

Verilog HDL Design using Vivado

Tool : Xilinx Vivado 2019.2

Kit : Ultra96 Training Kit

Chapter 1 Introduction

- **Introduction to Verilog HDL**
- **Introduction to Xilinx Devices and Tools**
- **Introduction to Inipro and Ultra96 Training Kit**
- **Vivado Installation**
- **Creating Vivado Project**

Introduction to Verilog HDL

- ❖ HDL(Hardware Description Language)은 하드웨어(디지털 논리 회로)를 묘사하는 언어라는 뜻이다.
- ❖ 현재 사용하고 있는 HDL은 Verilog HDL과 VHDL이 있으며 이 중에서 Verilog HDL에 대해서 알아본다.
- ❖ Verilog HDL은 1983~1984년에 Prabhu Goel과 Phil Moorby에 의해서 만들어 졌으며 관련 소유권을 Cadence사가 1990년에 사들였다.
- ❖ VHDL의 성공사례들의 증가로 인해 위기를 느낀 Cadence는 1995년에 OVI(Open Verilog International)에 오픈하게 되고 이후에 OVI에서 내 놓은 안이 IEEE 표준(1364-1995)으로 채택되면서 널리 쓰이게 되었다.
- ❖ 이 후에 좀 더 확장되고 업데이트되어 2001년에 1364-2001 그리고 2005년에 1364-2005가 IEEE 표준으로 채택되어 사용하고 있다.
- ❖ Verilog HDL은 C언어의 문법과 비슷하여 엔지니어들로부터 많은 호응을 얻고 있다.
- ❖ VHDL보다 출발이 조금 늦었지만 현재는 VHDL보다 사용자가 더 많은 것으로 알려져 있다.
- ❖ 특히 Verilog HDL은 처음 개발할 때부터 시뮬레이션을 쉽고 편하게 하기 위한 언어로 개발되어 시뮬레이션에 강점이 있어서 시뮬레이션을 통해 시스템 검증을 주로 해야 하는 팹리스 업체들은 대부분 Verilog HDL을 사용하고 있다.
- ❖ Verilog HDL은 보통 HDL을 제외하고 편의상 Verilog라고 부른다.

Chapter 1 Introduction

- Introduction to Verilog HDL
- **Introduction to Xilinx Devices and Tools**
- Introduction to Inipro and Ultra96 Training Kit
- Vivado Installation
- Creating Vivado Project

Introduction to Xilinx

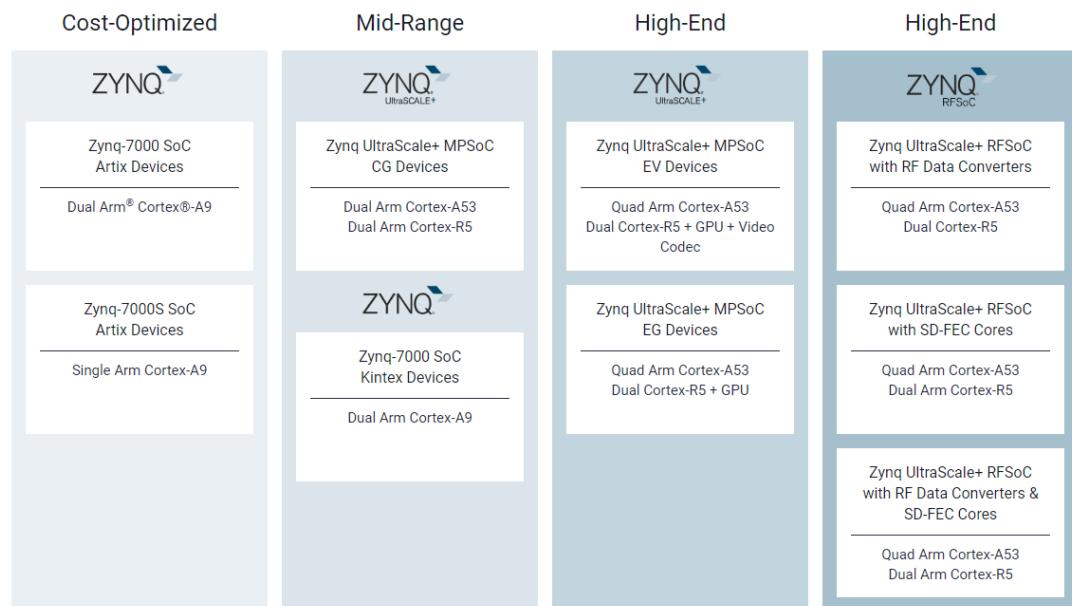
- ❖ 자일링스는 인텔과 함께 전세계 FPGA 시장을 양분하고 있으며 현재 이 두 회사가 FPGA 시장의 90% 이상을 점유하고 있다.
- ❖ 인텔은 2015년에 알테라를 인수하면서 FPGA 벤더 중 하나가 되었으며 Lattice Semiconductor 와 Microsemi가 특정 분야에서는 두각을 나타내어 일정 점유율을 차지하고 있다.
- ❖ 자일링스는 칩을 설계하는 팹리스(Fabless) 회사로 분류된다. 여기서 Fab은 제조 설비를 의미하는 fabrication의 줄임말이며 Fabless는 fabrication과 less를 합친 합성어이다.
- ❖ 팹리스는 반도체를 생산하는 설비가 없는 회사들을 지칭하며 반도체 설계만을 전문적으로 하는 회사들을 분류하여 부르는 말이다.
- ❖ 이런 팹리스 회사들은 설계한 칩을 생산하기 위해 반도체 생산라인을 가진 업체에게 주문 의뢰하여 생산한 칩을 판매한다.
- ❖ 반도체 생태계(Ecosystem)에서 주문의뢰를 받아서 생산을 해주는 업체들이 있는데 이런 업체를 파운드리(Foundry)라고 부르며 파운드리는 반도체 설계는 하지 않고 주문의뢰를 받아서 위탁생산만을 전문적으로 하는 회사들을 분류하는 말이다.

Xilinx FPGA Devices



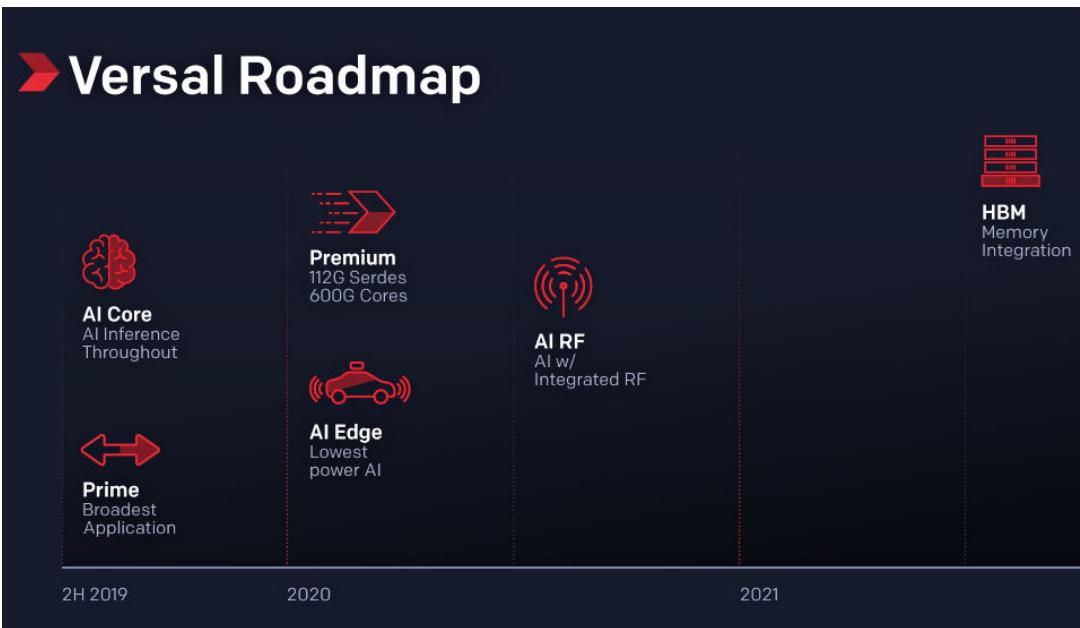
- ❖ 위 그림에서 위에 있는 45nm 와 같은 숫자는 회로선 폭의 크기이며 반도체를 만드는 공정의 정밀도를 의미한다.
- ❖ 자일링스의 최신 FPGA는 16nm 공정으로 생산하는 UltraScale+이다.
- ❖ FPGA 시리즈는 저가형 중간형 고가형 디바이스 패밀리로 나누어지는데 위 그림에서 아래에 위치한 디바이스가 저가형이고 위로 올라갈수록 고가형의 디바이스이다.

Xilinx SoC Devices



- ❖ 자일링스는 Zynq라는 이름의 SoC디바이스들을 판매하고 있으며 이런 SoC 디바이스들은 기본적으로 Xilinx FPGA와 ARM Cortex 시리즈의 Processing System이 결합한 형태이다.
- ❖ Zynq 디바이스들은 Zynq-7000 SoC, Zynq UltraScale+ MPSoC, Zynq UltraScale+ RFSoC와 같은 3종류의 SoC 디바이스들로 나누어진다.
- ❖ 본 교육에서는 Zynq UltraScale+ MPSoC중 EG Devices를 탑재한 Ultra96 보드를 포함한 Ultra96 Training Kit를 사용하여 실습을 한다.

Xilinx ACAP Devices



- ❖ 자일링스는 최근 새로운 종류의 디바이스인 ACAP(Adaptive Compute Acceleration Platform)으로 버설(Versal)을 공개했다.
- ❖ 오랫동안 FPGA 벤더였던 자일링스는 7nm공정으로 생산된 새로운 플랫폼인 버설을 공개하며 컴퓨팅 가속시장을 주도하고 있는 GPU에 도전장을 내민것으로 평가된다.
- ❖ 그동안 컴퓨팅 가속부분에 관심 있는 많은 플레이어들은 FPGA의 유연성과 병렬계산 능력을 주목하고 있었으며 이런 부분을 버설이 현실화 할 것으로 기대된다.
- ❖ 현재는 버설 디바이스 중 AI Core Series와 Prime Series만 공개한 상태이며 그림 1-3과 같은 버설 로드맵에 따라 향후 다른 시리즈의 디바이스들을 출시할 예정이다.

Introduction to Xilinx Tools

- ❖ 자일링스 디바이스를 설계 및 구현하려면 자일링스의 개발 툴을 사용해야 한다.
- ❖ 자일링스의 개발 툴은 하드웨어 개발 툴과 소프트웨어 개발 툴로 나누어진다.
- ❖ 하드웨어 개발 툴에는 Verilog와 VHDL과 같은 HDL을 사용하여 RTL설계를 하기 위한 Vivado툴과 HLS(High Level Synthesis)를 사용하여 하드웨어를 설계하기 위한 Vivado HLS가 있다.
- ❖ C,C++, OpenCL과 같은 소프트웨어를 통해 하드웨어 설계 및 소프트웨어 개발을 할 수 있는 툴은 기존에는 SDK, SDSoC와 SDAccel이 있었는데 2019.2버전부터 Vitis로 통합되었다.
- ❖ 본 교육에서는 Verilog를 사용하여 RTL설계를 하는 부분에 대해서 다룰 예정이니 Vivado 를 사용한다.

Chapter 1 Introduction

- Introduction to Verilog HDL
- Introduction to Xilinx Devices and Tools
- **Introduction to Inipro and Ultra96 Training Kit**
- Vivado Installation
- Creating Vivado Project

Introduction to Inipro

- ❖ 이니프로는 FPGA 관련 제품과 교육 및 디자인 서비스를 제공하는 회사이다.
- ❖ 이니프로 홈페이지(<https://www.inipro.net/>)에 들어가면 이니프로에서 판매하고 있는 다양한 제품들과 제공하고 있는 교육 및 디자인 서비스들을 확인할 수 있다.
- ❖ 본 교육에서 사용되는 소스코드 및 관련자료는 이니프로 github사이트 (<https://github.com/inipro>)의 verilog_vivado 저장소를 통해 다운로드하여 사용할 수 있다.
- ❖ 이니프로에서 운영하는 커뮤니티 사이트인 이니프로 카페 (<https://cafe.naver.com/plduser/>)를 이용하면 추후에 다양한 경험을 가지고 있는 카페 회원들로부터 답변을 들을 수 있다.

Introduction to Ultra96 Training Kit



- ❖ Ultra96 Training Kit는 Ultra96 보드에 여러 가지 실습을 하기 위해 필요한 모듈 및 액세서리들을 하나로 묶어서 만든 번들 제품으로 이니프로에서만 판매하는 제품이다.
- ❖ Ultra96보드는 96Boards의 한 종류이고 96Boards는 Linaro재단에서 ARM코어가 포함된 여러가지 SoC칩들을 하나의 하드웨어 품 팩터를 가진 보드들로 구성하여 하드웨어들 간의 호환성을 높이고자 만든 것으로 96Boards웹사이트 (<https://www.96boards.org/>)에 들어가면 자세한 내용을 확인할 수 있다.
- ❖ 본 교육에서는 Ultra96 Training Kit를 사용한 여러가지 실습들을 통해 Verilog HDL을 이용하여 디지털 시스템을 설계하고 검증하는 내용을 다룬다.

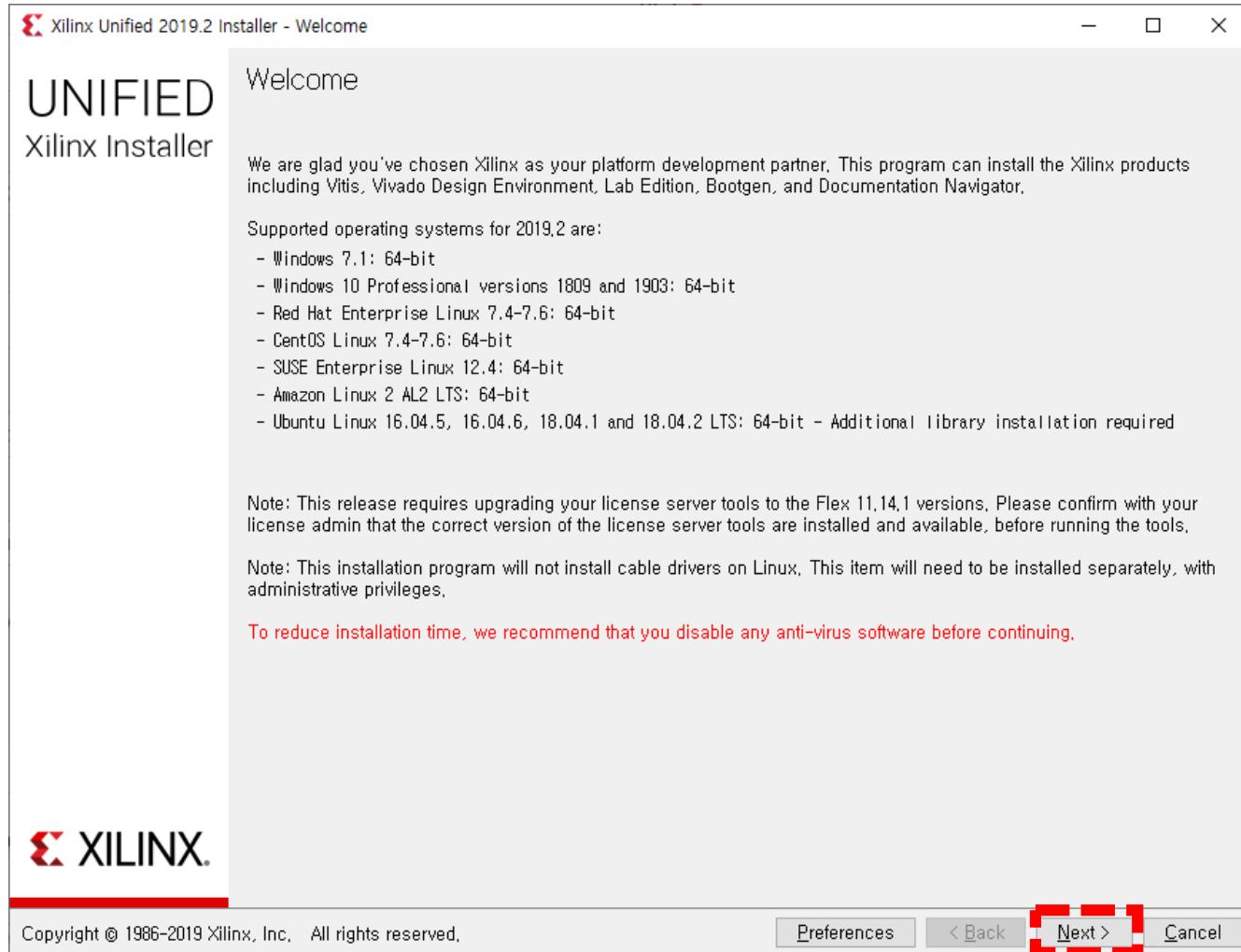
Chapter 1 Introduction

- Introduction to Verilog HDL
- Introduction to Xilinx Devices and Tools
- Introduction to Inipro and Ultra96 Training Kit
- **Vivado Installation**
- Creating Vivado Project

