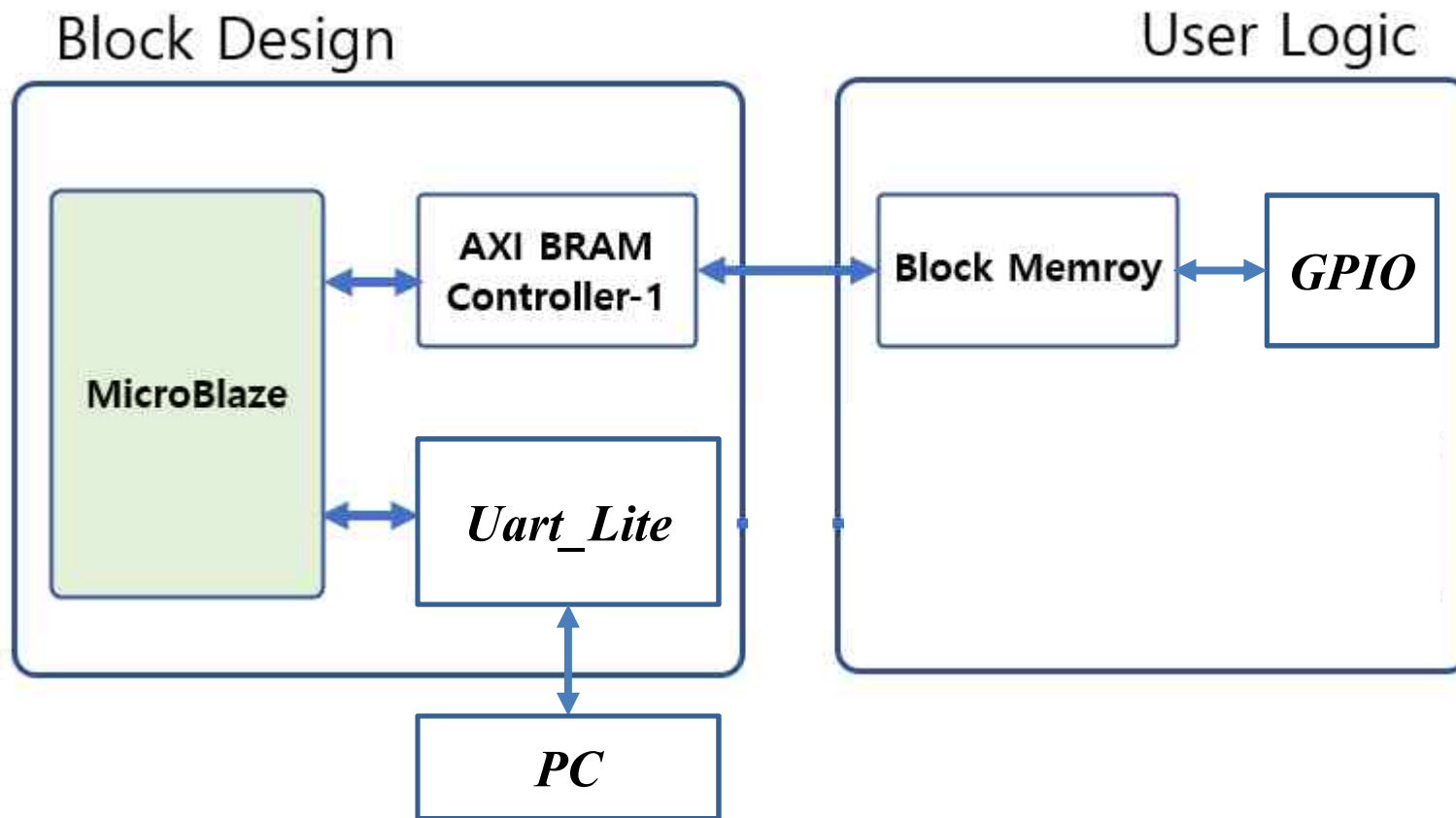


➤ SoC Peripheral RTC Design Project

Block Memory Interface Implementation bram02_ch10.xpr

➤ Block Memory Interface Implementation 실습

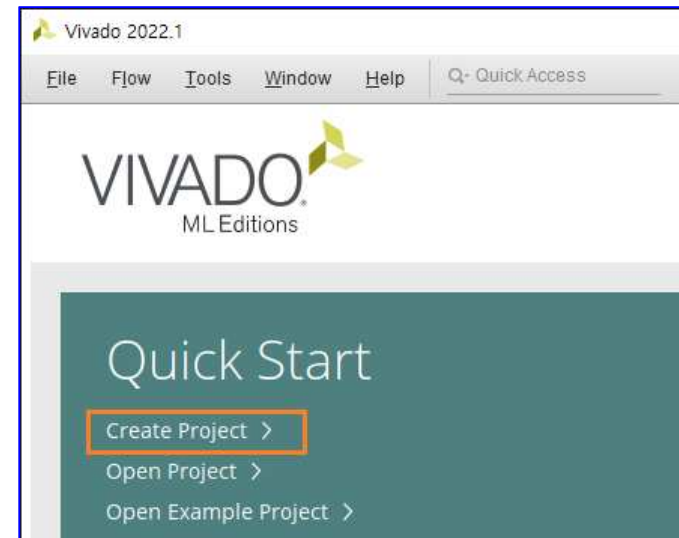
- *Interface Block*과 *Memory Block*을 *Block Design*에서 지원하는 방법을 사용하여 구현
- *Application SW*에서는 구현된 메모리에 데이터를 *read/write* 하는 것을 구현



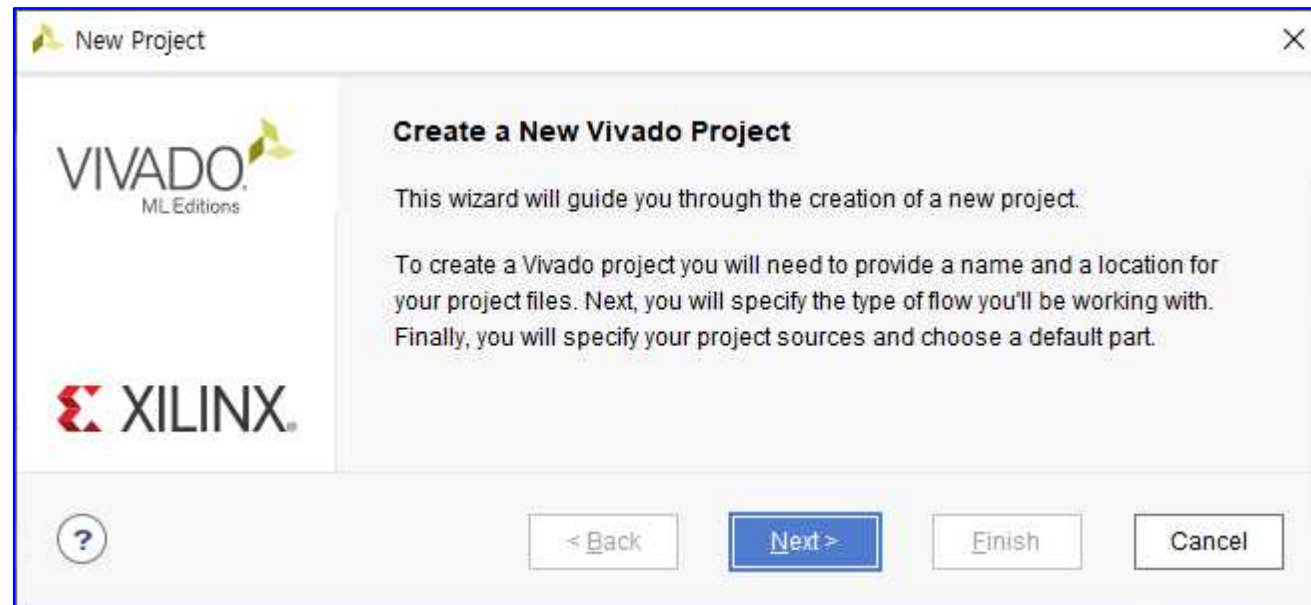
Block Memory Interface Implementation bram02_ch10.xpr

➤ 프로젝트 생성

- vivado 2022.1을 실행하고, Create Project를 클릭



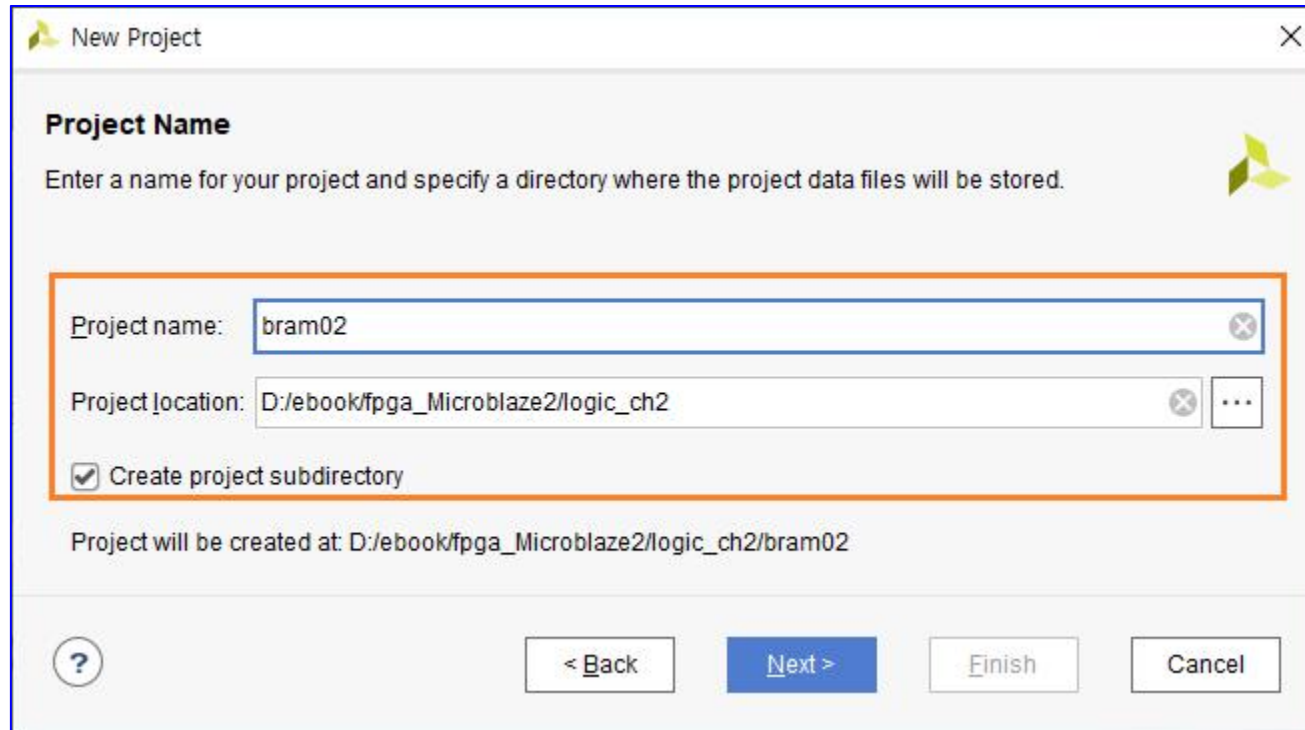
- Next를 클릭



Block Memory Interface Implementation bram02_ch10.xpr

➤ 프로젝트 생성

- 프로젝트 이름을 “**bram02**”로 하고 프로젝트 위치를 설정하고 Next를 클릭



New Project

Project Name
Enter a name for your project and specify a directory where the project data files will be stored.