Step 1 Download Vivado

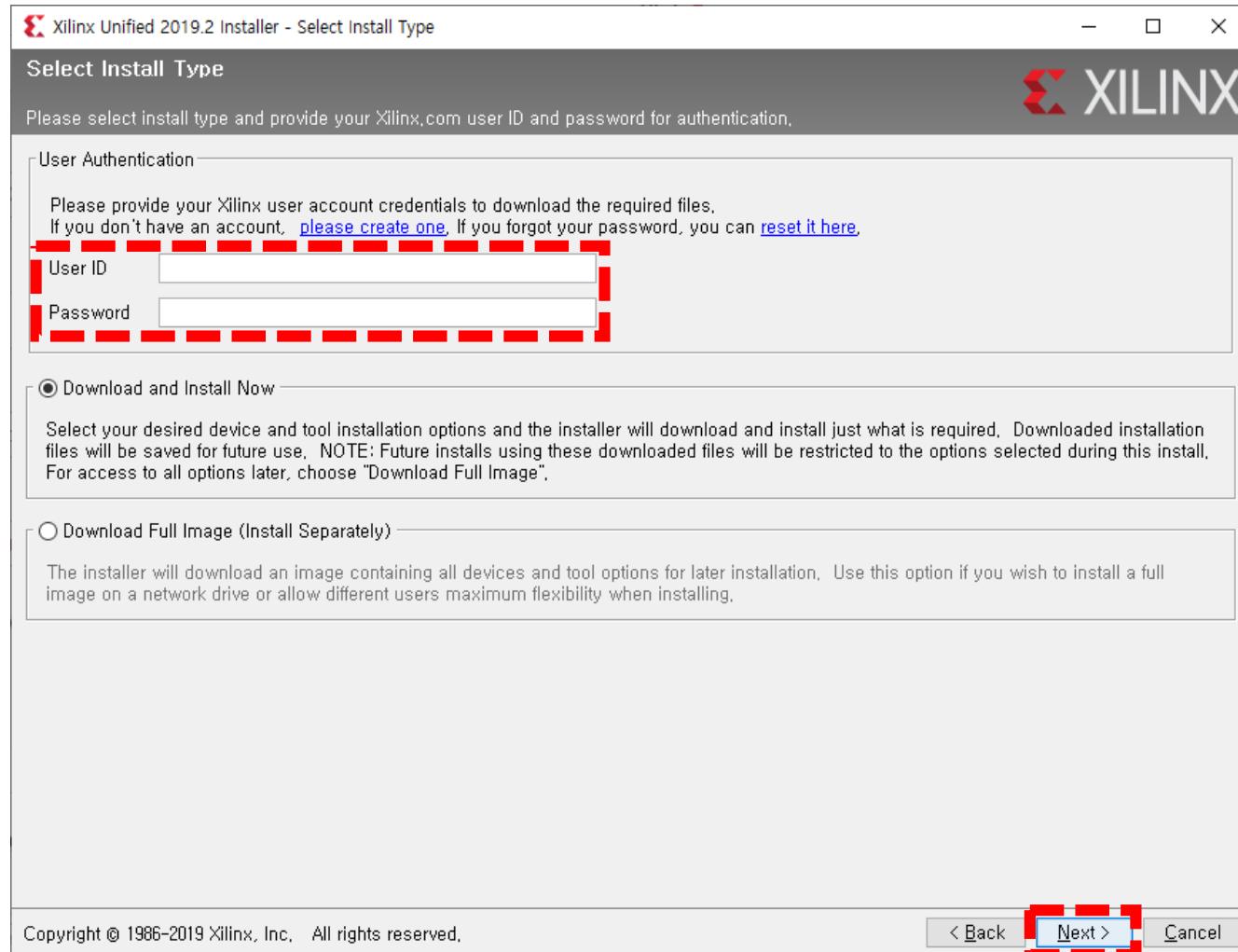
- ❖ 자일링스 홈페이지(www.xilinx.com)에 접속한다.
- ❖ SUPPORT 메뉴 아래 Downloads & Licensing 메뉴를 클릭한다.
- ❖ Web Installer를 사용하는 방식과 Single File을 다운로드하여 설치하는 방식이 있는데 Web Installer방식은 작은 사이즈의 Web Installer를 다운로드 받아서 설치할 때 인터넷을 통해 필요한 파일을 다운로드 받아서 설치하는 방식이고 Single File 방식은 설치에 필요한 전체 파일을 다운로드 받아서 설치하는 방식이다.
- ❖ 여기서는 Web Installer를 사용할 예정이므로 Web Installer를 다운로드 받는다.
- ❖ Xilinx 로그인 화면이 나오면 이미 가입이 되어 있으면 기존ID로 로그인하고 가입되어 있지 않으면 계정을 만들어서 로그인해야 한다. 정상적으로 로그인이 되면 다운로드가 시작된다.

Step 2 Vivado Installation 1



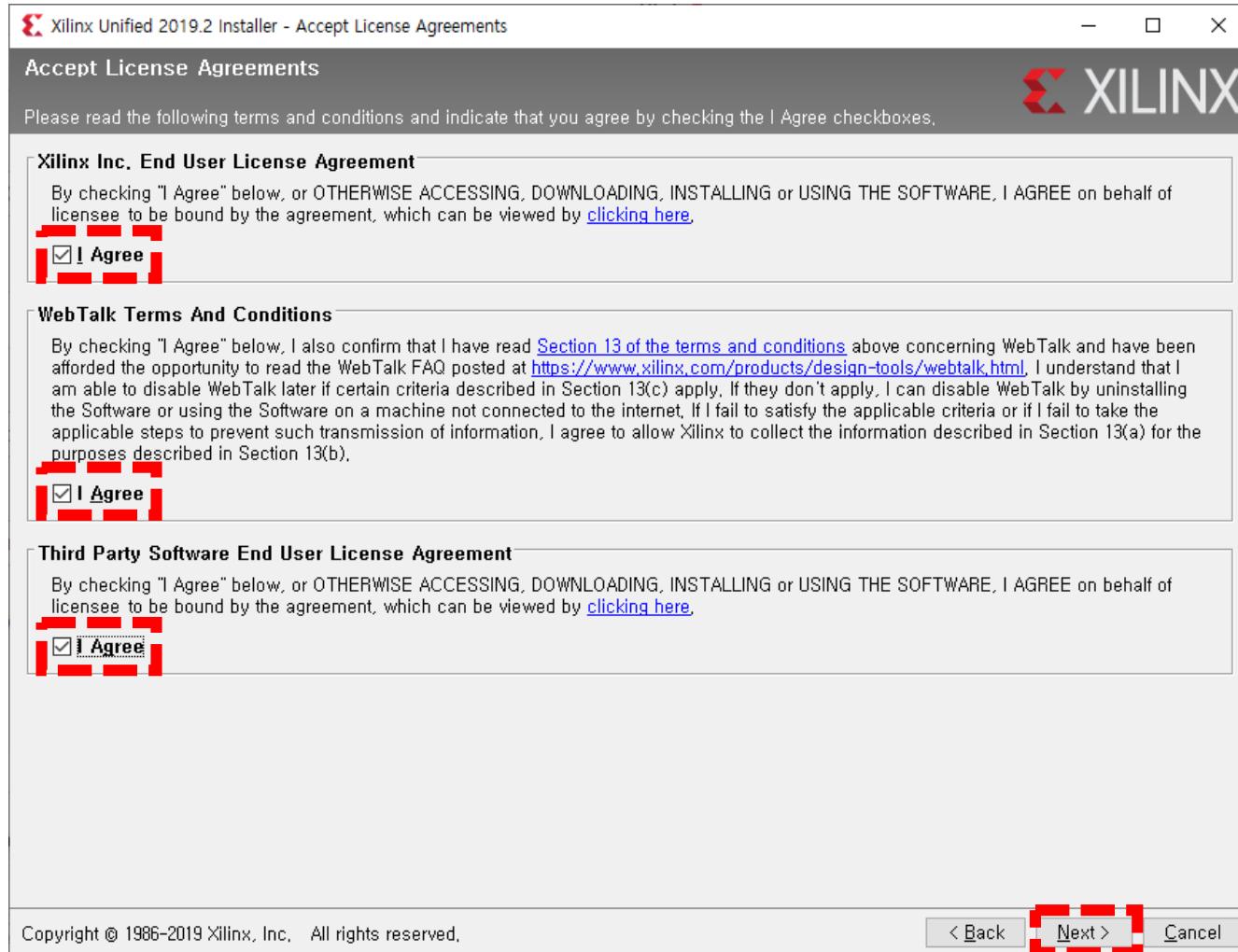
- ❖ 다운로드가 완료되면 다운로드한 설치 파일을 더블 클릭하여 실행한다.
- ❖ 설치 초기화면이 나오면 Next 버튼을 클릭한다.

Step 2 Vivado Installation 2



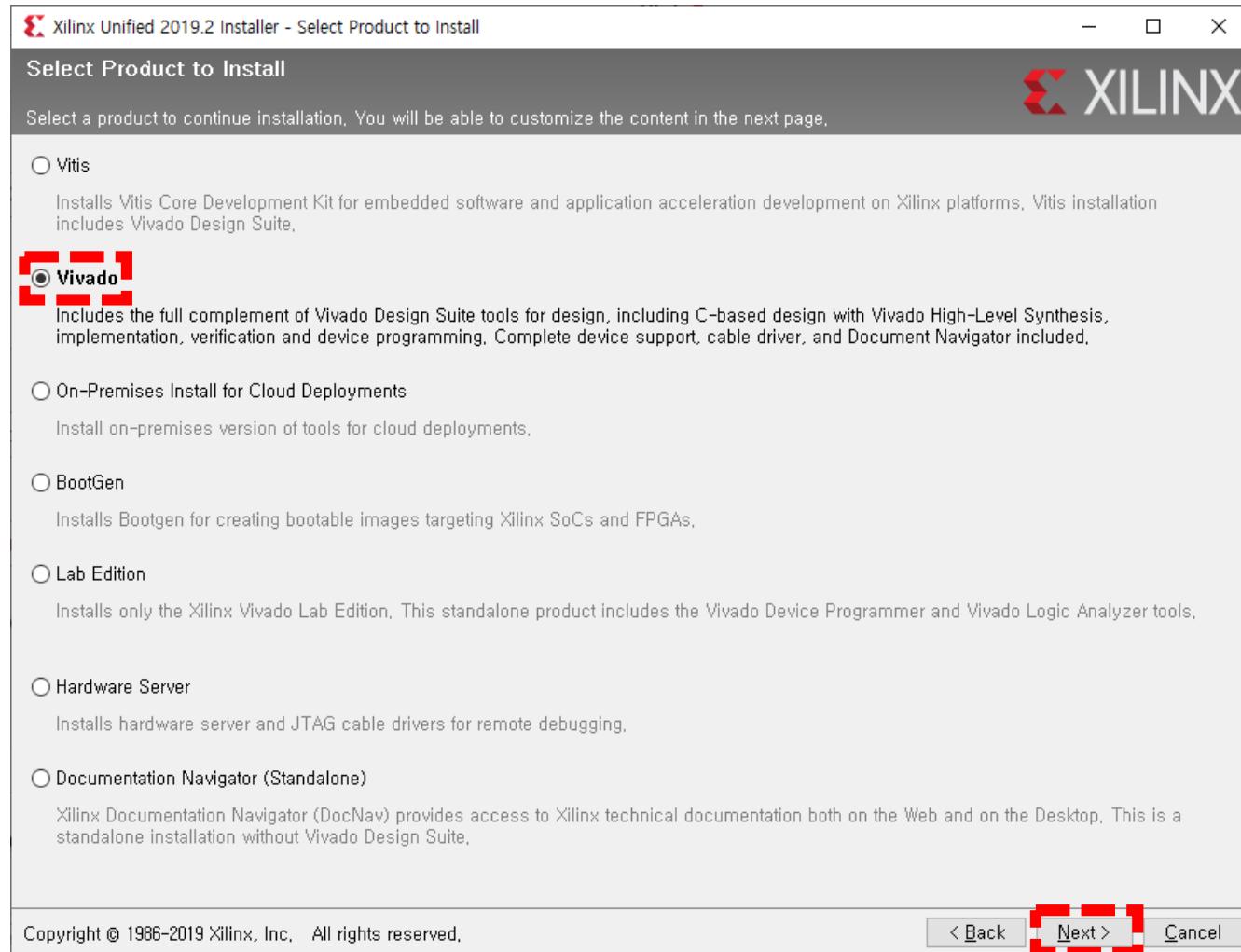
❖ 자일링스 유저임을 확인하는 창이 나오면 자일링스 웹페이지에 가입한 ID와 Password를 입력한 후 Next 버튼을 클릭한다.

Step 2 Vivado Installation 3



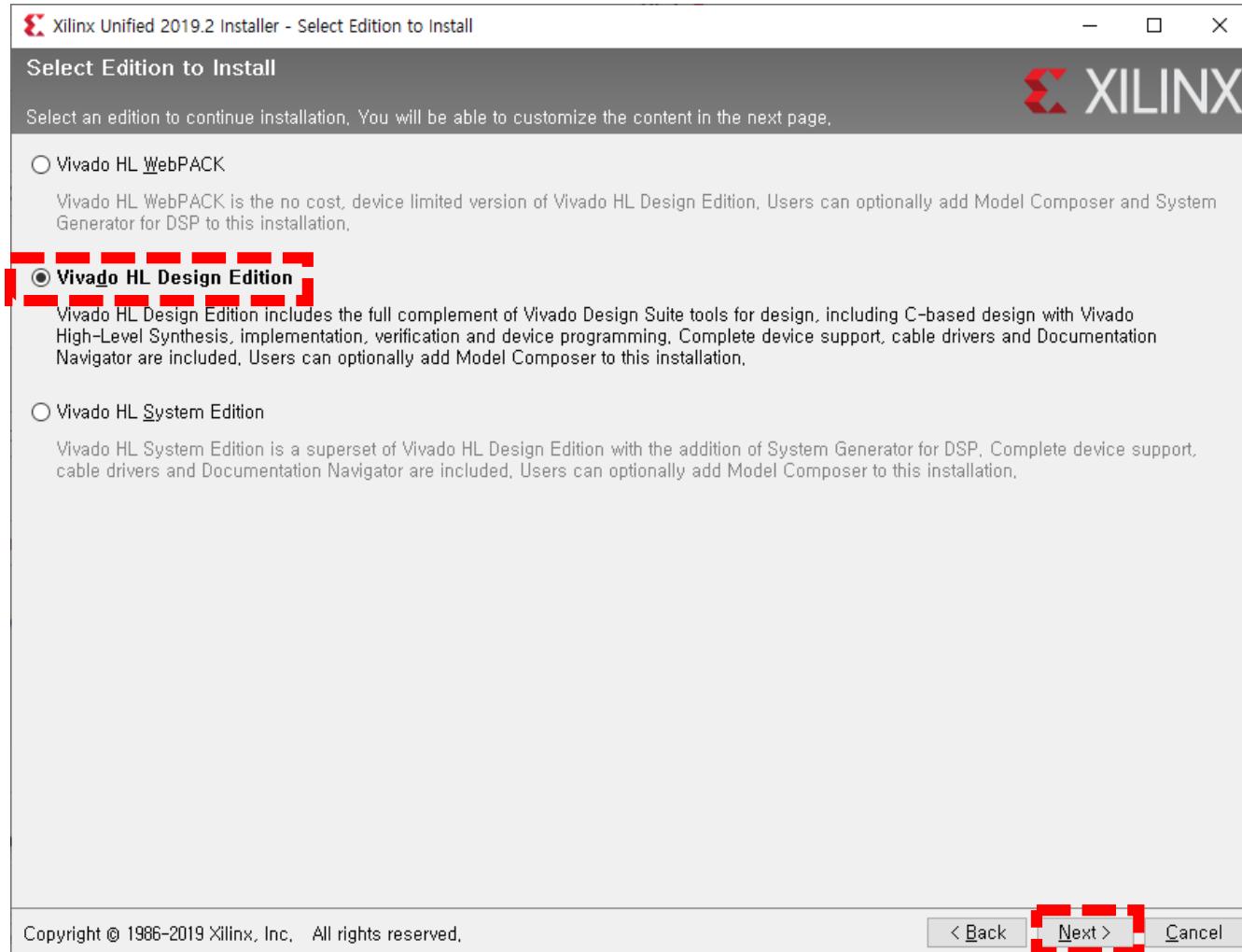
❖ I Agree 체크박스를 모두 선택
후 Next 버튼을 클릭한다.

Step 2 Vivado Installation 4



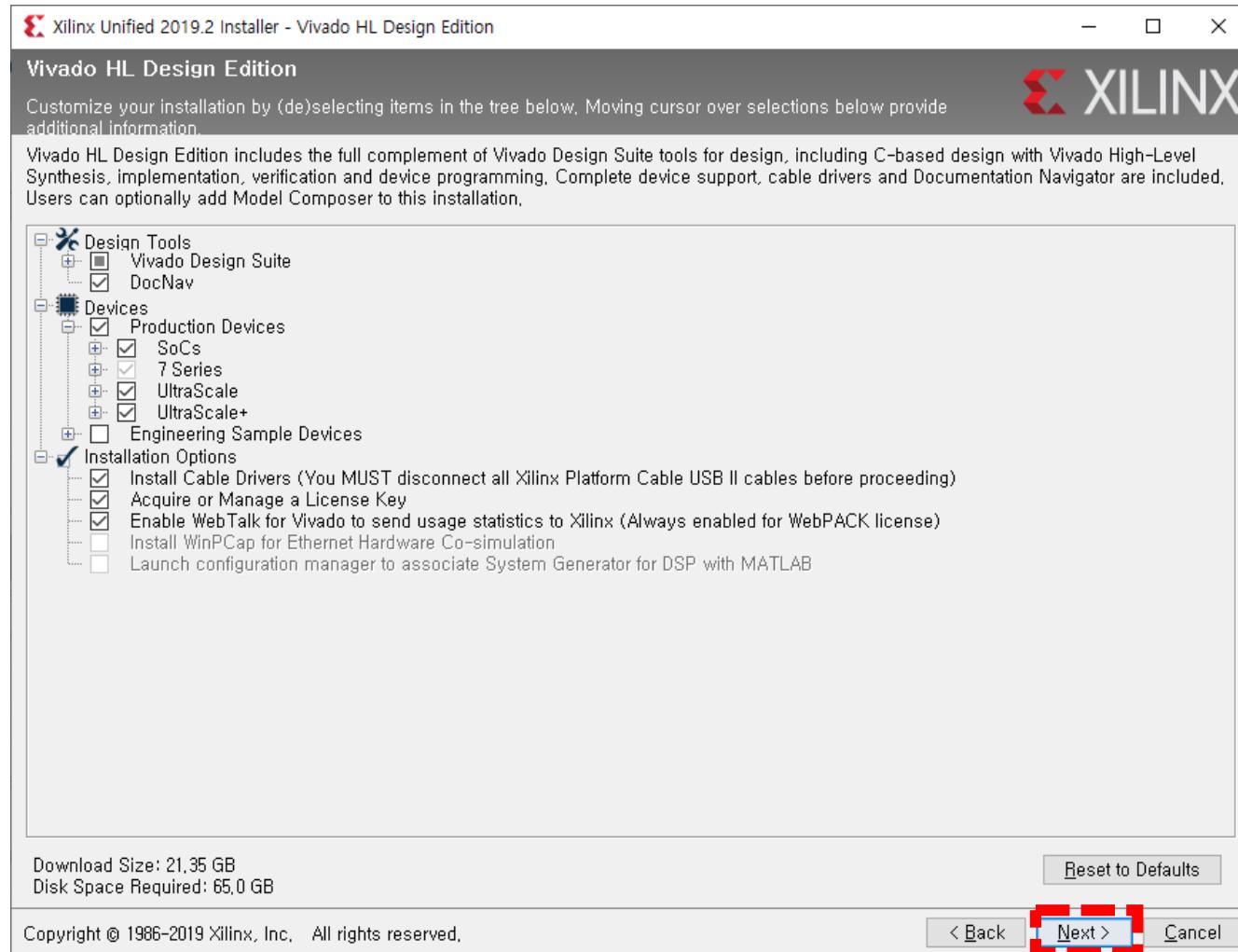
❖ 어떤 툴을 설치 할건지 선택하는 화면이 나오면 Vivado를 선택한 후 Next버튼을 클릭한다.

Step 2 Vivado Installation 5



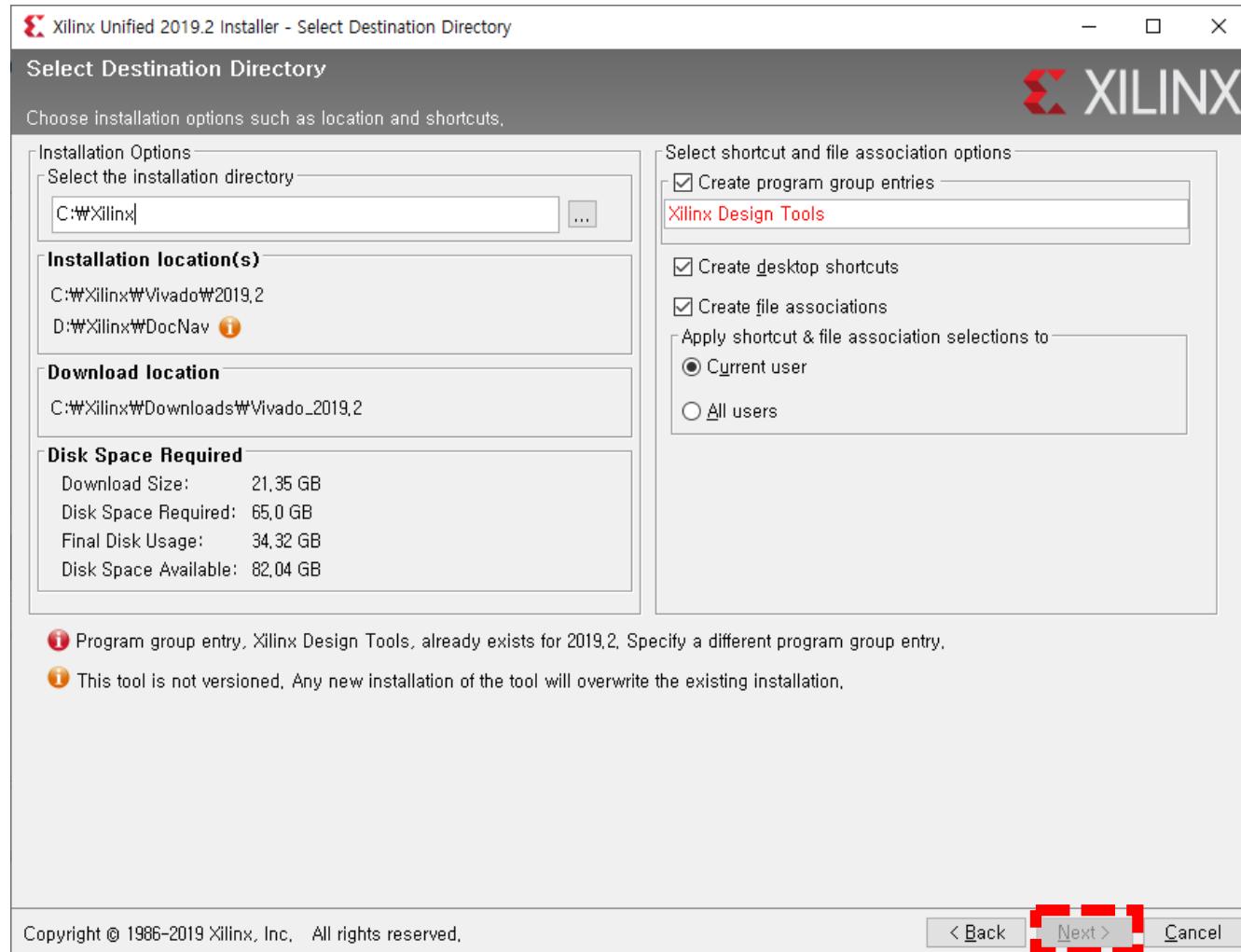
- ❖ 어떤 에디션을 설치 할건지 선택하는 화면이 나오면 Vivado HL Design Edition을 선택한 후 Next 버튼을 클릭한다.
- ❖ Vivado HL Webpack은 무료 버전이고 Vivado HL Design Edition과 System Edition은 유료 버전으로 자일링스로부터 라이선스를 구입하면 Xilinx Product Licensing Site(<https://xilinx.com/getlicense>)를 통해 라이선스를 발급받아서 사용할 수 있다.
- ❖ 본 교육에서는 Ultra96 보드에 포함된 바우처를 사용하여 Vivado HL Design Edition 라이선스를 발급받아 사용한다.

Step 2 Vivado Installation 6



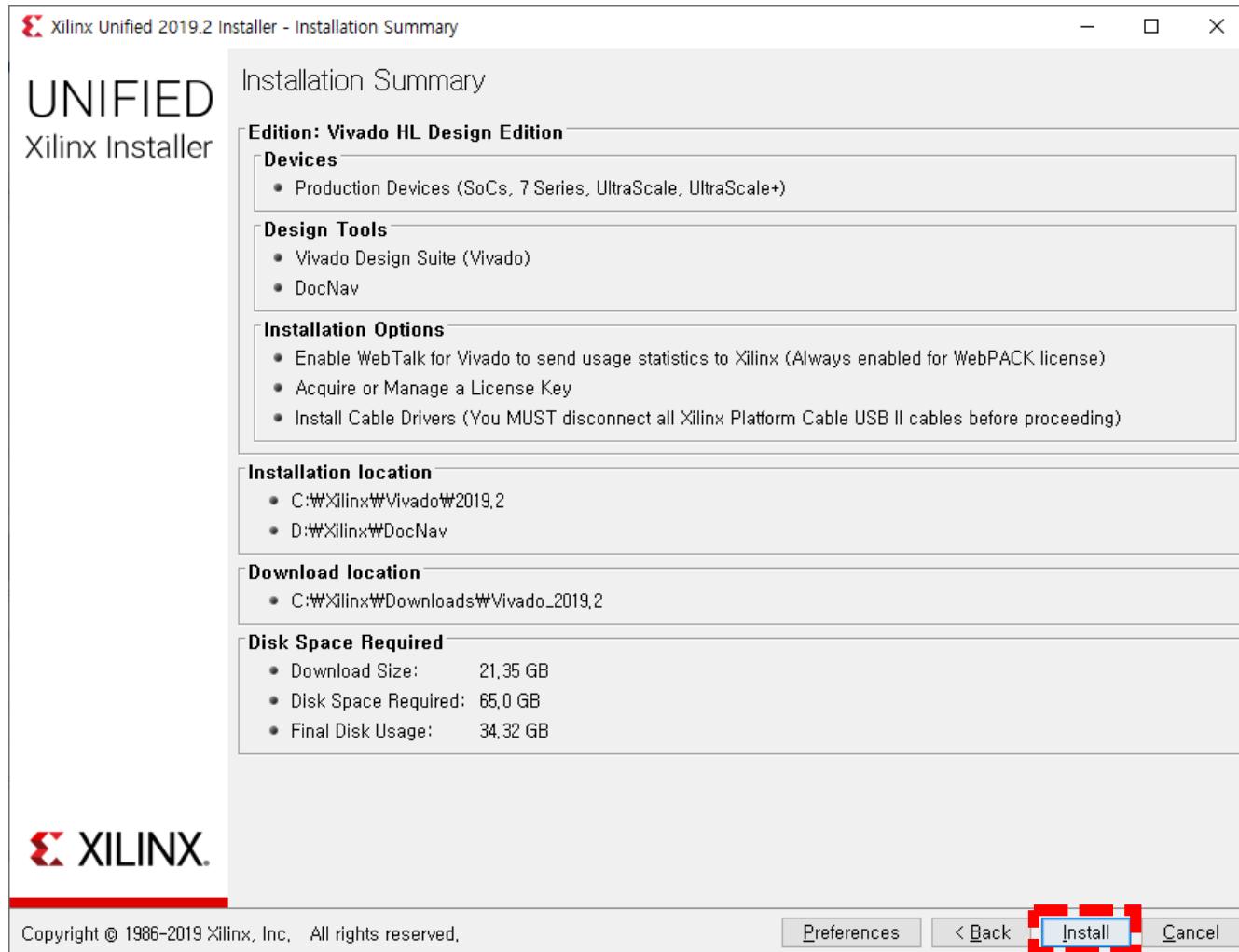
❖ 설치할 툴을 선택하는 화면이 나오면 디폴트 상태 그대로 두고 Next 버튼을 클릭한다.

Step 2 Vivado Installation 7



❖ 툴을 설치할 경로를 선택하는 화면이 나오면 디폴트 상태 그대로 두고 Next 버튼을 클릭한다.

Step 2 Vivado Installation 8

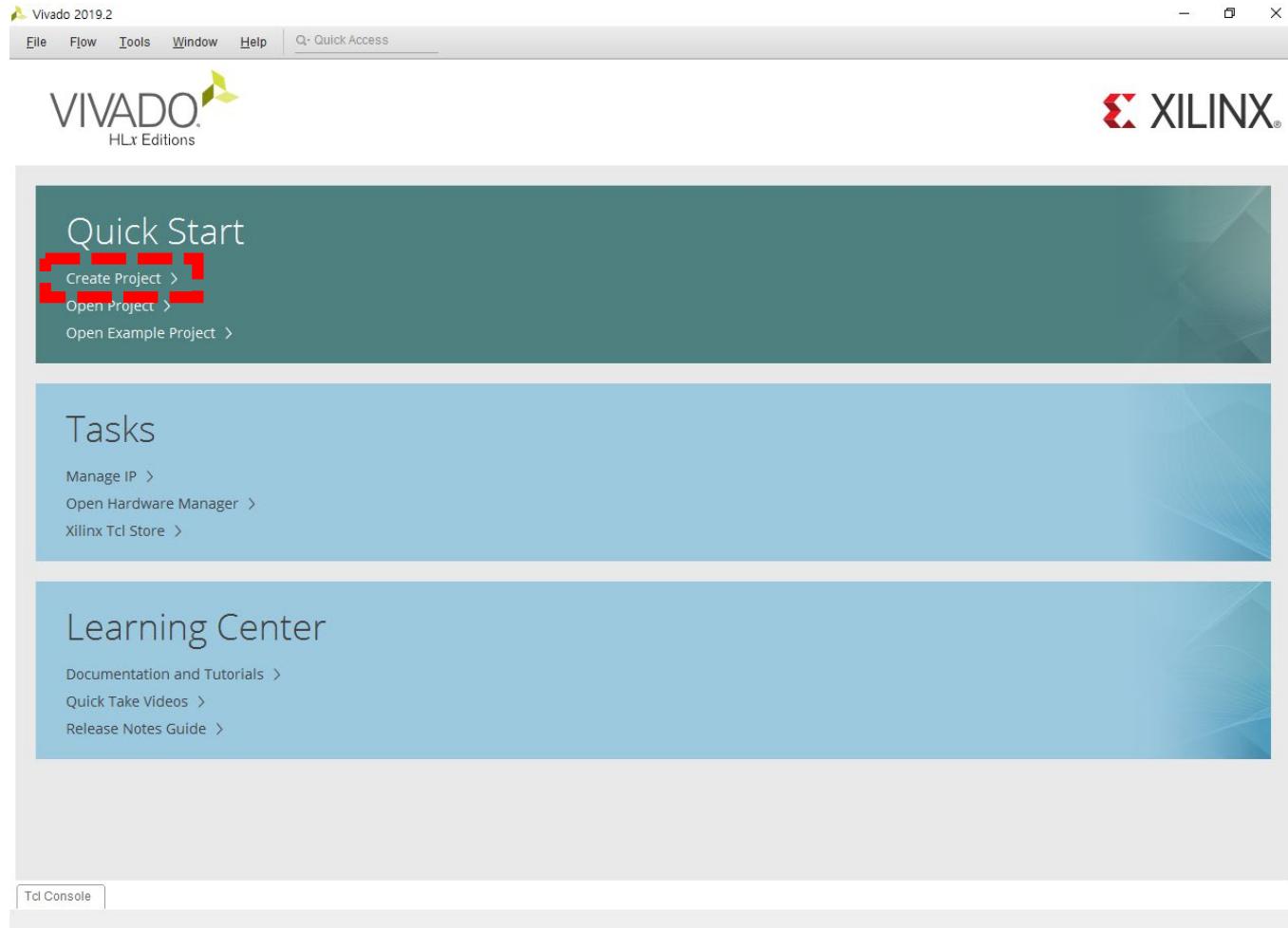


❖ 설치할 내용을 종합한 화면이다. 내용을 살펴보고 특이점이 없으면 Install 버튼을 클릭하여 설치를 시작한다. 설치시간은 컴퓨터 성능과 인터넷 속도에 따라 다르지만 일반적으로 30분에서 1시간정도 소요된다.