Project name:

Project location:

☒ Create project subdirectory

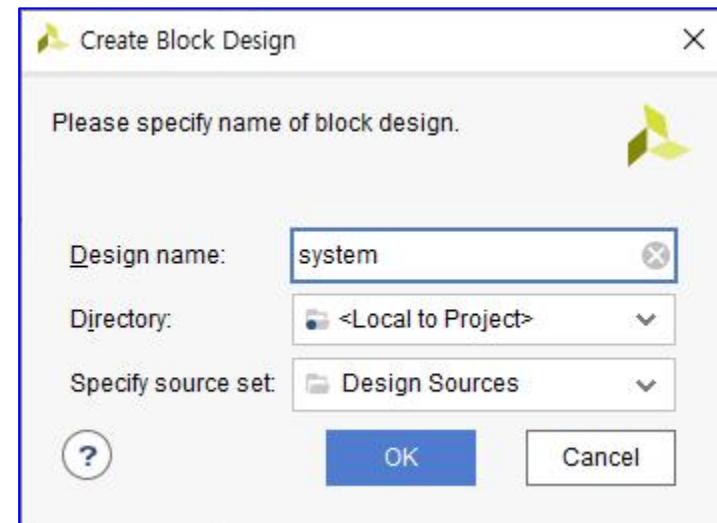
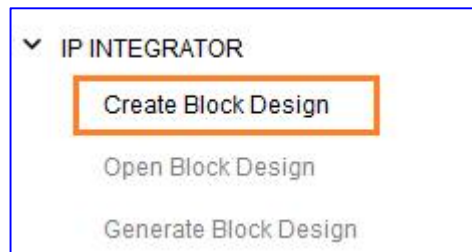
Project will be created at: D:/ebook/fpga_Microblaze2/logic_ch2/bram02

- Project Type : RTL Project* 를 선택하고 Next를 클릭
- Add Sources, Add Constraints* : Next를 클릭 → 파일은 나중에 추가
- Part : xc7a35tcsq324-1*을 선택 → Next를 클릭
- Finish*를 클릭 → 프로젝트를 생성

Block Memory Interface Implementation bram02_ch10.xpr

➤ Block Design

- 새로운 **Block** 생성을 위하여 **PROJECT MANAGER** ➔ **IP INTEGRATOR** ➔ **Create Block Design** 클릭



- Design Name : system** 으로 하고 OK를 클릭

- Diagram** 원도가 생성되었음

➔ **Add IP** (+ 아이콘)을 클릭하여 **5개의 IP를 추가**함 ➔ **Run Block Automation, Run Connection Automation**
은 **5개를 모두 추가한 후에 진행**

1. **MicroBlaze**
2. **AXI Uartlite**
3. **AXI GPIO**
4. **AXI BRAM Controller**
5. **Block Memory Generator**

- 5개의 IP를 추가한 후에 Run Block Automation**을 클릭

Block Memory Interface Implementation ^{bram02_ch10.xpr}

➤ Block Design

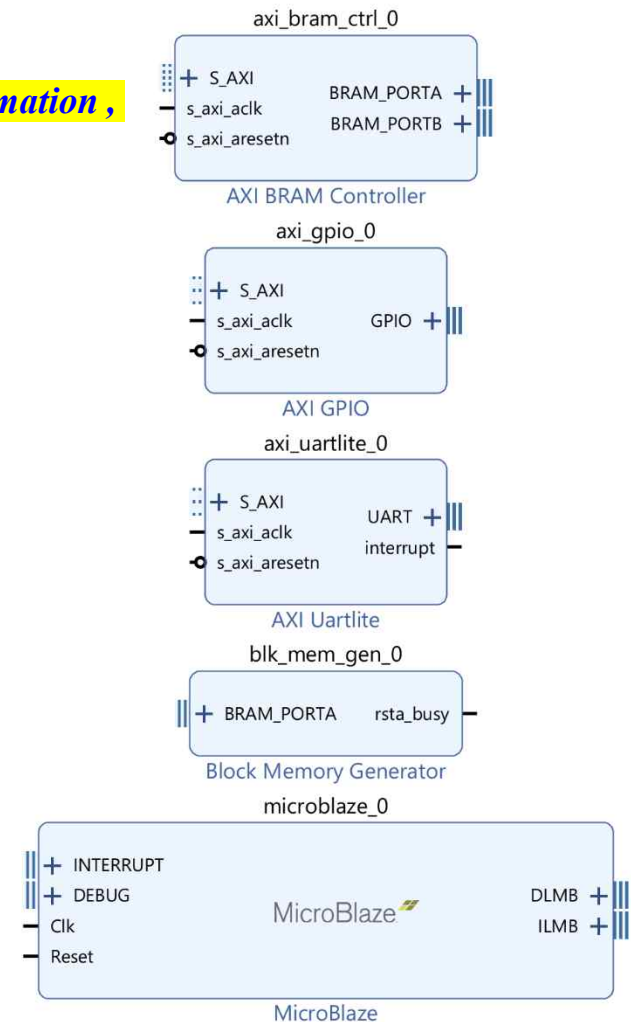


- **Diagram** 원도가 생성되었음

➔ Add IP (+ 아이콘)을 클릭하여 **5개의 IP를 추가**함 ➔ **Run Block Automation**,
Run Connection Automation 은 5개 모두 추가한 후에 진행

1. MicroBlaze
2. AXI Uartlite
3. AXI GPIO
4. AXI BRAM Controller
5. Block Memory Generator

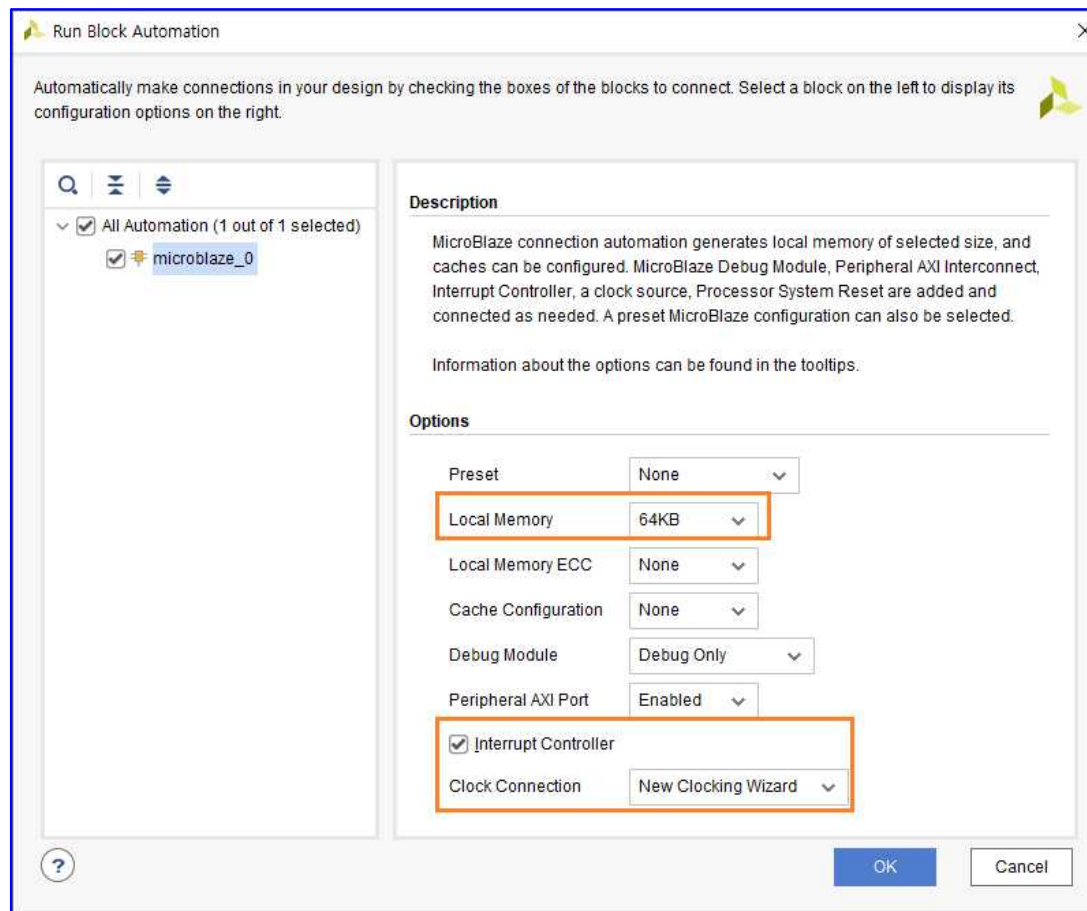
- **5개의 IP를 추가한 후에 Run Block Automation** 을 클릭