Chapter 1 Introduction

- Introduction to Verilog HDL
- Introduction to Xilinx Devices and Tools
- Introduction to Inipro and Ultra96 Training Kit
- Vivado Installation
- Creating Vivado Project

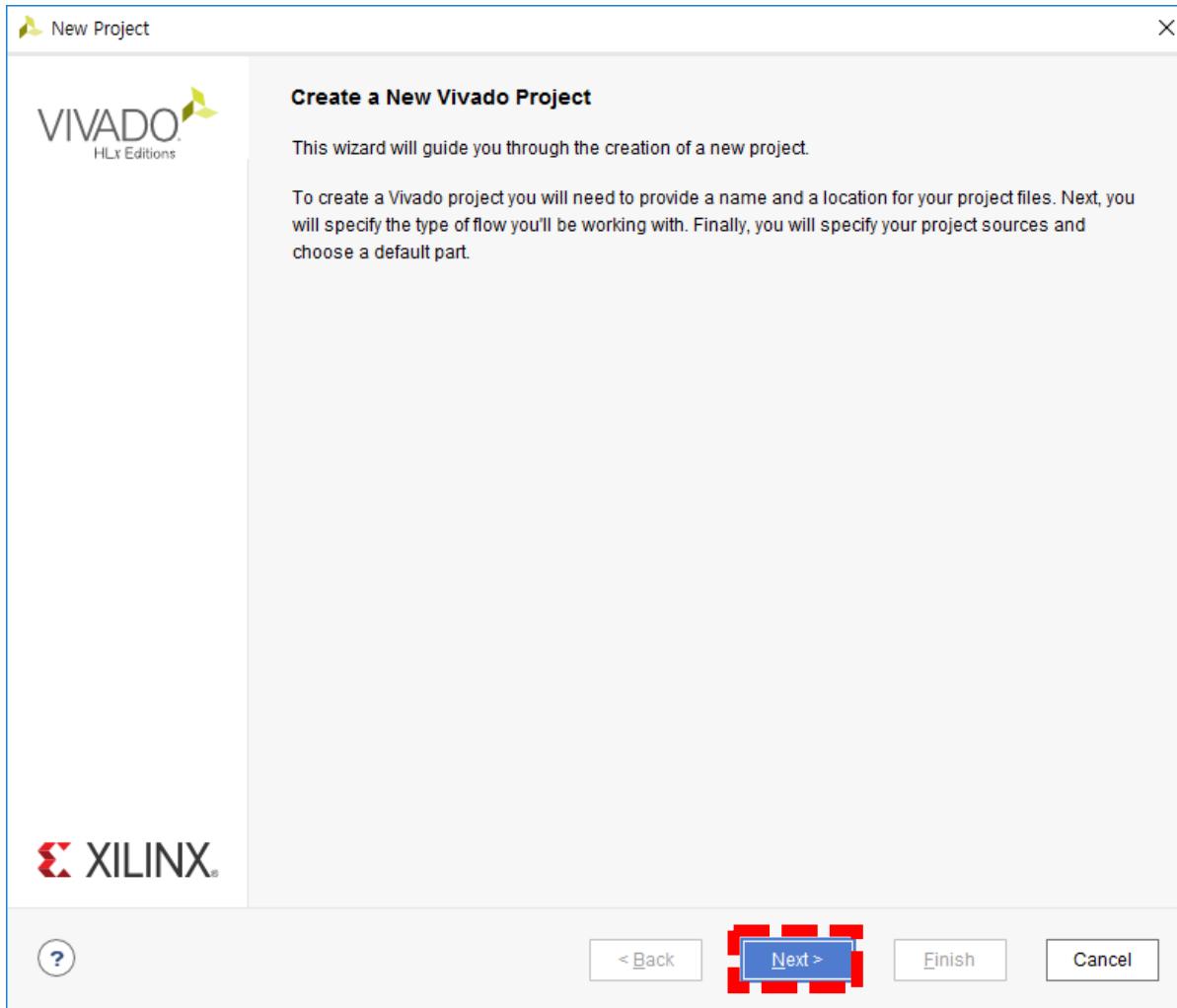
Step 1 Creating Vivado Project 1



❖ 바탕화면 또는 시작화면에서 Vivado를 클릭하여 Vivado를 실행한다.

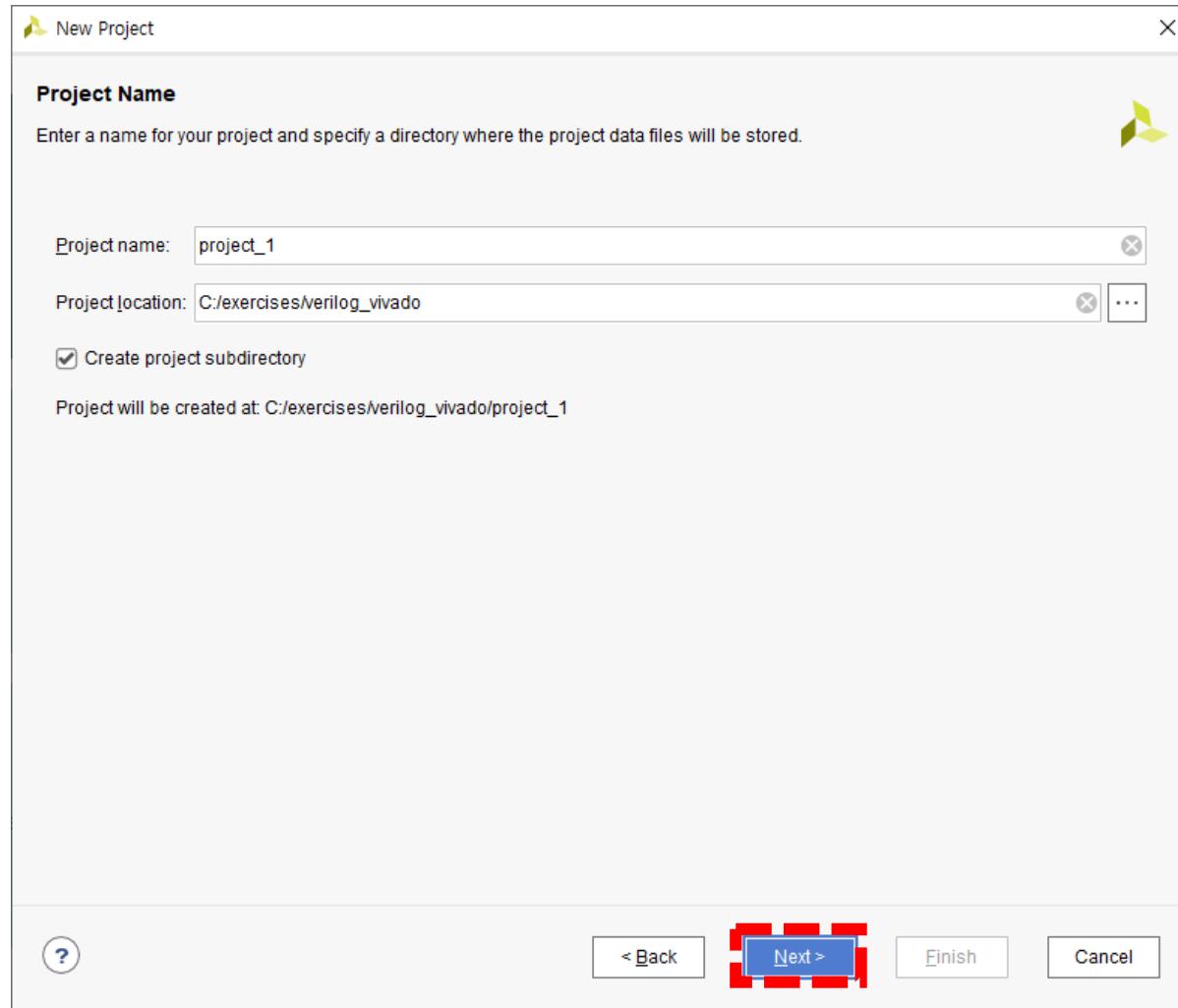
❖ Vivado 실행 초기화면에서 Quick Start 아래 Create Project를 클릭한다.

Step 1 Creating Vivado Project 2



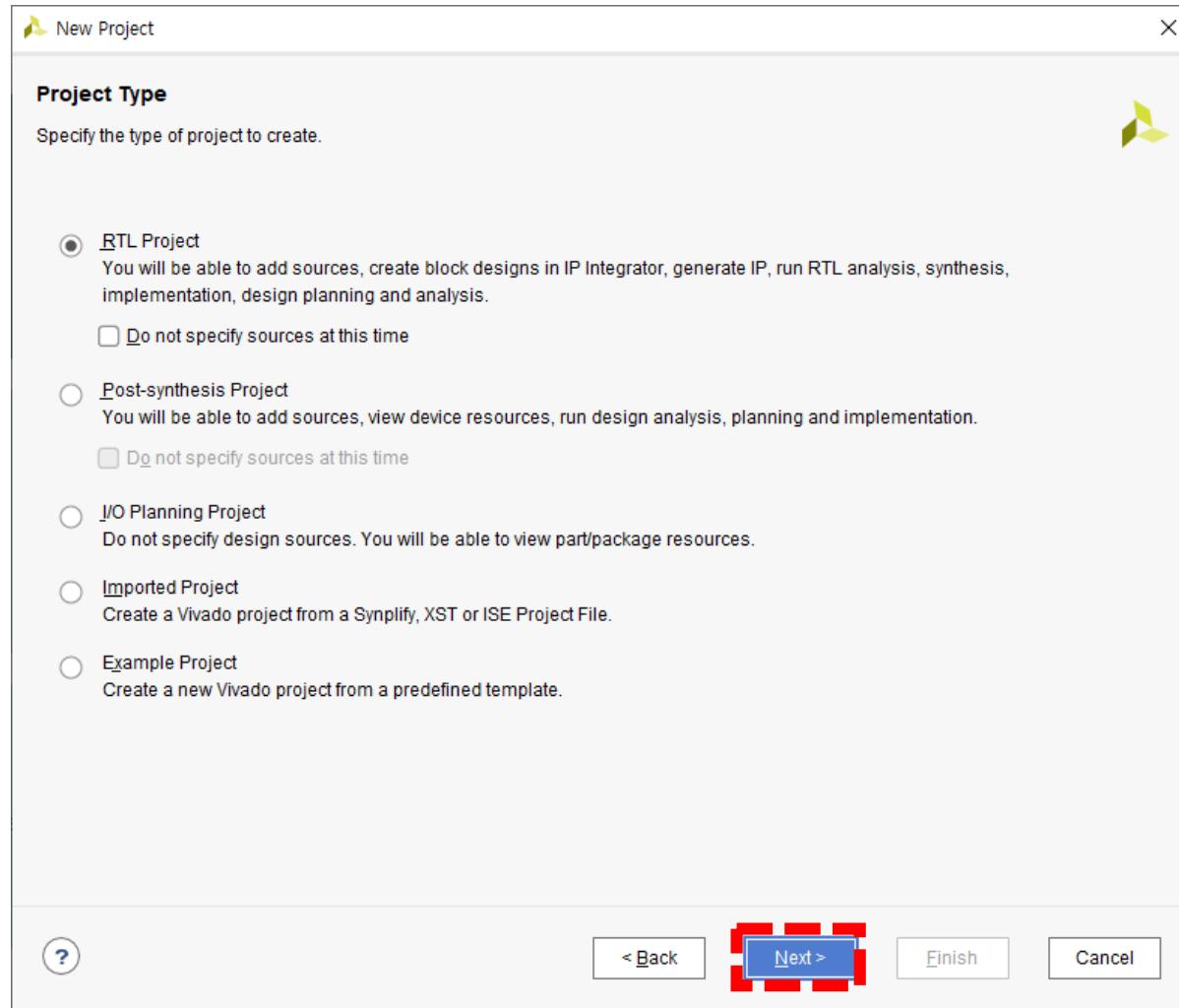
❖ 프로젝트 생성을 위한 초기화
면이 나오면 Next 버튼을 클릭
한다.

Step 1 Creating Vivado Project 3



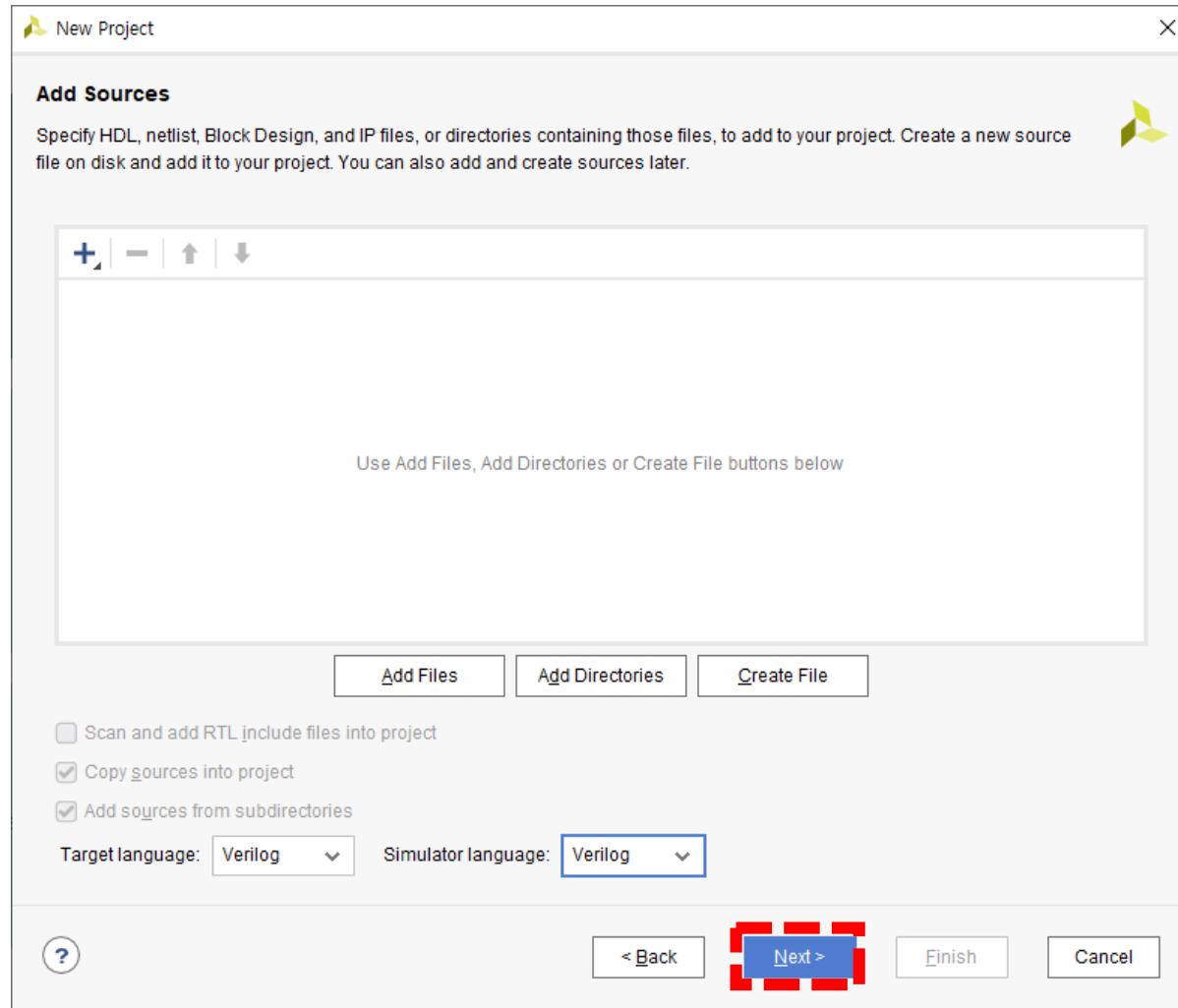
- ❖ 프로젝트 이름과 경로를 선택하는 화면이 나오면 본인이 사용할 프로젝트 경로를 선택하고, 프로젝트 이름은 디폴트 그대로 project_1을 사용한다.
- ❖ 폴더가 생성되어 있지 않으면 폴더를 생성해야 하므로 Create project subdirectory 체크박스를 체크하고, Next 버튼을 클릭한다.

Step 1 Creating Vivado Project 4



- ❖ 프로젝트 타입을 선택하는 화면이 나오면 RTL 설계에 대한 내용을 배울 예정이니 RTL Project를 선택한 후 Next 버튼을 클릭 한다.
- ❖ Do not specify sources at this time 옵션은 소스파일없이 프로젝트를 생성하는 옵션인데 프로젝트 생성단계에서 소스파일을 추가하는 메뉴를 살펴보기 위해서 여기서는 체크를 해제하고 진행하도록 한다.

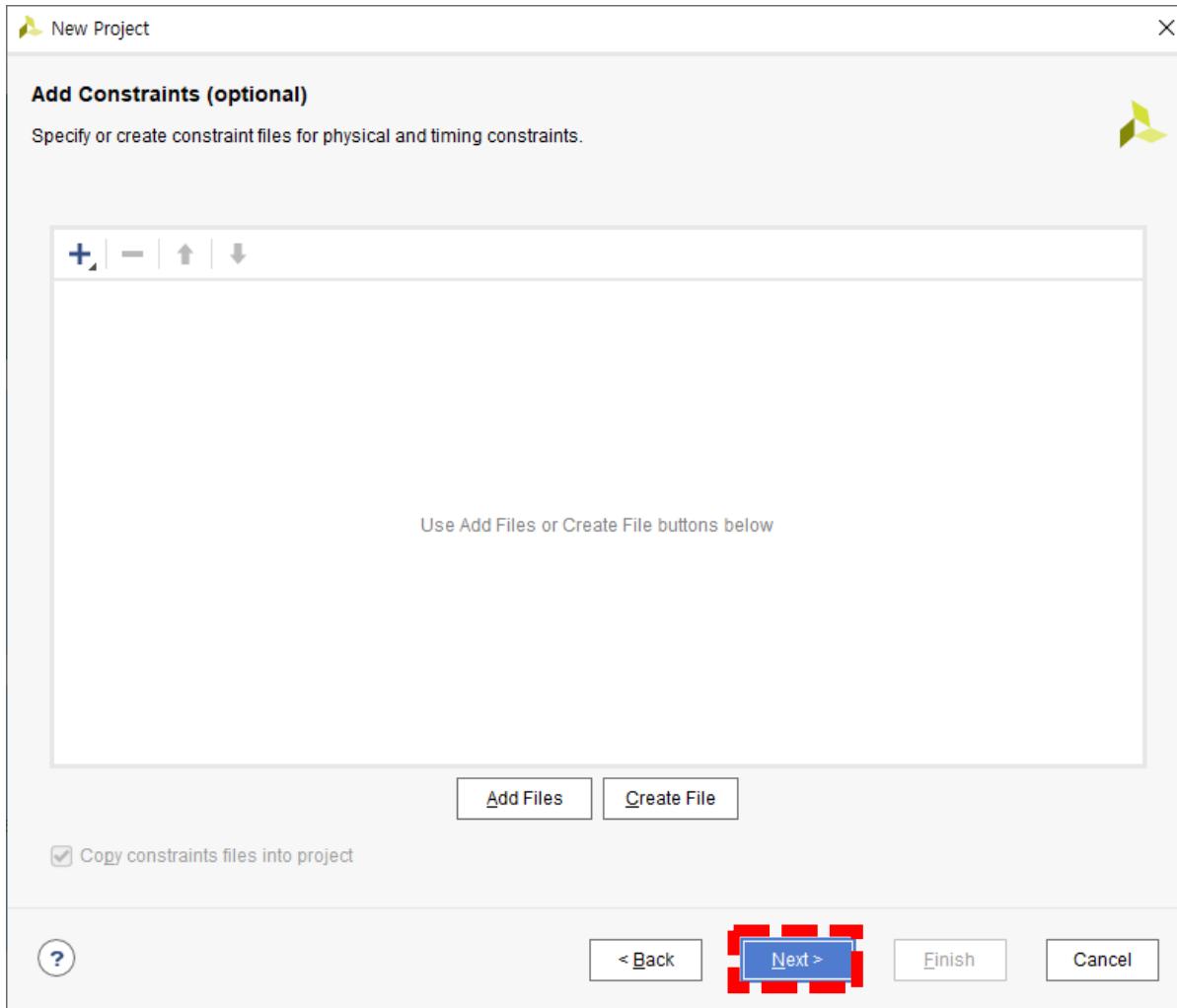
Step 1 Creating Vivado Project 5



❖ 기존에 작성된 소스파일을 프로젝트에 추가하는 화면이 나오면 추가할 소스파일이 없으니 넘어간다.

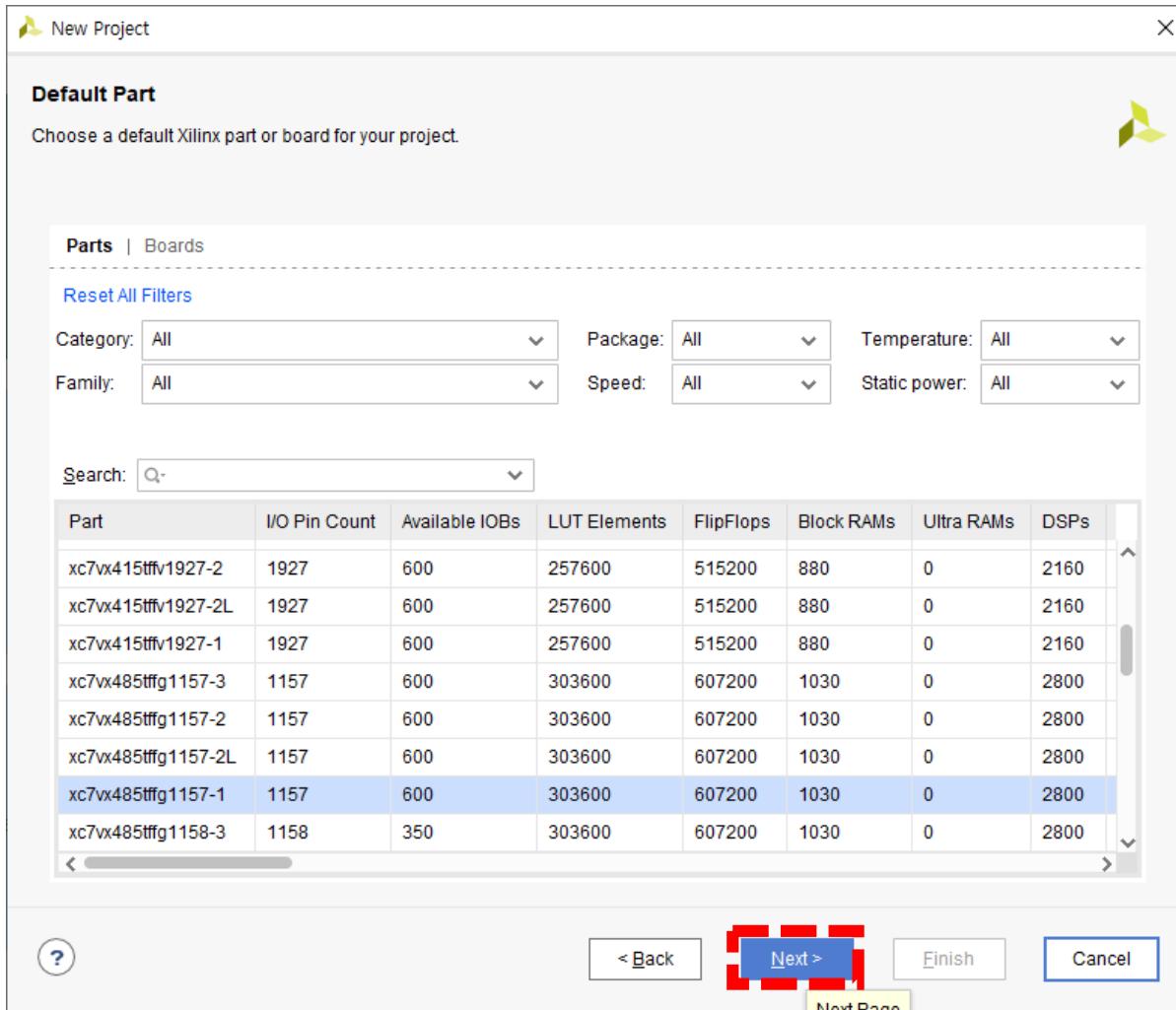
❖ Target language와 Simulator language 옵션에서 그림과 같이 Verilog를 선택하고 Next 버튼을 클릭한다.

Step 1 Creating Vivado Project 6



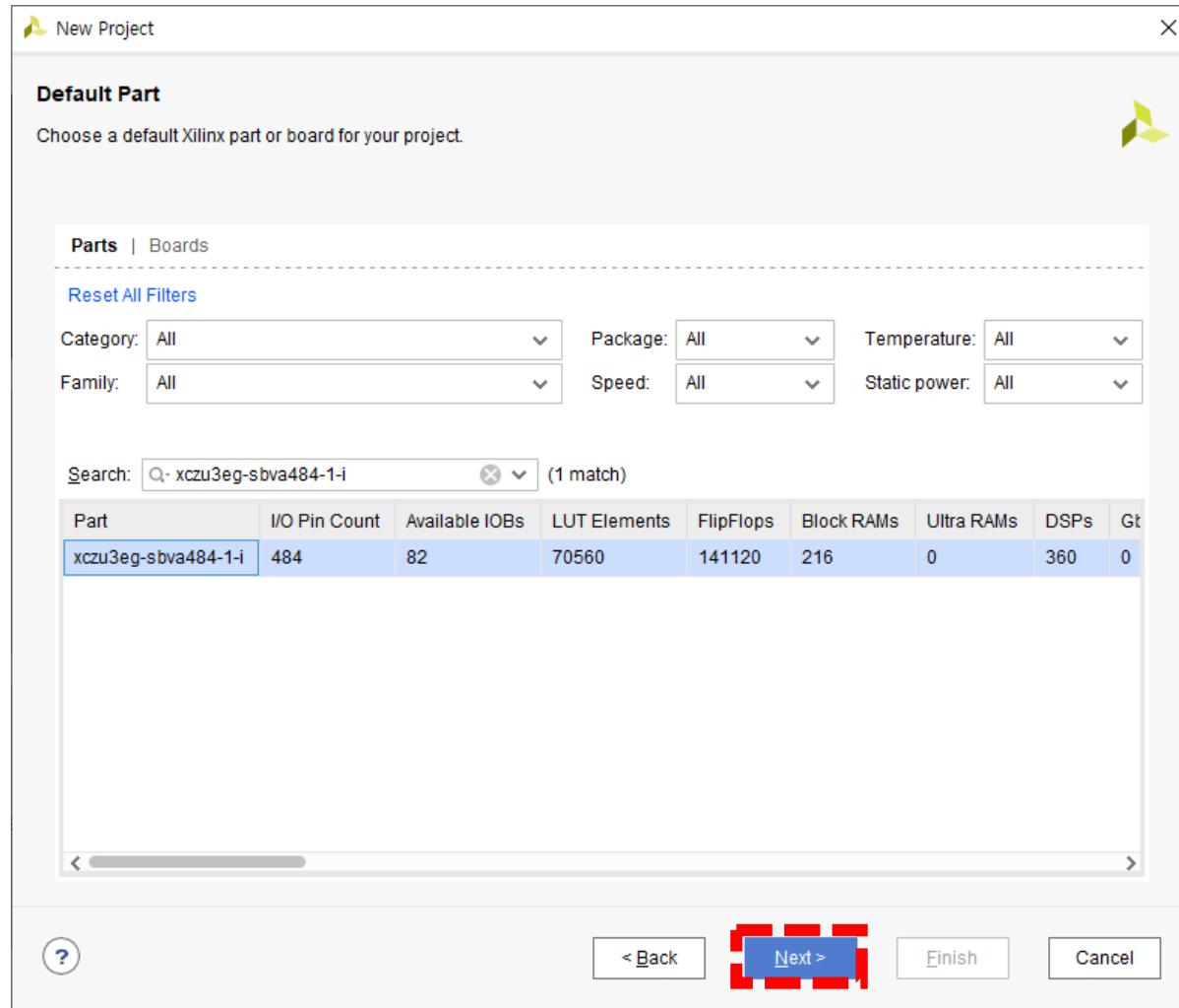
- ❖ Constraints를 추가하는 화면이 나온다.
- ❖ Constraints는 사전적 의미로 제한하다 라는 말로써 FPGA 설계에 필요한 제한조건을 만들기 위한 것이다.
- ❖ 예를 들어, 가장 기본적인 Constraints에는 Pin Constraints가 있는데 Pin Constraints란 FPGA Device에 HDL을 사용하여 설계한 하드웨어의 입출력 포트가 FPGA Device의 어떤 Pin을 사용할지 지정하는 것이다.
- ❖ 본 교육에서는 Ultra96 보드를 사용할 예정이니 Ultra96 보드에 맞게 Pin Constraints를 해주어야 하지만 여기에서는 기존에 작성된 Constraints 파일이 없으니 Next 버튼을 클릭하여 다음 단계로 넘어간다.

Step 1 Creating Vivado Project 7



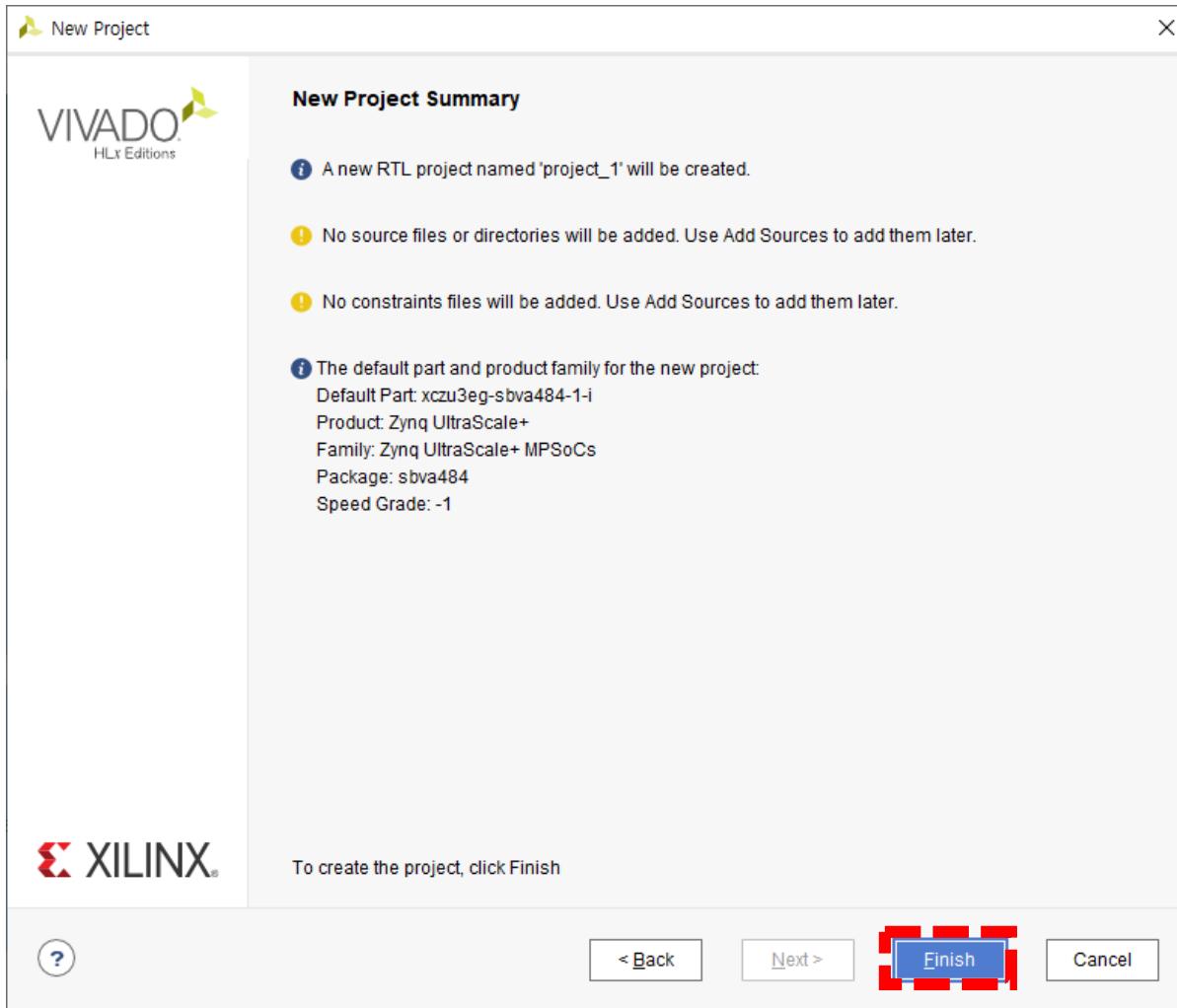
❖ 어떤 FPGA Device를 사용할지 선택하는 화면이 나오면 디바이스를 선택하기 위한 여러 가지 방법이 있지만, Search 창을 통해 디바이스의 파트번호를 입력하면 쉽게 선택할 수 있다.