Block Memory Interface Implementation bram02_ch10.xpr

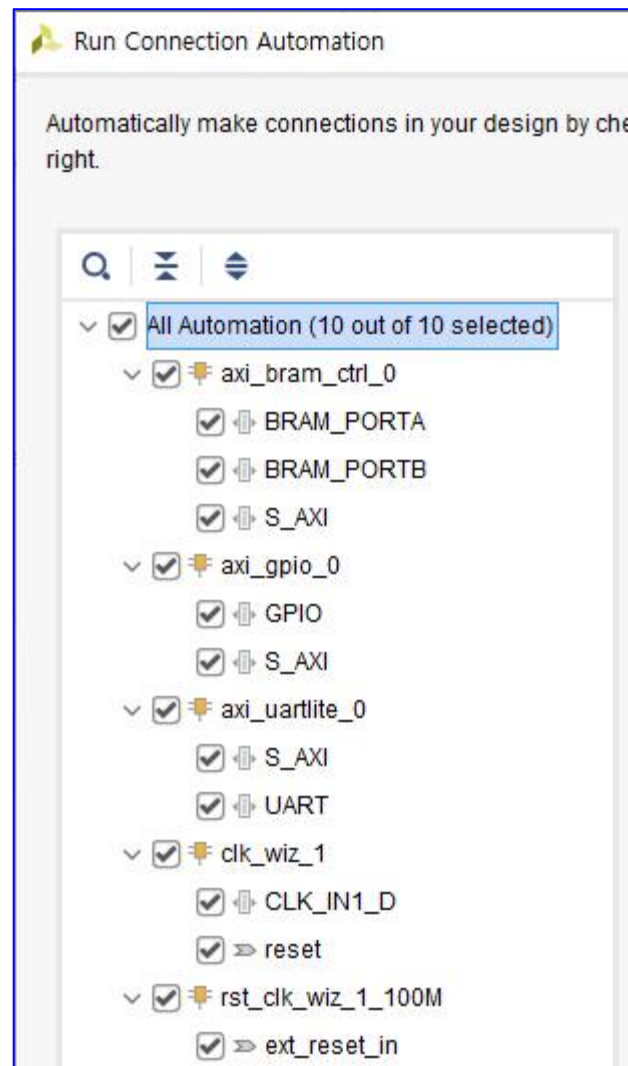
➤ **Microblaze**의 속성을 설정

- **Local Memory : 64KB** (32KB도 가능하지만, 64KB로 넉넉히 설정)
- **Interrupt Controller : check**
- **Clock Connection : New Clocking Wizard**



Block Memory Interface Implementation^{bram02_ch10.xpr}

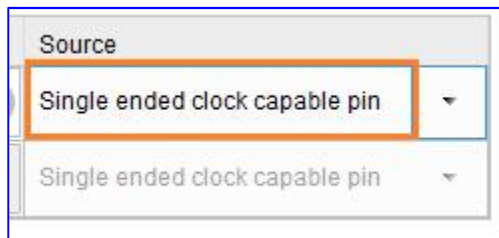
- *Microblaze*의 속성을 설정
 - **Run Connection Automation**을 클릭합니다. *All Automation*을 클릭해서 전체 모듈을 선택하고 OK를 클릭



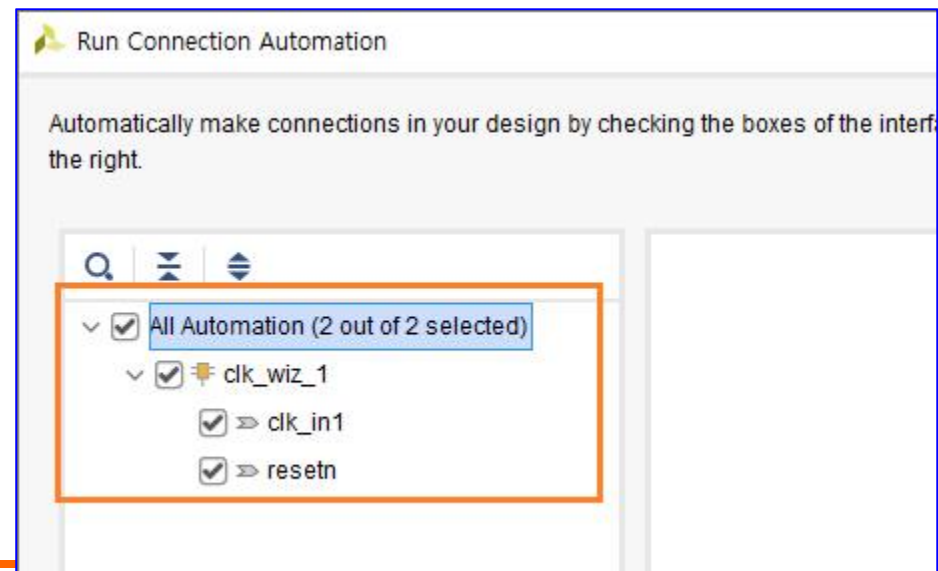
Block Memory Interface Implementation^{bram02_ch10.xpr}

➤ Microblaze의 속성을 설정

- **Clocking Wizard(*clk_wiz_1*)**의 속성을 변경함 ➔ 해당 IP를 더블 클릭하면 속성창이 나타남
- **Input Clock Source : Single ended Clock capable pin**
- **Reset Type : Active Low**



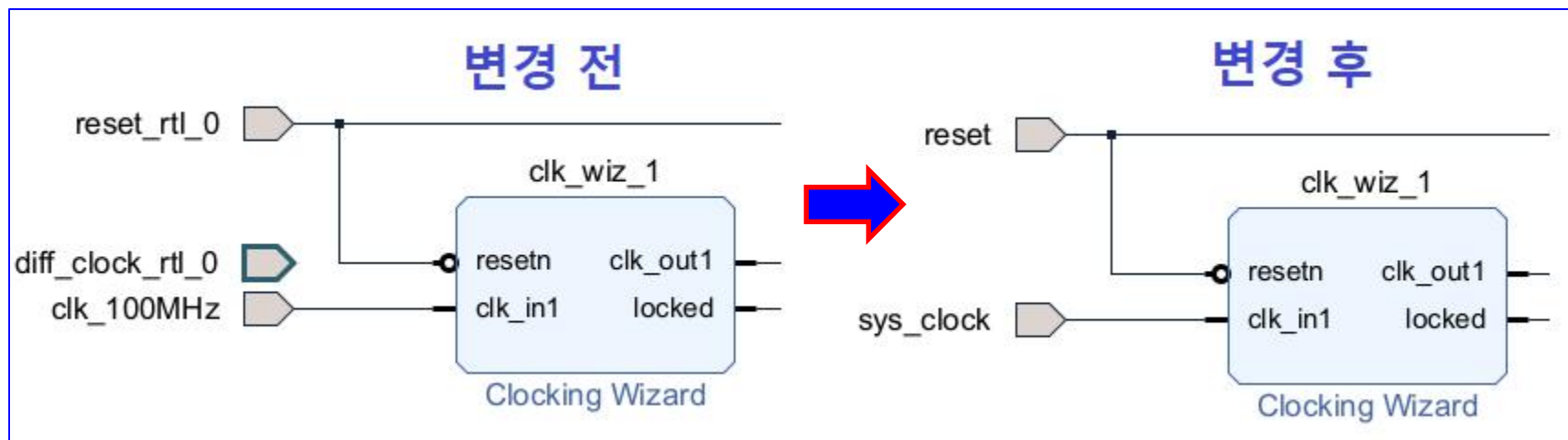
- **Run Connection Automation**이 다시 활성화 되었음 ➔ *reset_rtl_0* 핀과 *diff_clock_rtl_0* 핀과 *clk_wiz_1* 을 서로 연결하기 위하여 **Run Connection Automation**을 클릭함 (수동으로 연결해도 됨)
- **All Automation**을 클릭하고 OK를 클릭



Block Memory Interface Implementation ^{bram02_ch10.xpr}

➤ Port Properties 수정

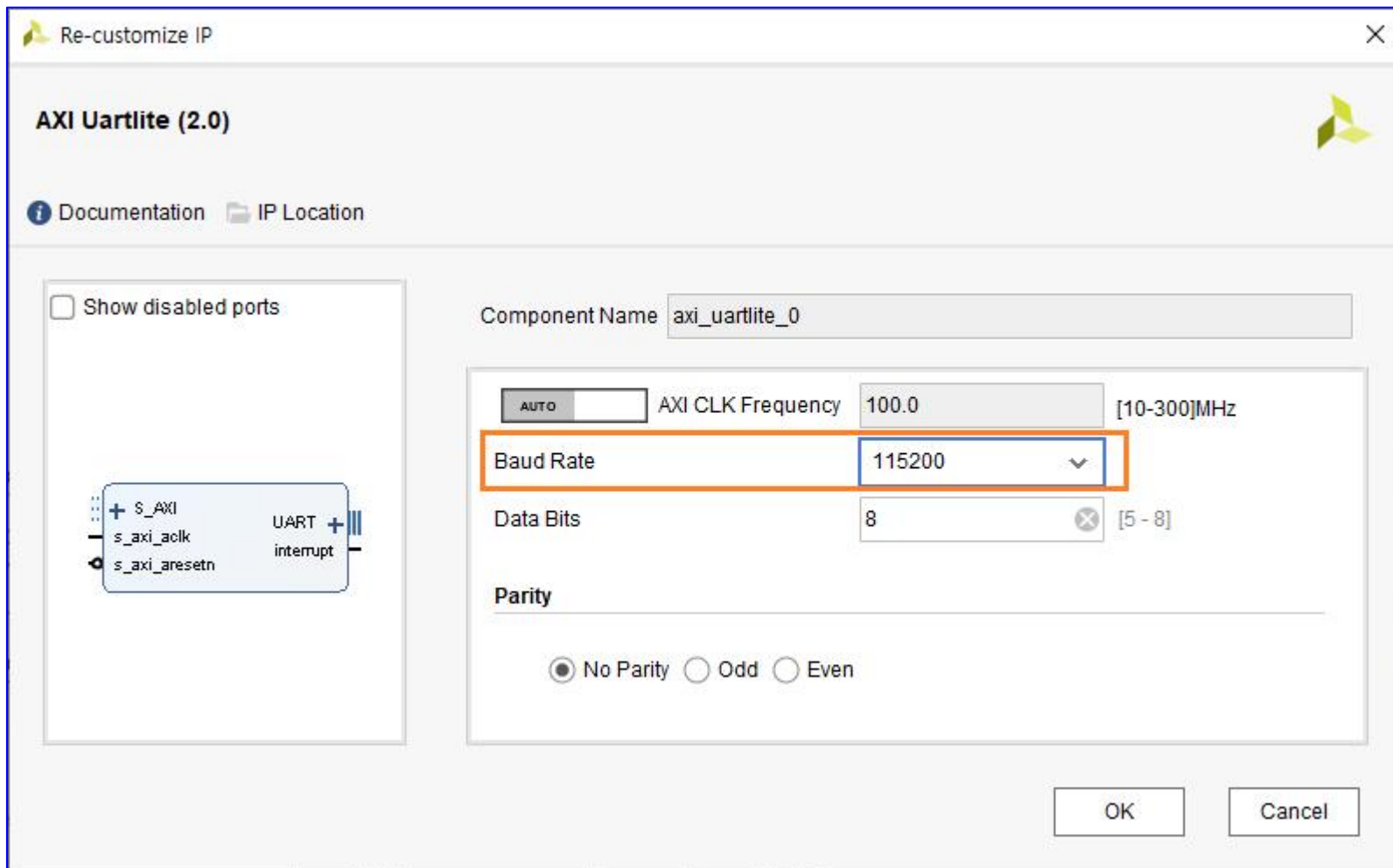
- 필요 없는 포트를 삭제하고 포트 이름을 변경함 ➔ 포트 이름 변경은 포트를 선택하면 왼쪽에 Port Properties 윈도가 나타남 ➔ Name을 변경하면 됨
 - diff_clock_rtl_0* 삭제
 - reset_rtl_0* ➔ *reset* 으로 변경
 - clk_100MHz* ➔ *sys_clock* 으로 변경
 - gpio_rtl_o* ➔ *gpio* 로 변경
 - uart_rtl_0* ➔ *uart* 로 변경



Block Memory Interface Implementation bram02_ch10.xpr

➤ **axi_uartlite_0**의 속성을 설정

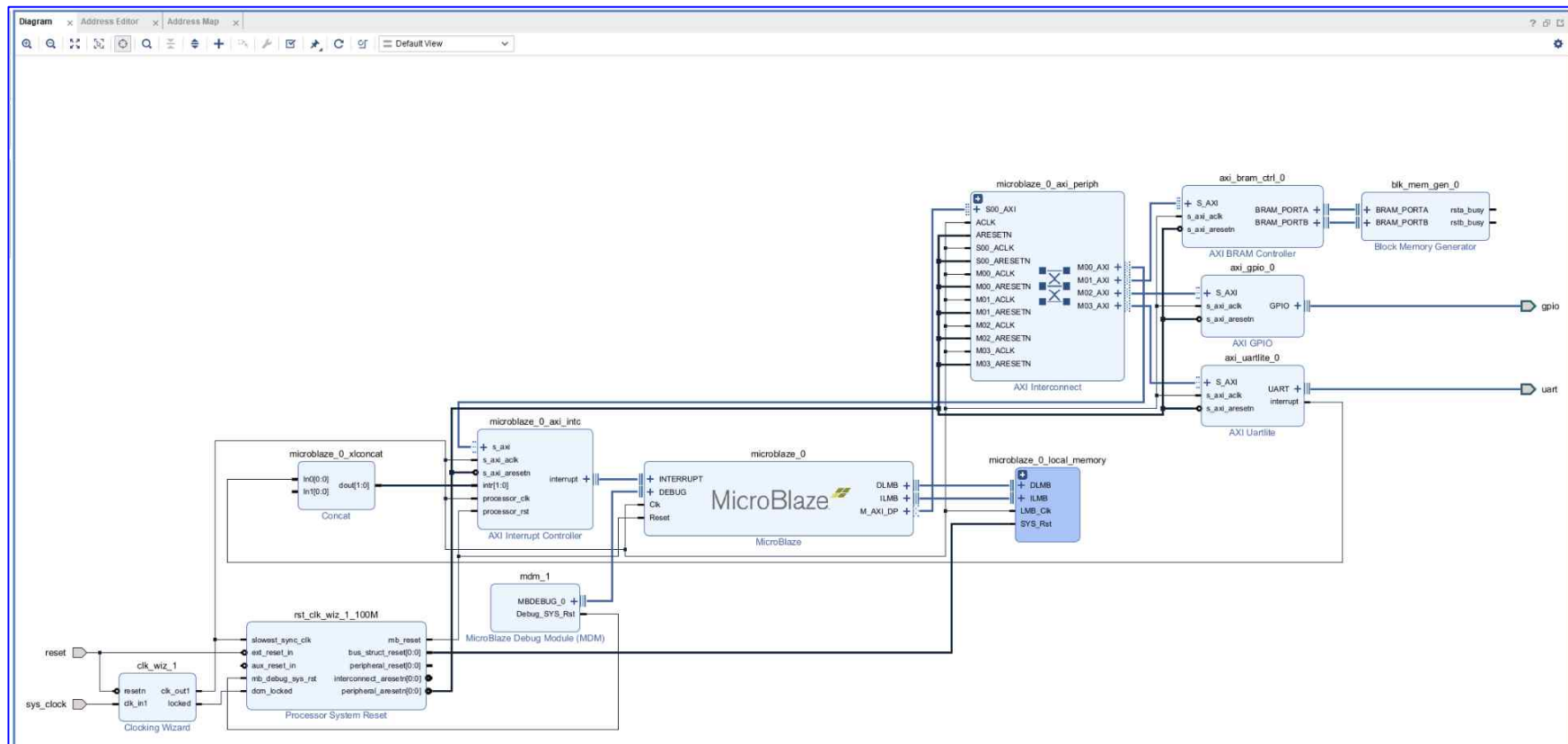
- axi_uartlite_0 을 더블 클릭해서 속성을 변경합니다. **Baud Rate : 115200** 으로 변경하고 OK를 클릭
- axi_uartlite_0의 interrupt 핀과 microblaze_0_xlconcat 의 In0[0:0]핀을 연결



Block Memory Interface Implementation bram02_ch10.xpr

➤ Diagram Implementation

- **Validate Design** 아이콘을 클릭함 ➔ 에러가 없으면 **Validation successful** 메시지 창이 나타남
- **Regenerate Layout** 아이콘을 클릭함
- 아래 그림은 지금까지 구현된 **Diagram**을 보여줌



Block Memory Interface Implementation bram02_ch10.xpr

- **AXI BRAM Controller**와 **Block Memory Generator**의 속성
 - Create HDL Wrapper... 를 실행하기 전에 추가된 AXI BRAM Controller와 Block Memory Generator의 속성을 확인 ➔ 여기서 설정된 속성은 다음장에서 사용자 로직에서 Block Memory Generator를 생성시 참고용으로 사용됨
 - **AXI BRAM Controller**를 더블 클릭함
 - ➔ 속성을 보면 **Number of BRAM interfaces : 2**로 설정되었음 ➔ 이는 **Dual Port Ram**을 위한 **Interface**임
 - ➔ Dual Port로 해도 무관하지만, 좀더 간단하게 구현하기 위하여 **Single Port Ram**으로 변경함 ➔ **Number of BRAM Interfaces : 1**로 수정함
 - ➔ **AXI4 Protocol**을 사용하고, **Data Width : 32**, **Memory Depth : 2048** (총 8KB) 임 ➔ OK를 클릭

Component Name axi_bram_ctrl_0

AXI Protocol AXI4

Data Width 32

Memory Depth (Auto) 2048

ID Width (Auto) 0

AUTO Support AXI Narrow Bursts No

Read Latency 1 [1 - 128]

Read Command Optimization No

BRAM Options

BRAM Instance (Auto) External

Number of BRAM interfaces 1

ECC Options

Enable ECC No

ECC TYPE Hamming

Enable Fault Injection No

ECC Reset Value 0

Block Memory Interface Implementation bram02_ch10.xpr

- **AXI BRAM Controller**와 **Block Memory Generator**의 속성
- **Block Memory Generator**를 더블 클릭함 → Memory Type을 변경하고 OK를 클릭함
 - **Mode : BRAM Controller**
 - **Memory Type : Single Port RAM**으로 변경
 - **Write Width, Read Width : 32**
 - **Write Depth (Read Depth) : 2048 (Memory Size : 8KB)**
 - **Operating Mode : Write First**
 - **Enable Port Type : Use ENA Pin**

Component Name: blk_mem_gen_0

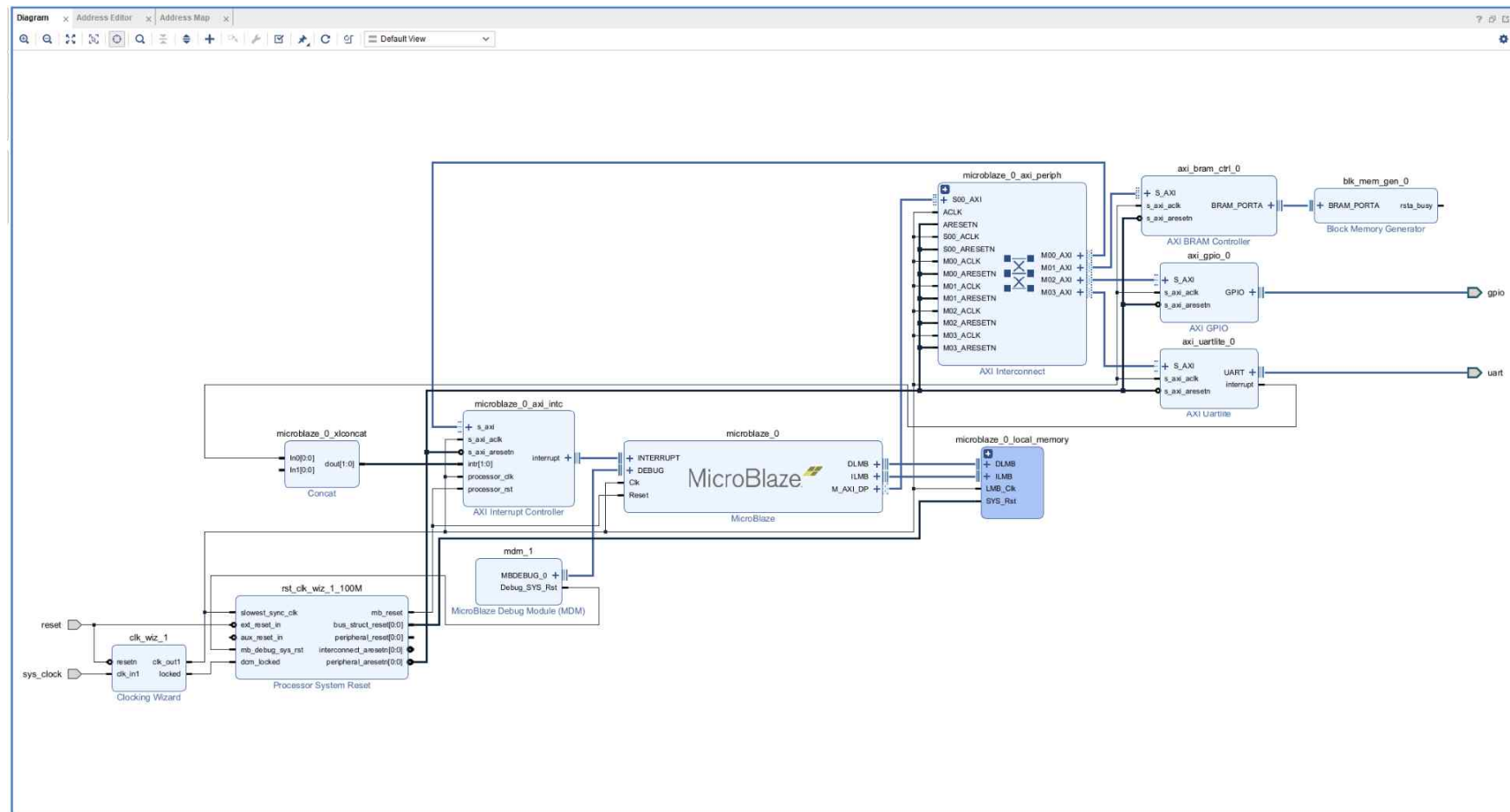
Basic	Port A Options	Other Options	Summary
Mode	BRAM Controller	<input checked="" type="checkbox"/> Generate address interface with 32 bits	
Memory Type	Single Port RAM	<input type="checkbox"/> Common Clock	