Step 1 Creating Vivado Project 8



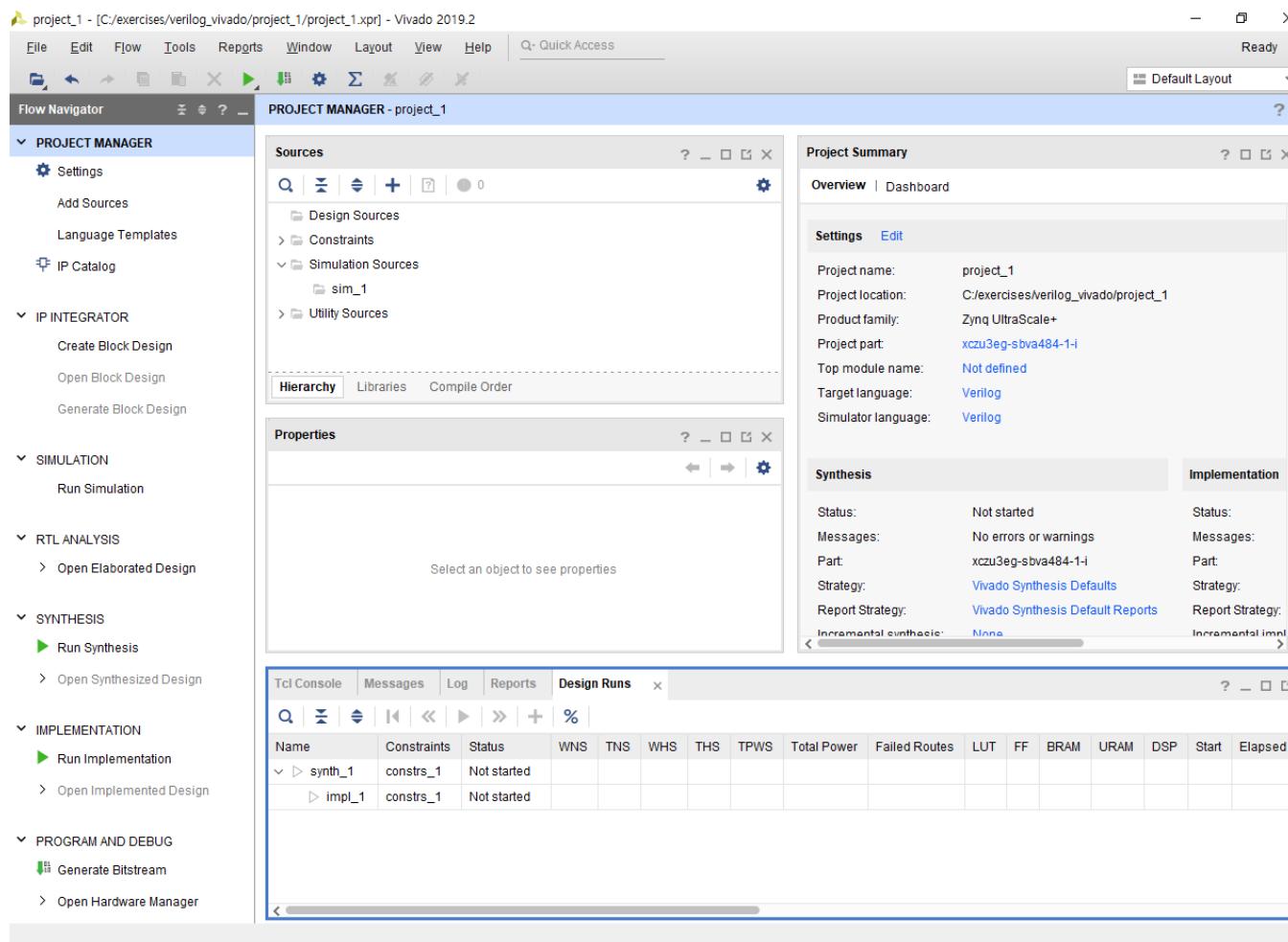
- ❖ Ultra96 보드에 탑재된 Device 인 xczu3eg-sbva484-1-i를 입력하면 디바이스가 하나만 남게 된다.
- ❖ 디바이스를 선택한 후 Next 버튼을 클릭한다.

Step 1 Creating Vivado Project 9



❖ Project Summary 화면이 나오면 프로젝트를 잘 만들었는지 내용 확인 후 Finish 버튼을 클릭하면 프로젝트 만들기가 완료된다.

Step 1 Creating Vivado Project 10



❖ 프로젝트 생성을 완료한 화면
이다.

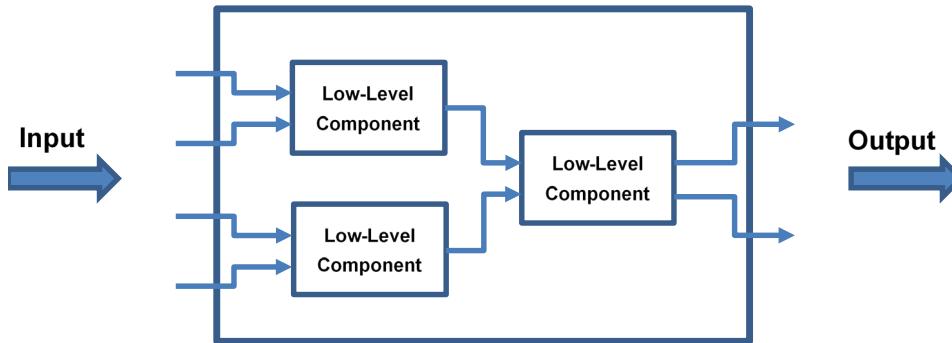
Chapter 2 Verilog HDL Basics

- **Hardware Modeling**
- Verilog HDL Basic Structure
- continuous assignment
- Vivado Design Flow

Hardware Modeling

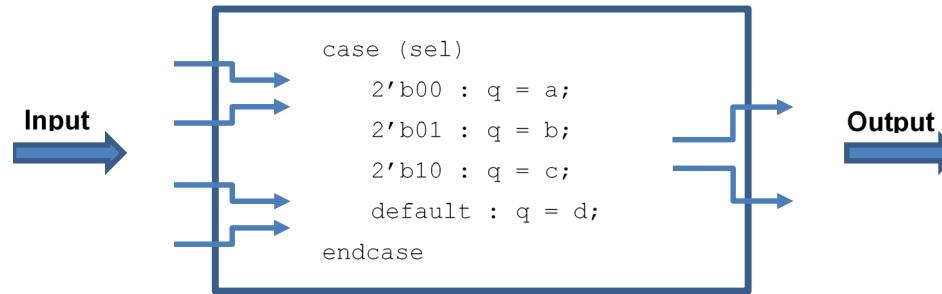
- ❖ Hardware Modeling이란 특정 하드웨어를 구현하기 위한 방법을 말한다.
- ❖ 이런 하드웨어 모델링 기법에는 Structural Modeling과 Behavioral Modeling의 두 가지 기법이 있다.

Structural Modeling



- ❖ Structural Modeling은 구조적으로 하드웨어를 모델링 한다는 의미로 하드웨어를 구성하는 부품, 소자 또는 모듈이라고 불리는 Low-Level Component 들을 서로 연결하여 하드웨어를 표현하는 기법을 말한다.
- ❖ 부품(Component) 소자(Element) 모듈(Module) 은 모두 같은 의미로 상위 레벨의 하드웨어를 구성하기 위한 하위 레벨의 하드웨어를 의미한다.

Behavioral Modeling



- ❖ Behavioral Modeling은 하드웨어가 어떻게 행동하는지를 소프트웨어 언어 표현 방식으로 표현하는 기법을 말한다.
- ❖ 소프트웨어 언어로 하드웨어가 어떻게 행동하는지를 표현하면 합성 툴(Synthesis Tool)이 동일하게 동작하는 하드웨어를 만들어 준다.
- ❖ 합성 툴은 소프트웨어를 컴퓨터가 알아들을 수 있는 기계어로 번역해 주는 컴파일러와 같이 하드웨어를 표현한 코드를 실제 하드웨어로 구성할 수 있는 소자들로 바꾸어 주는 툴이다.
- ❖ 예를 들어, 위 그림에서는 하드웨어가 어떻게 동작하는지를 case문을 사용하여 표현하였고 이 코드를 합성하면 표현한 대로 동작하는 하드웨어를 만들어 주는데 이와 같은 하드웨어 모델링 기법을 Behavioral Modeling이라고 부른다.

Chapter 2 Verilog HDL Basics

- Hardware Modelings
- Verilog HDL Basic Structure
- continuous assignment
- Vivado Design Flow

Verilog HDL Basics

- ❖ Verilog는 대소문자를 다르게 인식한다. (※ 대문자 A로 선언한 것과 소문자 a로 선언한 것은 다른 것으로 인식하며 참고로 VHDL은 대소문자를 같은 것으로 인식한다.)
- ❖ Verilog의 모든 예약어들은 소문자로 작성해야 한다.
- ❖ 문장의 끝은 ; (세미콜론)으로 끝낸다.
- ❖ 주석처리는 C언어와 같이 한 줄 주석은 // 여러 줄의 주석은 /* */을 사용한다.

Verilog HDL Basic Structure 1

- ❖ Verilog로 하드웨어의 기본 형태인 모듈을 표현할 때 module로 시작하여 endmodule로 끝낸다.
- ❖ module 뒤에는 모듈이름을 쓴다. (※ 설계할 하드웨어의 특징을 규정할 수 있는 이름으로 이름을 짓는 것이 좋다.)
- ❖ 모듈이름 뒤에는 모듈의 입출력포트 리스트를 작성한다. (※ 다른 하드웨어와 연결될 포트를 열거하는 부분이다.)
- ❖ 모듈의 첫 라인은 외부에서 이 모듈을 볼 때 볼 수 있는 모듈의 이름과 입출력포트를 명시한다.

```
module 모듈이름 ( 포트리스트 );
    포트 선언부
    데이터 타입 선언부
    회로 구현부
endmodule
```

Verilog HDL Basic Structure 2

- ❖ 포트 선언부와 데이터 타입 선언부는 하드웨어가 서로 연결될 와이어(넷)를 선언하는 부분이다.
- ❖ 포트 선언부는 이 모듈과 외부 회로가 연결될 포트를 선언하는 곳이고 데이터 타입 선언부는 모듈내에서 연결될 와이어가 필요한 경우 선언하는 곳이다.
- ❖ 선언부 뒤에 작성되는 회로 구현부에는 포트로 선언된 와이어와 데이터타입으로 선언된 와이어가 서로 어떻게 연결되어 있는지 또는 어떤 하위레벨의 모듈이 연결되어 있는지를 표현하여 회로를 구현하는 부분이다.

```
module 모듈이름 ( 포트리스트 );
    포트 선언부
    데이터 타입 선언부
    회로 구현부
endmodule
```

Chapter 2 Verilog HDL Basics

- Hardware Modelings
- Verilog HDL Basic Structure
- continuous assignment
- Vivado Design Flow

continuous assignment 1

- ❖ Verilog assign문은 좀 더 정확하게 continuous assignment라고 부른다.
- ❖ Verilog에서 assign문은 와이어를 연결하는 데 사용한다.
- ❖ CAD(Computer aided Design)툴에서 회로를 설계할 때 마우스를 사용하여 선을 연결 하듯이 Verilog에서는 assign문을 사용하여 선을 연결한다.
- ❖ 위 코드는 a, b 두 개의 포트로 들어오는 신호를 AND 연산하여 result 포트로 출력하는 ander라는 모듈이다.

```
`timescale 1 ns / 1 ps

module ander (a, b, result);

    input a, b;
    output result;

    assign result = a & b;

endmodule
```

continuous assignment 2

- ❖ 첫 번째 줄에 나오는 timescale 문은 순서대로 reference time unit과 resolution을 정하기 위한 것으로 시뮬레이션을 할 때 기준이 되는 시간을 설정하기 위한 문장이다. (※ 이에 대한 자세한 내용은 뒤에 시뮬레이션 챕터에서 다루도록 한다.)
- ❖ 두 번째 문장은 모듈이름과 포트리스트를 표현한 것이다. (※ 은 ander라는 이름을 가지고 있고 a, b, result라는 세 개의 포트를 가진 모듈이다.)
- ❖ 그 다음 문장들은 포트 선언부로 a, b는 입력포트로 result는 출력포트로 각각 선언한 것이다.
- ❖ 그 다음 문장의 assign 문은 선(wire)을 연결하기 위한 것이다. (※ ander 모듈은 a와 b 포트로 입력되는 신호를 AND하여 result 포트로 출력하는 모듈이 된다.)

```
'timescale 1 ns / 1 ps

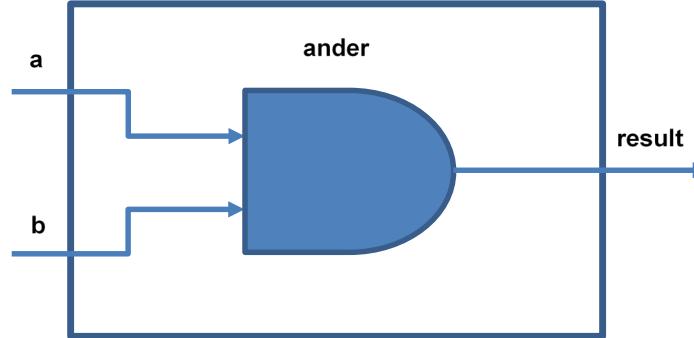
module ander (a, b, result);

    input a, b;
    output result;

    assign result = a & b;

endmodule
```

continuous assignment 3

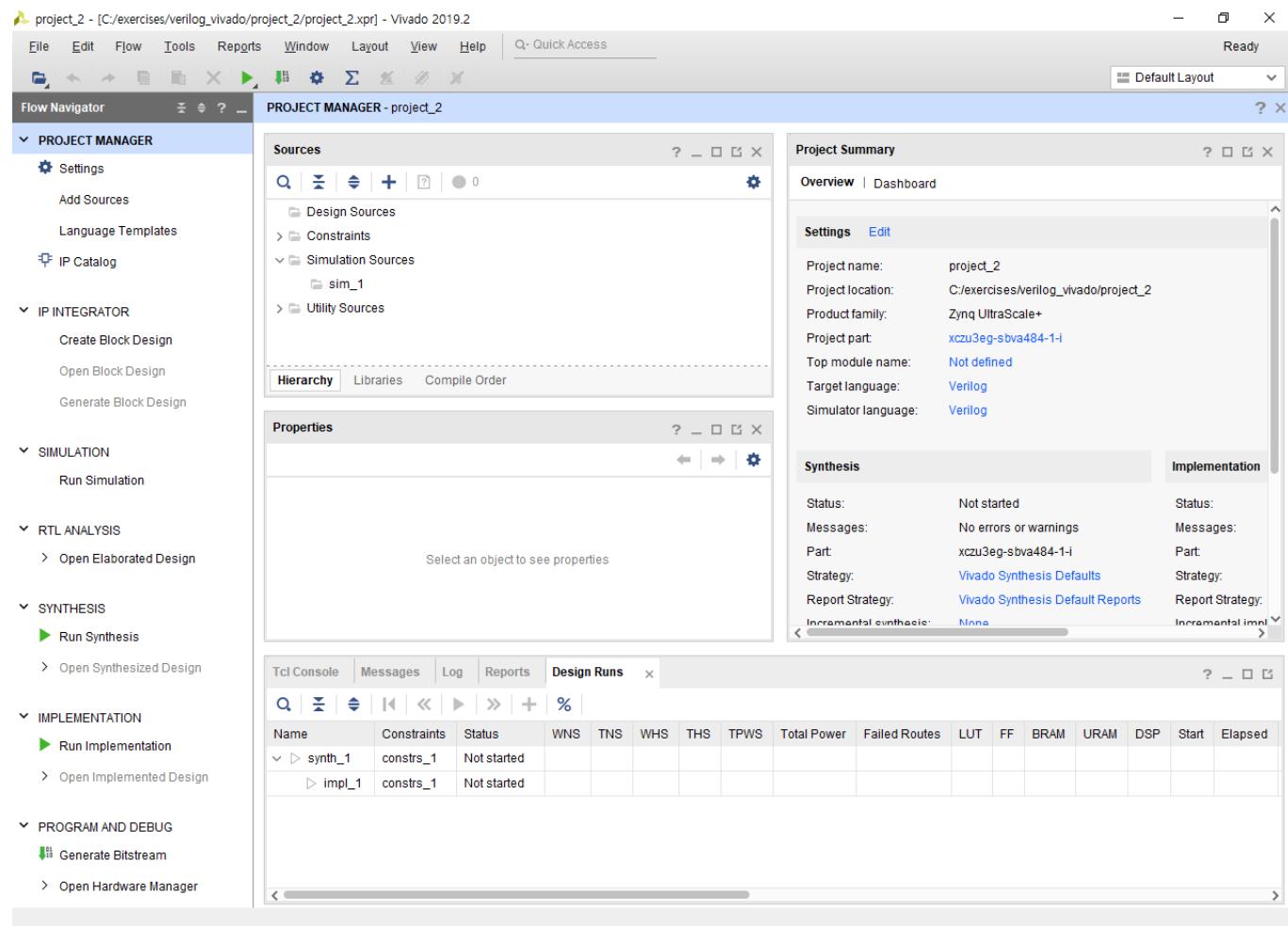


- ❖ a와 b포트는 AND 게이트의 입력 포트로 연결되고 result 포트는 AND게이트의 출력 포트로 연결된다.
- ❖ ander 모듈은 위 그림과 같은 회로가 된다.
- ❖ assign 문은 와이어를 연결하기 위해 사용한다.