Basic	Port A Options	Other Options	Summary
Memory Size (in words)			
Write Width	32	Range: 32 to 1024 (bits)	
Read Width	32		
Write Depth	2048	Range: 2 to 1048576	
Read Depth	2048		
Operating Mode		Write First	Enable Port Type
			Use ENA Pin
Port A Optional Output Registers			
<input type="checkbox"/> Primitives Output Register		<input type="checkbox"/> Core Output Register	
<input type="checkbox"/> SoftECC Input Register		<input type="checkbox"/> REGCEA Pin	
Port A Output Reset Options			
<input checked="" type="checkbox"/> RSTA Pin (set/reset pin)		Output Reset Value (Hex) 0	
<input type="checkbox"/> Reset Memory Latch		Reset Priority CE (Latch or Register Enable)	
READ Address Change A			
<input type="checkbox"/> Read Address Change A			

Block Memory Interface Implementation bram02_ch10.xpr

➤ 수정된 **Design Diagram**

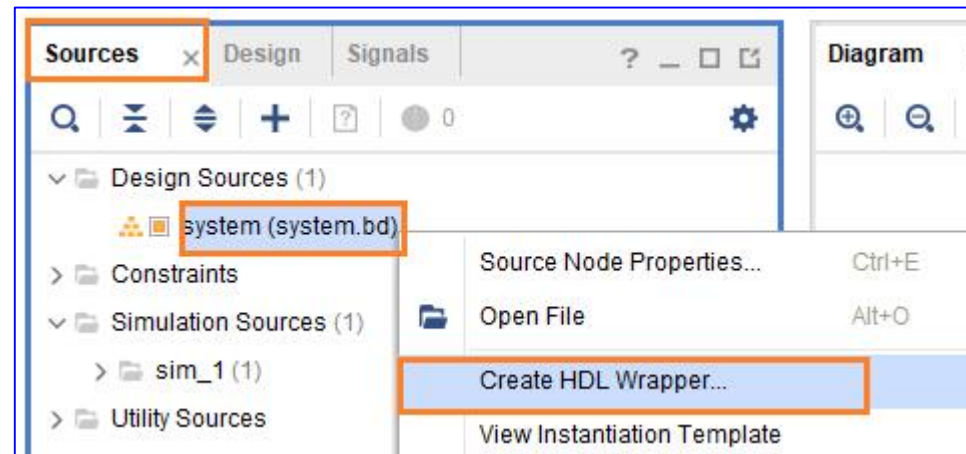
- Design이 변경되었음 ➔ Validate Design을 클릭함 ➔ 에러가 없으면 Validation Successful 메시지 윈도가 나타남
- 아래는 수정된 Diagram을 보여줌



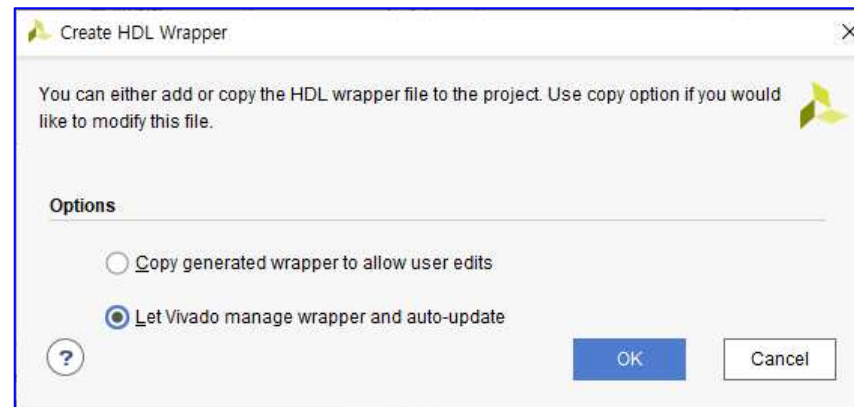
Block Memory Interface Implementation bram02_ch10.xpr

➤ Create HDL Wrapper

- Sources 탭에서 Design Sources → system → 우클릭 → Create HDL Wrapper... 를 클릭



- OK를 클릭함



- HDL Wrapper 파일이 생성되었음 (system_wrapper.v)

Block Memory Interface Implementation bram02_ch10.xpr

➤ Constraints 파일 추가

- xdc 파일을 추가함 ➔ 다운로드 받은 파일을 복사한 후, Constraints – 우클릭 – Add Sources... 를 클릭해서 해당파일을 추가함 ➔ 파일명은 “bram02.xdc”

- **## Clock signal**

- `set_property -dict { PACKAGE_PIN W5 IOSTANDARD LVCMOS33 } [get_ports sys_clock]`
- `create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports sys_clock]`

- **## Switches**

- `#set_property -dict { PACKAGE_PIN T1 IOSTANDARD LVCMOS33 } [get_ports {sw[14]}]`
- `set_property -dict { PACKAGE_PIN R2 IOSTANDARD LVCMOS33 } [get_ports {reset}]`

- **## LEDs**

- `set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33 } [get_ports {gpio_tri_o[0] }]; #LED4`
- `set_property -dict { PACKAGE_PIN E19 IOSTANDARD LVCMOS33 } [get_ports {gpio_tri_o[1] }]; #LED5`
- `set_property -dict { PACKAGE_PIN U19 IOSTANDARD LVCMOS33 } [get_ports {gpio_tri_o[2] }]; #LED6`
- `set_property -dict { PACKAGE_PIN V19 IOSTANDARD LVCMOS33 } [get_ports {gpio_tri_o[3] }]; #LED7`
- `#set_property -dict { PACKAGE_PIN W18 IOSTANDARD LVCMOS33 } [get_ports {led[4]}]`

- **##USB-RS232 Interface**

- `set_property -dict { PACKAGE_PIN B18 IOSTANDARD LVCMOS33 } [get_ports uart_rxd]`
- `set_property -dict { PACKAGE_PIN A18 IOSTANDARD LVCMOS33 } [get_ports uart_txd]`

- **## Configuration options, can be used for all designs**

- `set_property CONFIG_VOLTAGE 3.3 [current_design]`
- `set_property CFGBVS VCCO [current_design]`

- **## SPI configuration mode options for QSPI boot, can be used for all designs**

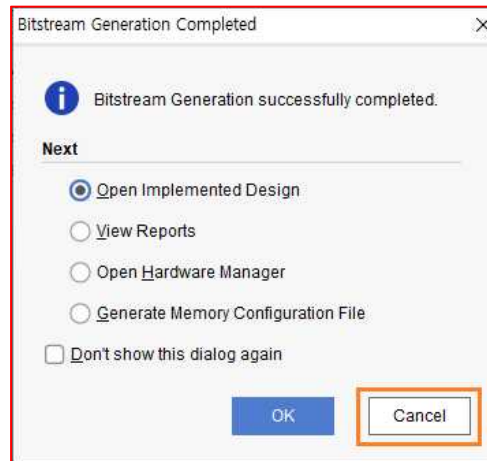
- `set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]`
- `set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]`
- `set_property CONFIG_MODE SPIx4 [current_design]`

- `port name`은 `system_wrapper.v` 파일을 참조해서 추가함

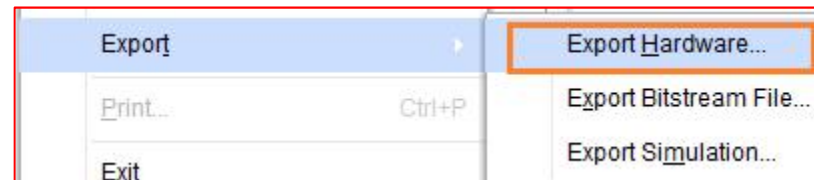
- **PROGRAM AND DEBUG ➔ Generate Bitstream** 을 클릭해서 Bitstream을 생성함 ➔ 모든 IP들을 Generation 하고, Synthesis, Implementation, Generate Bitstream 까지 진행하므로 시간이 몇 분 정도 소요됨

Block Memory Interface Implementation ^{bram02_ch10.xpr}

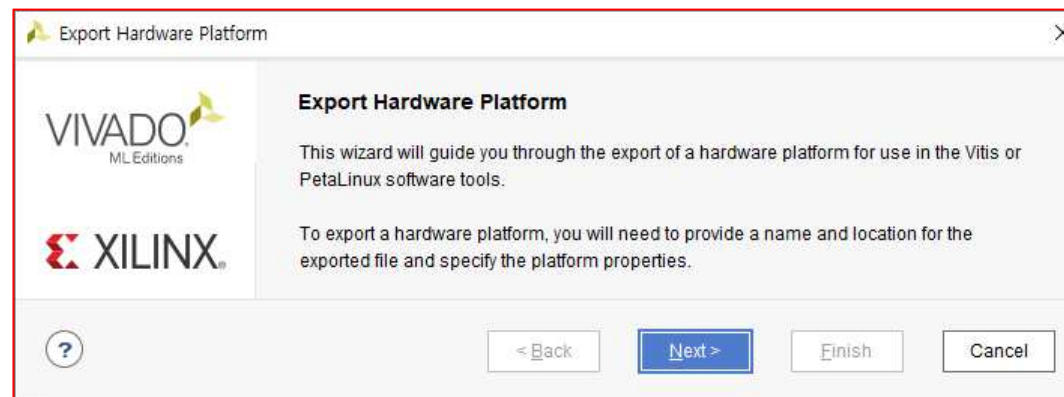
- **Bitstream**이 생성되면, 완료 윈도우가 나타남 → **Cancel** 을 클릭



- **.xsa** 파일을 생성하기 위하여 **File – Export – Export Hardware...** 를 클릭

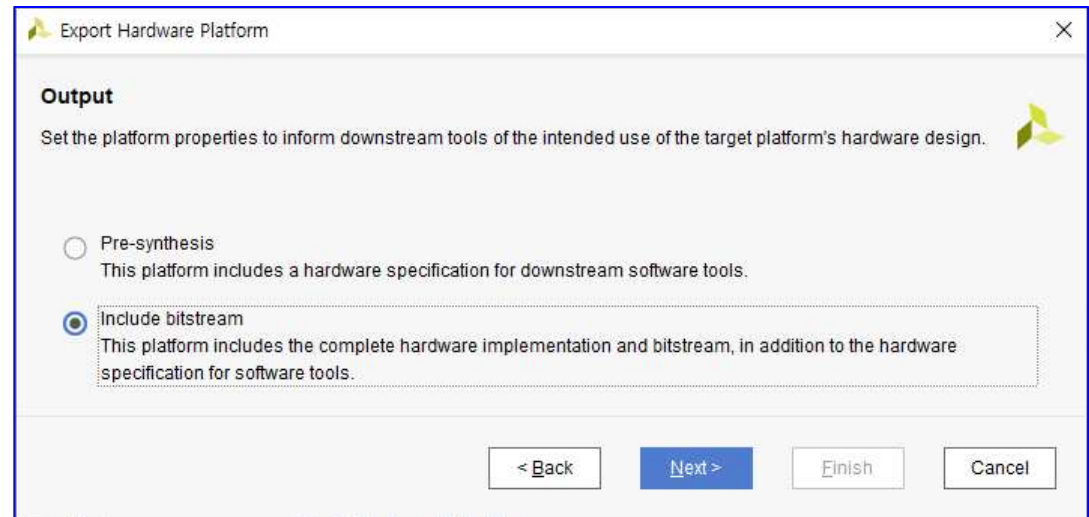


- **Next**를 클릭



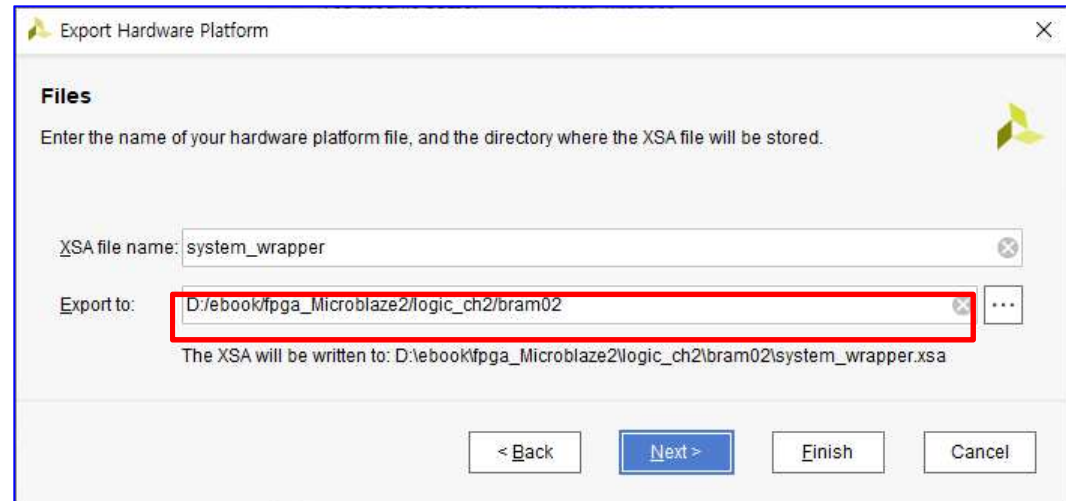
Block Memory Interface Implementation ^{bram02_ch10.xpr}

- Include bitstream 을 선택하고 Next를 클릭



The screenshot shows the 'Export Hardware Platform' dialog box, specifically the 'Output' tab. The title bar reads 'Export Hardware Platform'. Below the title bar, the text 'Output' is followed by the instruction: 'Set the platform properties to inform downstream tools of the intended use of the target platform's hardware design.' There are two radio button options: 'Pre-synthesis' (which is unselected) and 'Include bitstream' (which is selected). The 'Include bitstream' option has a description: 'This platform includes the complete hardware implementation and bitstream, in addition to the hardware specification for software tools.' At the bottom right, there are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

- 파일명, 위치를 확인하고 Next를 클릭



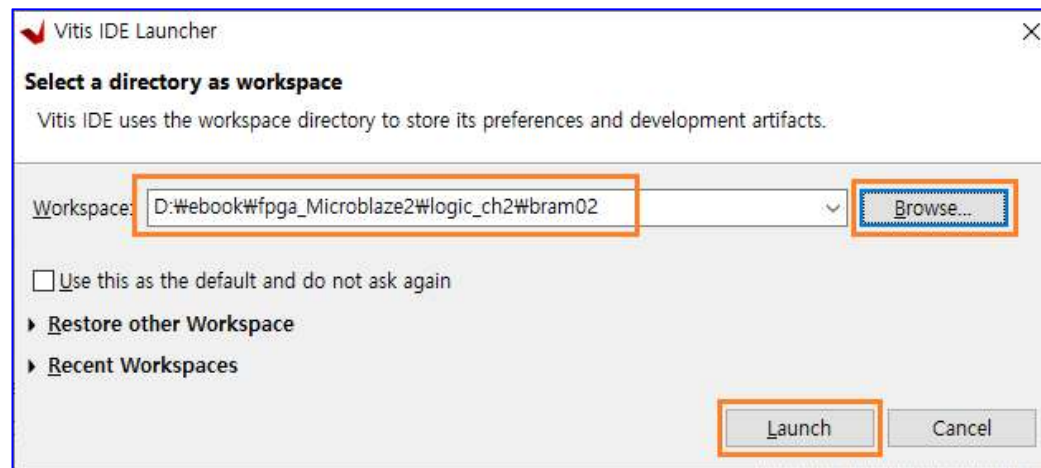
The screenshot shows the 'Export Hardware Platform' dialog box, specifically the 'Files' tab. The title bar reads 'Export Hardware Platform'. Below the title bar, the text 'Files' is followed by the instruction: 'Enter the name of your hardware platform file, and the directory where the XSA file will be stored.' There are two input fields: 'XSA file name:' with the text 'system_wrapper' and 'Export to:' with the text 'D:\ebook\tpga_Microblaze2\logic_ch2\bram02'. The 'Export to:' field is highlighted with a red rectangle. Below the 'Export to:' field, there is a text line: 'The XSA will be written to: D:\ebook\tpga_Microblaze2\logic_ch2\bram02\system_wrapper.xsa'. At the bottom right, there are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

- 마지막으로 Finish를 클릭하면 xsa 파일이 생성

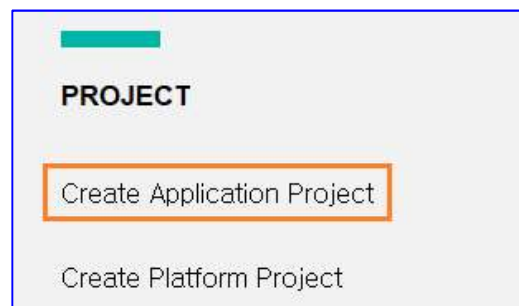
Block Memory Interface Implementation bram02_ch10.xpr

➤ Application SW Implementation

- 이전 장에서는 생성된 Block Memory를 Access 하는 응용 SW를 구현함 ➔ Tools – Launch Vitis IDE 를 클릭하여 it is를 실행함 ➔ Browse... 를 클릭해서 프로젝트 폴더로 이동 후 Launch 를 클릭



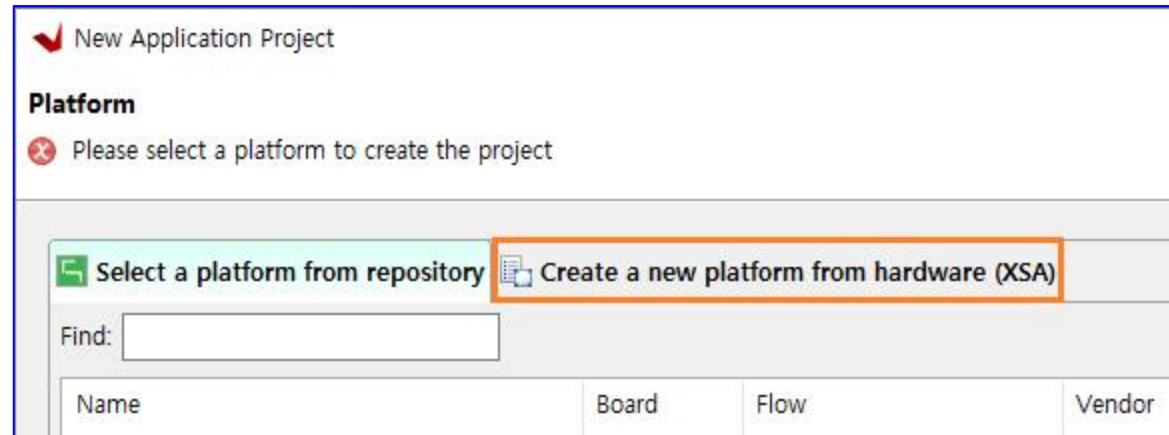
- Vitis IDE가 실행됨 ➔ PROJECT ➔ Create Application Project를 클릭