Chapter 2 Verilog HDL Basics

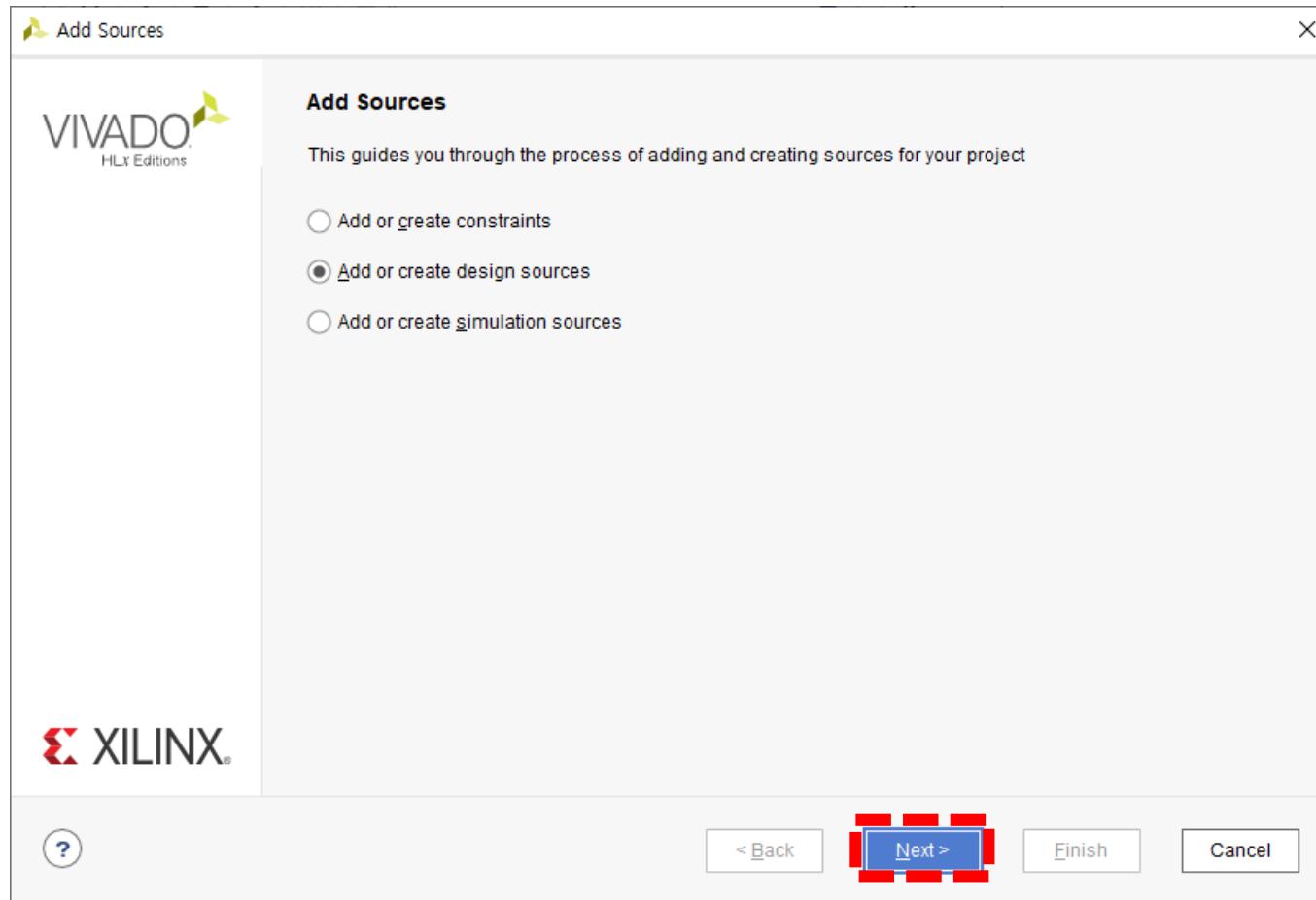
- Hardware Modelings
- Verilog HDL Basic Structure
- continuous assignment
- Vivado Design Flow

Step 1 Creating Vivado Project



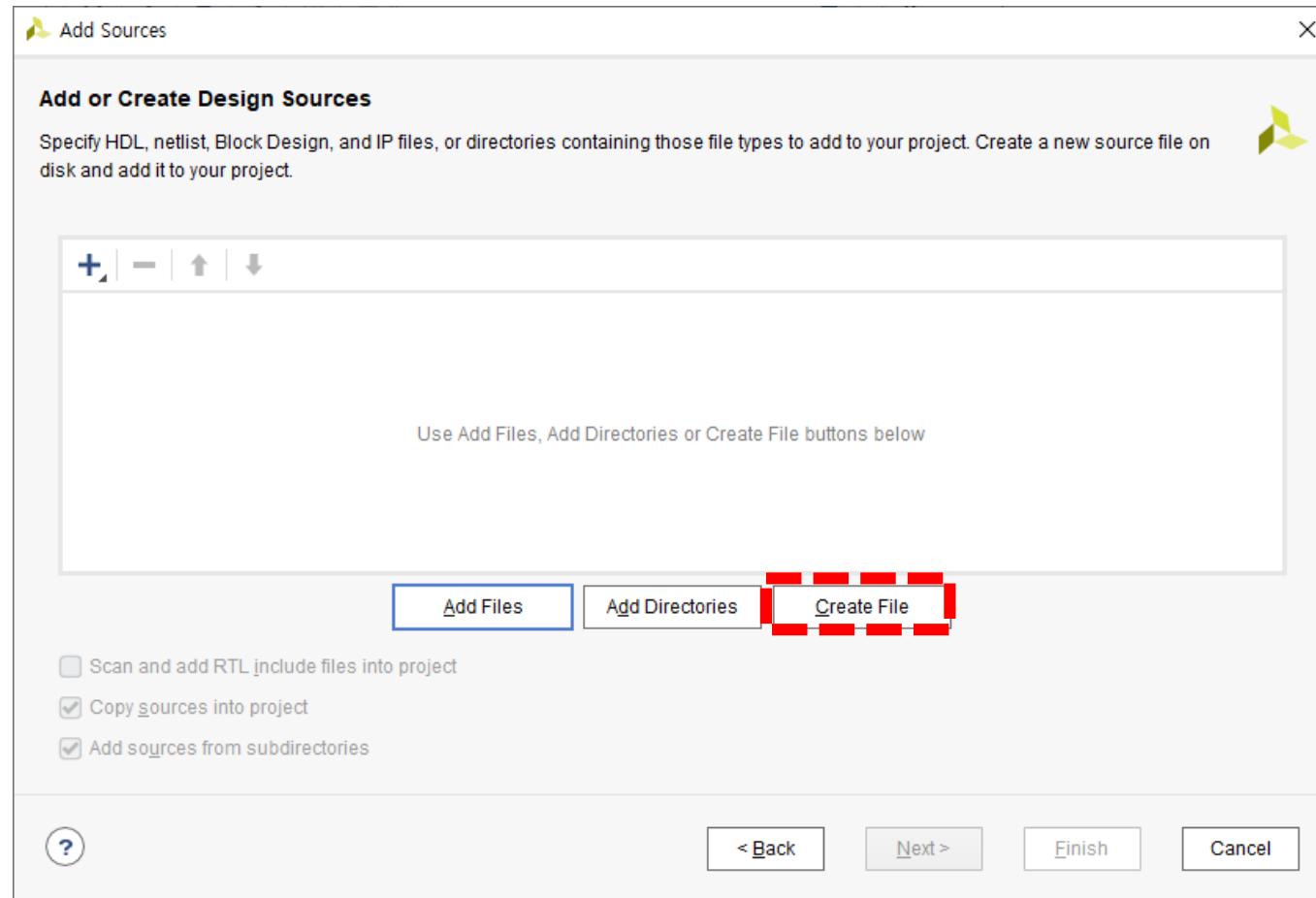
❖ Chapter 1의 Vivado 프로젝트 만들기를 참조하여 project_2 프로젝트를 만든다.

Step 2 Add Source in Vivado Project 1



- ❖ Flow Navigator ⇒ Project Manager ⇒ Add Sources를 클릭한다.
- ❖ Add Sources 창이 나오면 Add or create design sources를 선택 후 Next 버튼을 클릭 한다.

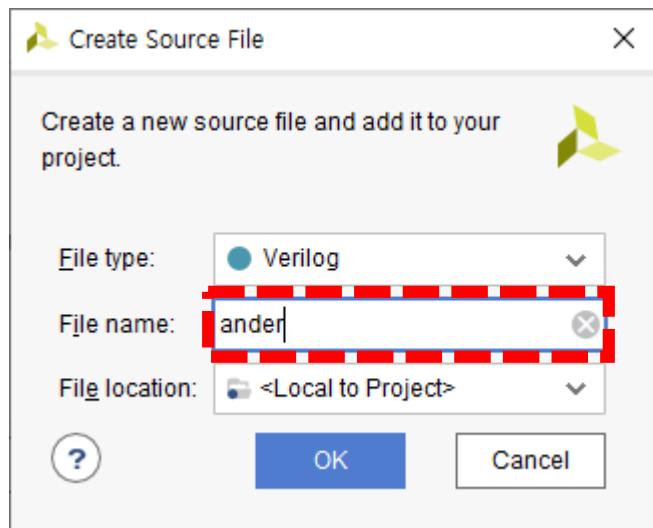
Step 2 Add Source in Vivado Project 2



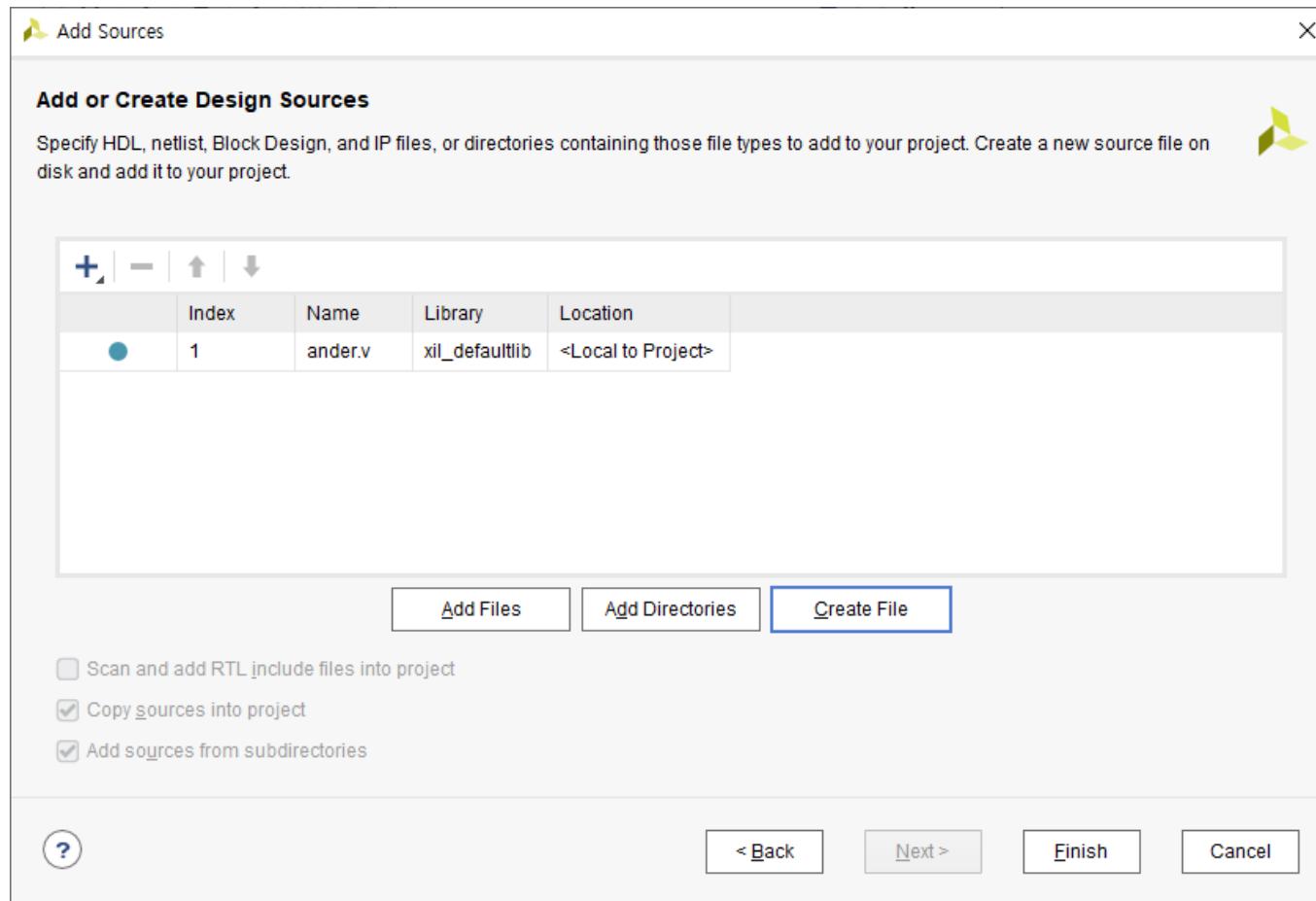
❖ 새로운 파일 생성하여 프로젝트에 추가하기 위해 Create File 버튼을 클릭한다.

Step 2 Add Source in Vivado Project 3

❖ 생성할 파일 타입은 Verilog를 선택하고 이름을 작성한다. 여기서는 ander 모듈을 생성할 예정이니 이름을 ander라고 작성하고 OK 버튼을 클릭한다