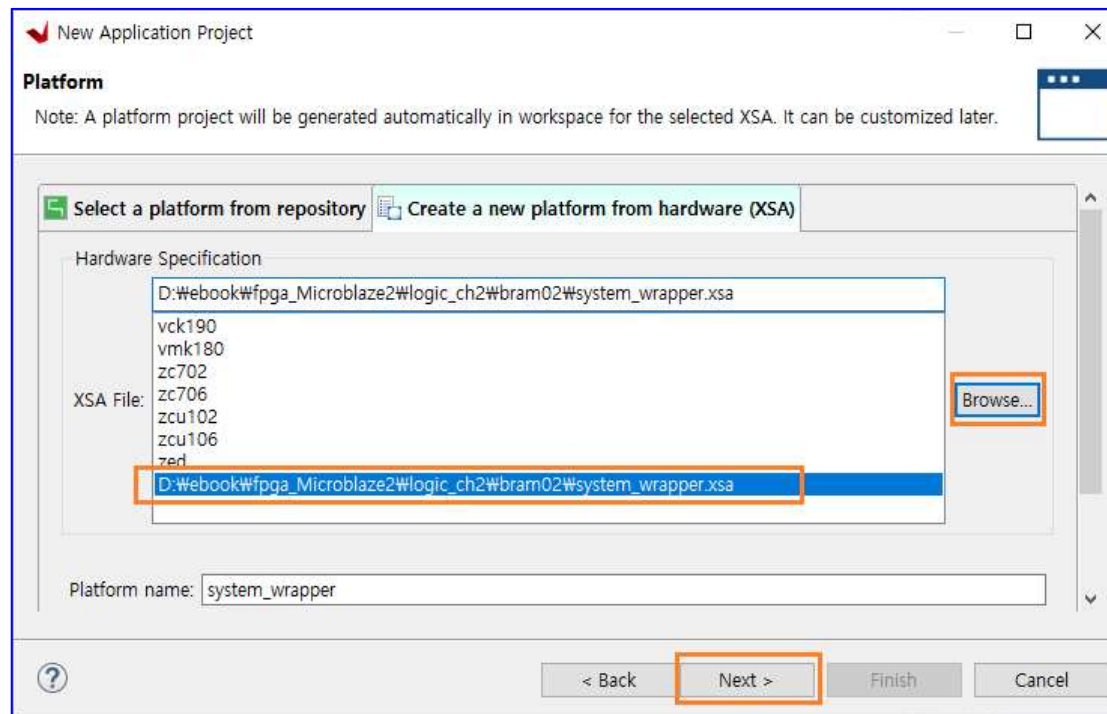
Block Memory Interface Implementation ^{bram02_ch10.xpr}

➤ Application SW Implementation

- Platform 윈도우에서 Create a new platform from hardware (XSA)를 클릭



- XSA File : 우측의 Browse... 를 클릭하여 생성한 system_wrapper.xsa 파일을 선택하고 Next 버튼을 클릭



Block Memory Interface Implementation bram02_ch10.xpr

➤ Application SW Implementation

- **Application Project name : bramApp02** 입력하고(또는 사용자 지정) Next를 클릭

New Application Project

Application Project Details
Specify the application project name and its system project properties

Application project name:

System Project
Create a new system project for the application or select an existing one from the workspace

Select a system project

System project details

System project name:

Target processor

Select target processor for the Application project.

Processor	Associated applications
microblaze_0	bramApp02

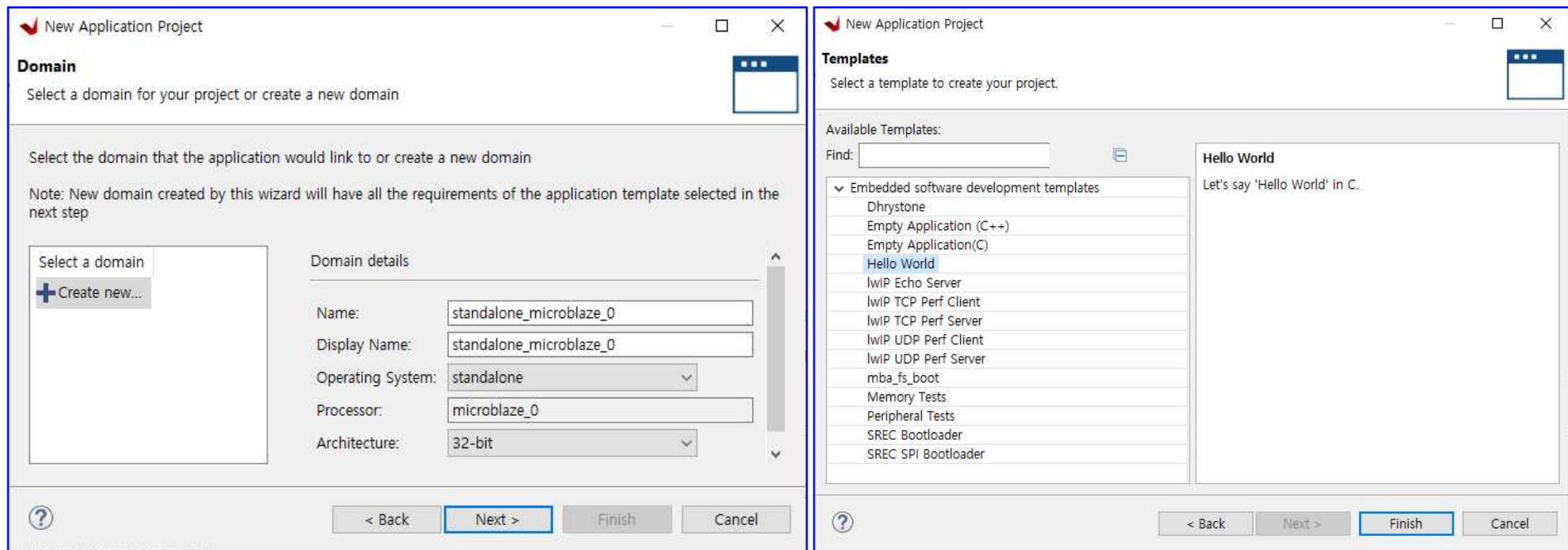
Show all processors in the hardware specification: ☒

< Back **Next >** Finish Cancel

Block Memory Interface Implementation bram02_ch10.xpr

➤ Application SW Implementation

- Next를 클릭 ➔ Template 에서 Hello World를 선택하고 Finish를 클릭하면 프로젝트가 생성



Block Memory Interface Implementation bram02_ch10.xpr

➤ Application SW Implementation

- helloworld.c 에 그림과 같이 코드 추가
- 65 – 69 : 32bits(4바이트)씩 총 8KB를 write 합니다.
- 71 – 78 : 32bits(4바이트)씩 총 8KB를 read 하고, write 한 데이터와 비교하여 error를 확인

```
48 #include <stdio.h>
49 #include "platform.h"
50 #include "xil_printf.h"
51 #include "sleep.h"
52
53
54 int main()
55 {
56     int i;
57     uint32_t rdata;
58     int err_cnt=0;
59
60     init_platform();
61
62     print("\r\n Hello World \r\n");
63     print("Successfully ran Hello World application\r\n");
64
65     xil_printf("write bram ..\r\n");
66     for(i=0; i<2048; i++)
67     {
68         Xil_Out32(XPAR_AXI_BRAM_CTRL_0_S_AXI_BASEADDR+(4*i), 0xaabb0000+i);
69     }
70
71     xil_printf("read bram ..\r\n");
72     for(i=0; i<2048; i++)
73     {
74         rdata = Xil_In32(XPAR_AXI_BRAM_CTRL_0_S_AXI_BASEADDR+(4*i));
75         if(rdata != 0xaabb0000+i) err_cnt++;
76         xil_printf("addr : %04x , w : %x r : %x \r\n", i, 0xaabb0000+i, rdata);
77     }
78     xil_printf(" error count : %d \r\n", err_cnt);
79
80     i = 0;
81     while(1)
82     {
83         sleep(100);
84         xil_printf("count : %d \r\n", i++);
85     }
86
87     cleanup_platform();
88     return 0;
89 }
```

Block Memory Interface Implementation bram02_ch10.xpr

- **axi_gpio_0**의 속성을 설정
 - **axi_gpio_0**를 더블 클릭해서 속성을 변경함 → gpio 핀은 총 4핀을 사용함 (각각 LD4 ~ LD7에 연결됨)
 - ✓ **All Outputs** : check
 - ✓ **GPIO Width** : 4
 - ✓ **Enable Dual Channel, Enable Interrupt** : 모두 **Uncheck**

GPIO

☐ All Inputs

☒ All Outputs

GPIO Width [1 - 32]

Default Output Value [0x00000000,0xFFFFFFFF]

Default Tri State Value [0x00000000,0xFFFFFFFF]

☐ Enable Dual Channel

Block Memory Interface Implementation bram02_ch10.xpr

➤ Application SW Implementation

- `Xil_In32()`, `Xil_Out32()` 함수는 *inline* 함수로 정의된 메모리에 Access 하는 함수임 → “`xil_io.h`”에 정의되어 있음

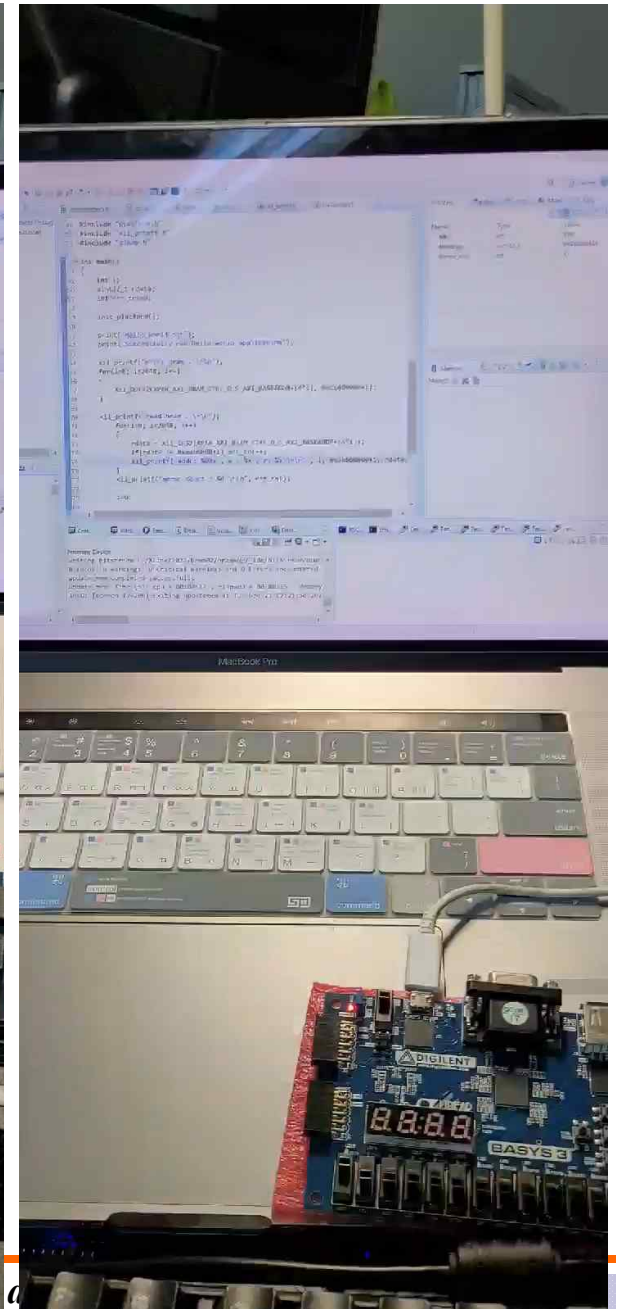
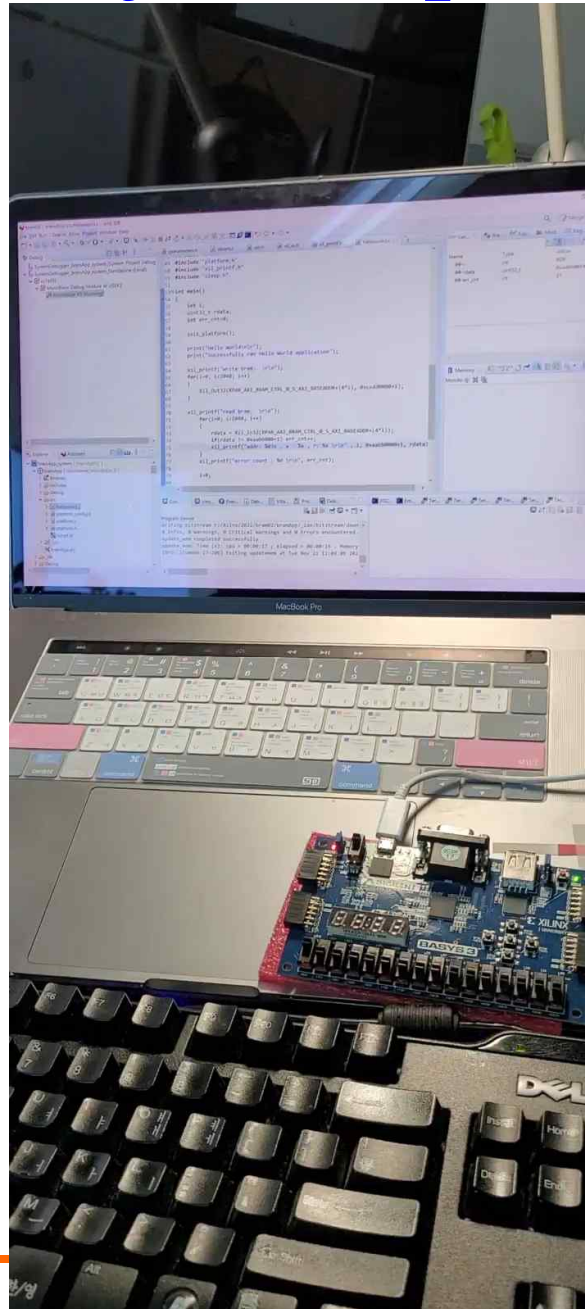
```
134 static INLINE u32 Xil_In32(UINTPTR Addr)
135 {
136     return *(volatile u32 *) Addr;
137 }

208 static INLINE void Xil_Out32(UINTPTR Addr, u32 Value)
209 {
210     /* write 32 bit value to specified address */
211     #ifndef ENABLE_SAFETY
212         volatile u32 *LocalAddr = (volatile u32 *)Addr;
213         *LocalAddr = Value;
214     #else
215         XStl_RegUpdate(Addr, Value);
216     #endif
217 }
```

Block Memory Interface Implementation bram02_ch10.xpr

➤ Application SW Implementation

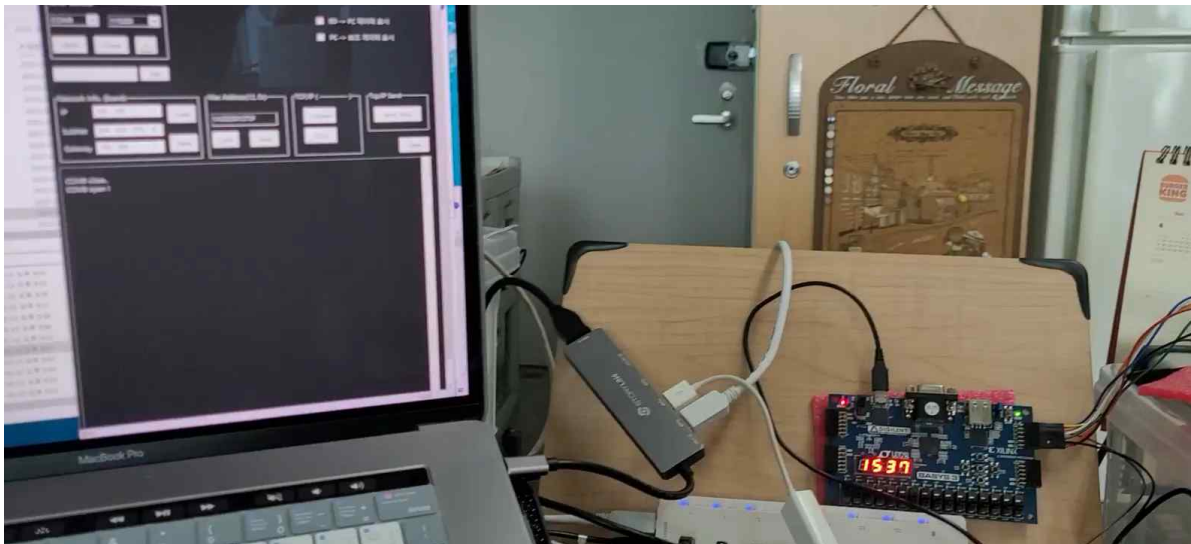
- 프로그램을 **Build : Project** → **Build All**
- 프로그램을 다운로드 : **Xilinx** → **Program Device**
- ComPortMaster를 이용한 결과확인
- Vitis-Terminal 을 이용한 결과 확인

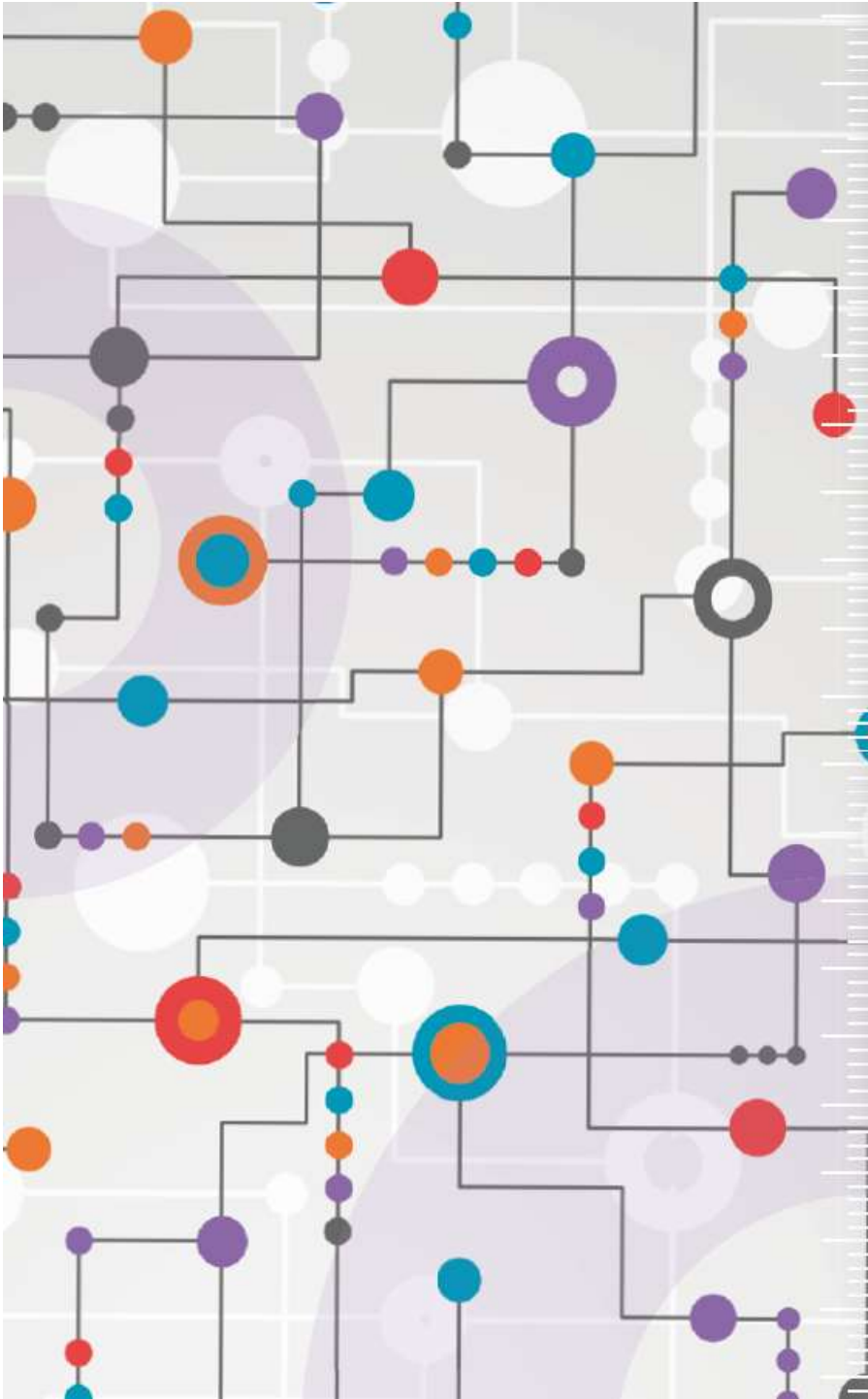


Block Memory Interface Implementation bram02_ch10.xpr

➤ Application SW Implementation

- 프로그램을 **Build : Project** ➔ **Build All**
- 프로그램을 다운로드 : **Xilinx** ➔ **Program Device**
- **WinIDT**를 이용한 결과 확인





수고하셨습니다.