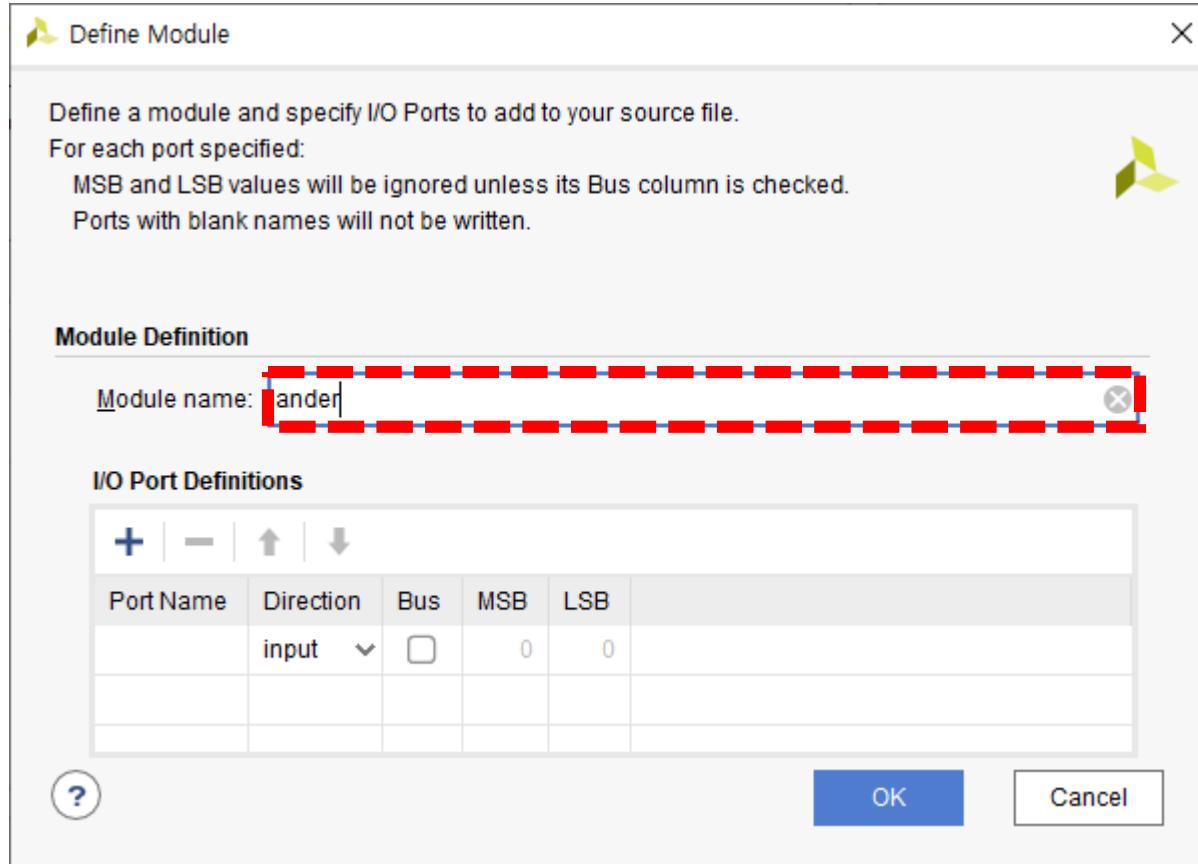


Step 2 Add Source in Vivado Project 4



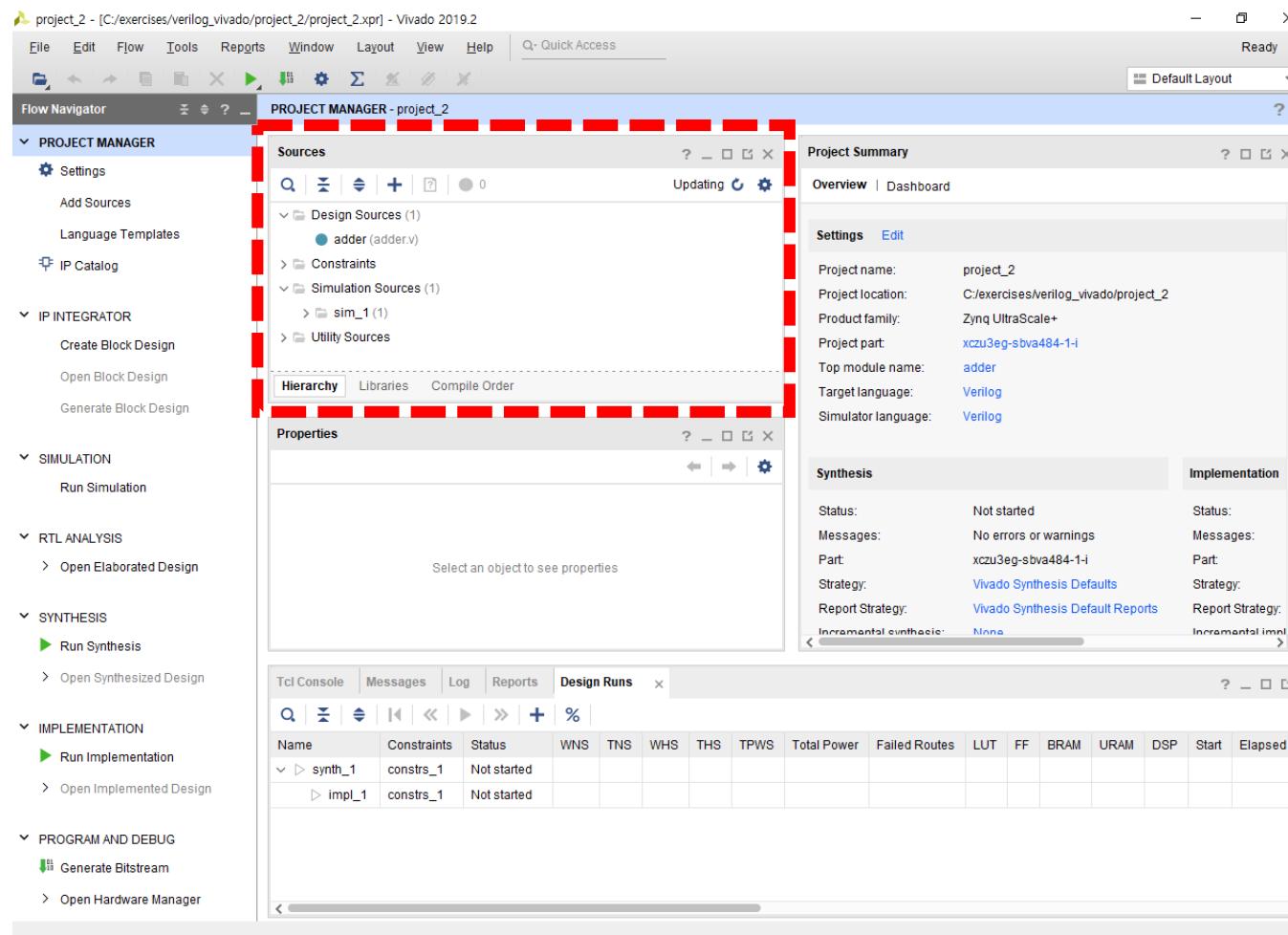
- ◆ 전에 생성한 ander.v 파일이 추가되어 있는 것을 확인할 수 있다.
- ◆ Finish 버튼을 클릭하여 소스 추가를 완료한다.

Step 2 Add Source in Vivado Project 5



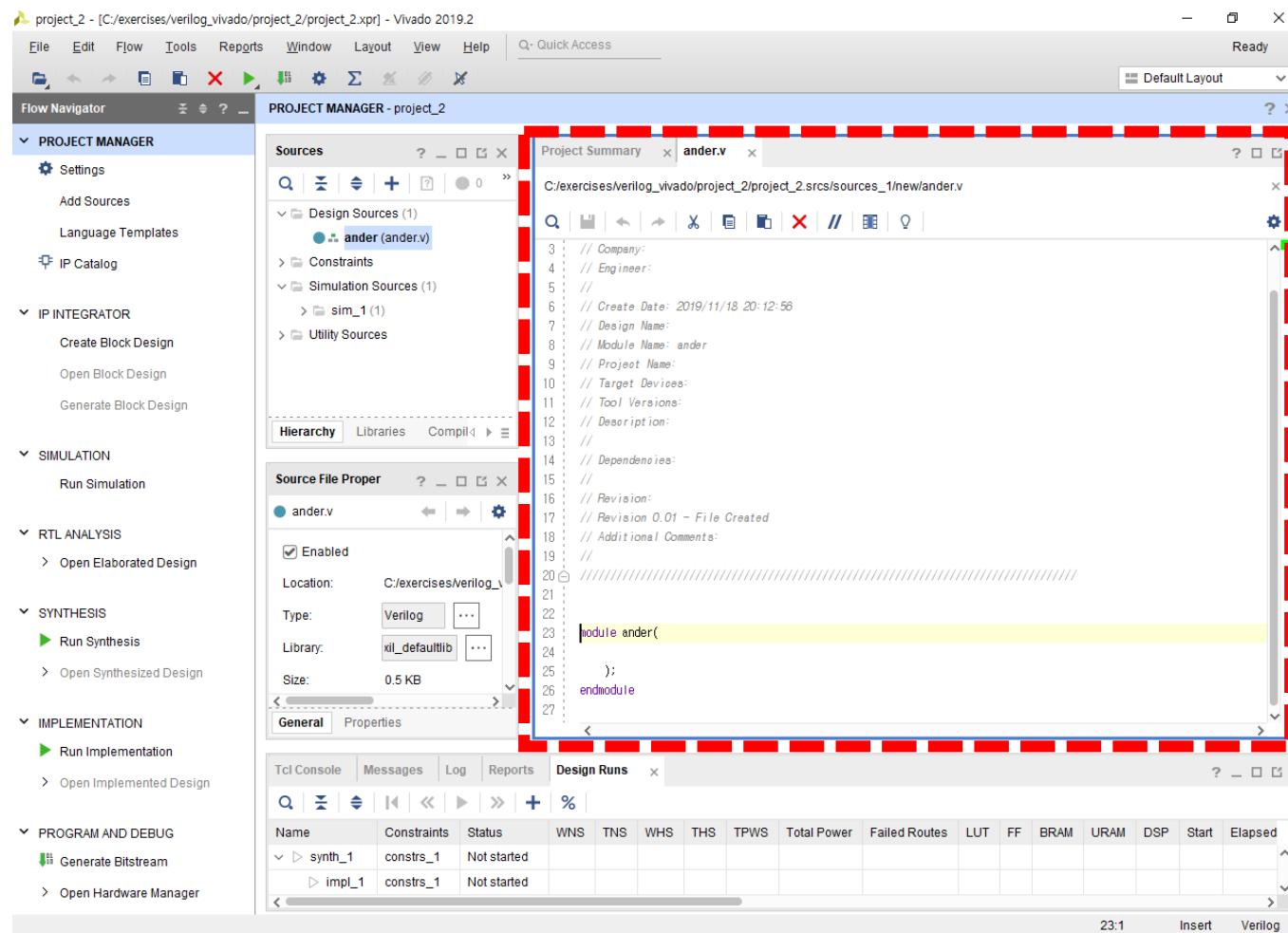
❖ 생성된 ander 모듈의 입출력 포트를 설정하는 화면이다. 여기서 포트를 설정하면 ander.v 파일의 포트 선언부를 자동으로 작성해준다. 여기서는 직접 코딩을 할 예정이니 아무 설정 없이 OK 버튼을 클릭한다.

Step 2 Add Source in Vivado Project 6



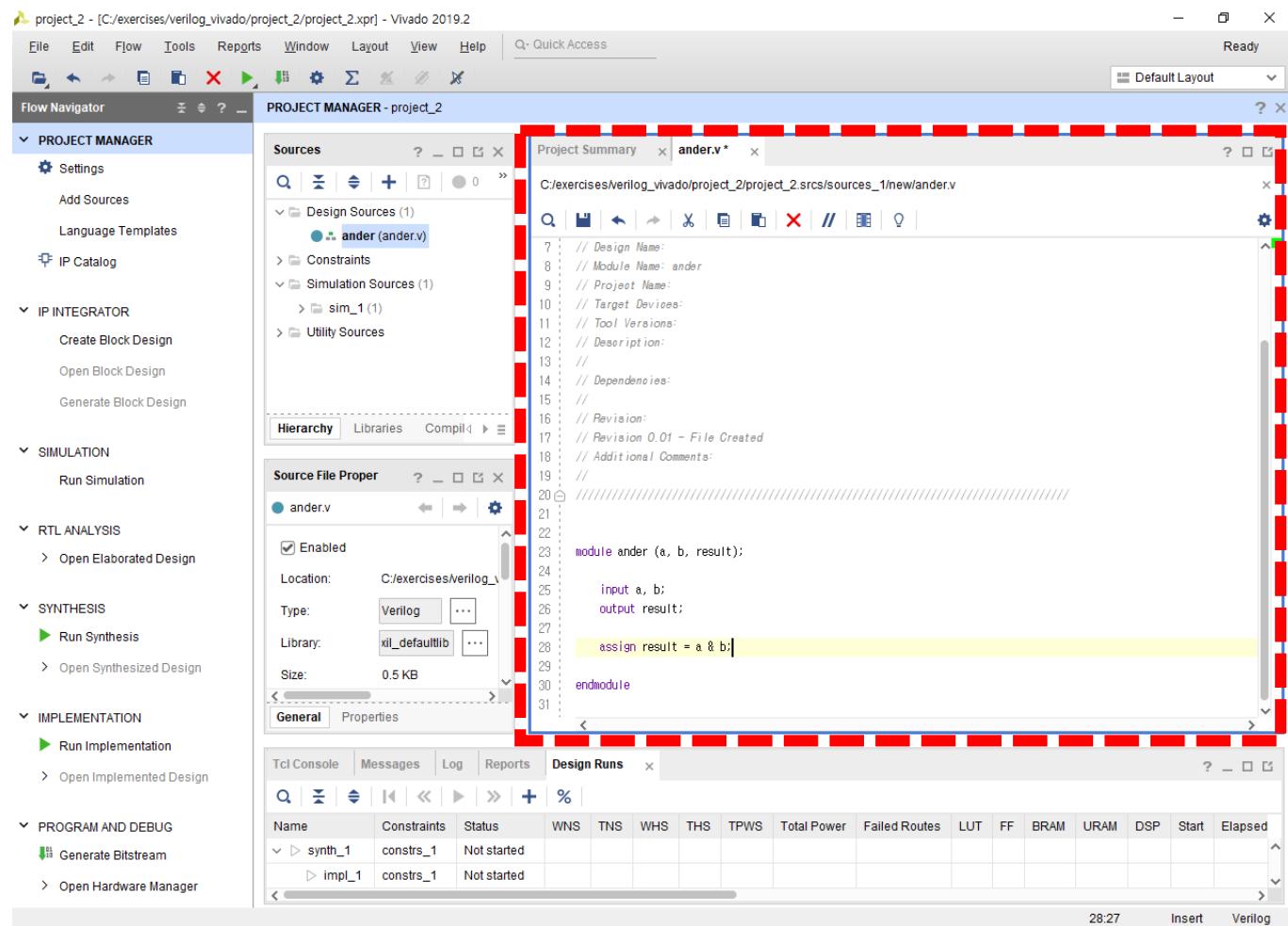
❖ 프로젝트 그림 2-13과 같이
Sources window안에 adder.v
소스파일이 추가된 것을 확인
할 수 있다.

Step 3 Modifying Source Code 1



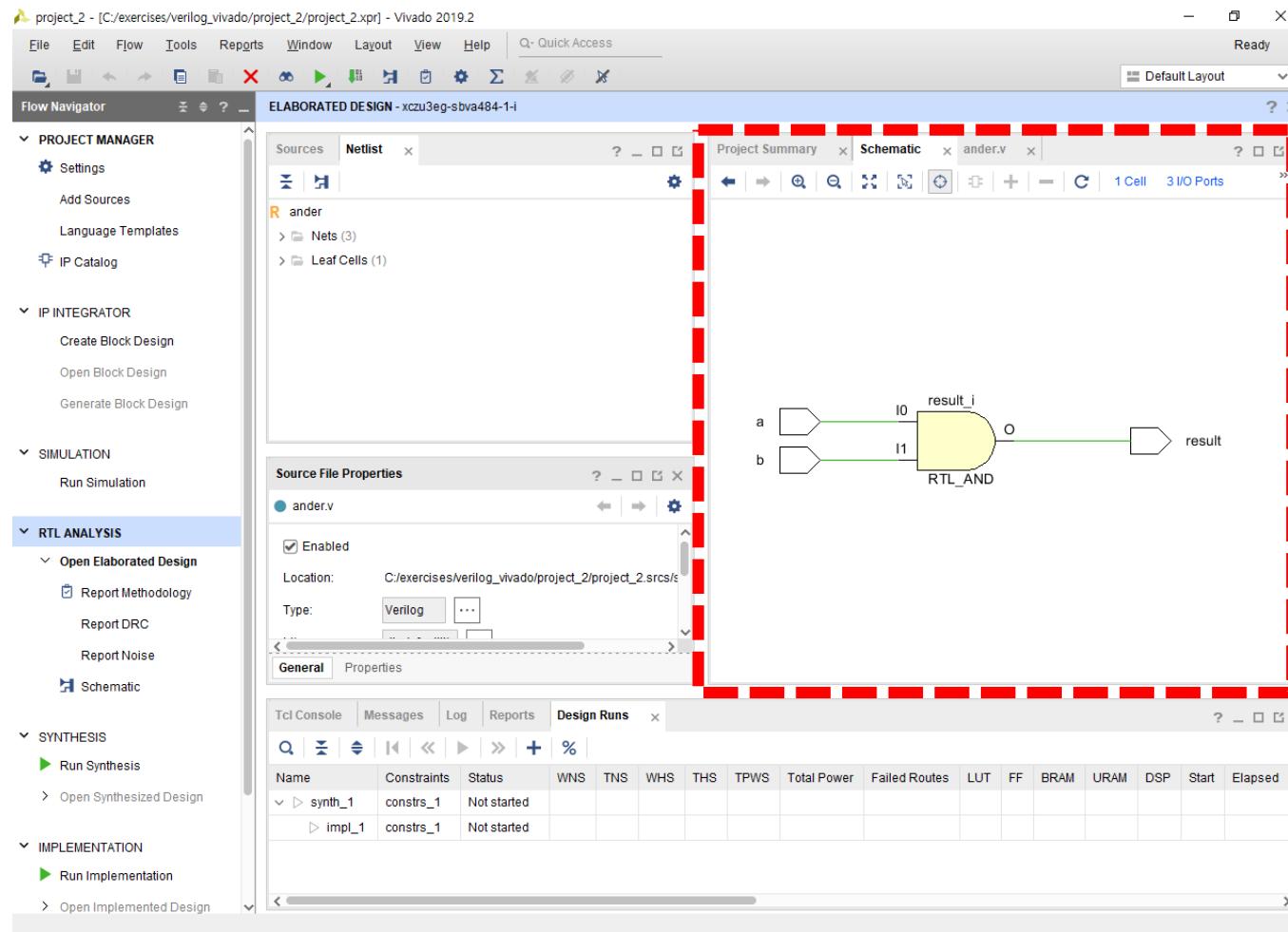
❖ Sources windows 안에
ander.v 파일을 더블 클릭하면
Editor window에 ander.v 파일
의 소스코드를 확인할 수 있다.
(※ Vivado는 Verilog의 기본적
인 형태를 만들어 준다.)

Step 3 Modifying Source Code 2



- ❖ Editor Window 안에 and 모듈을 코딩한다. (※ //는 주석 처리된 부분이므로 무시해도 된다.)
- ❖ 코딩이 완료되면 File ⇒ Text Editor ⇒ Save File 메뉴 또는 Ctrl+S키를 사용하여 파일을 저장한다.

Step 4 Schematic in Elaboration

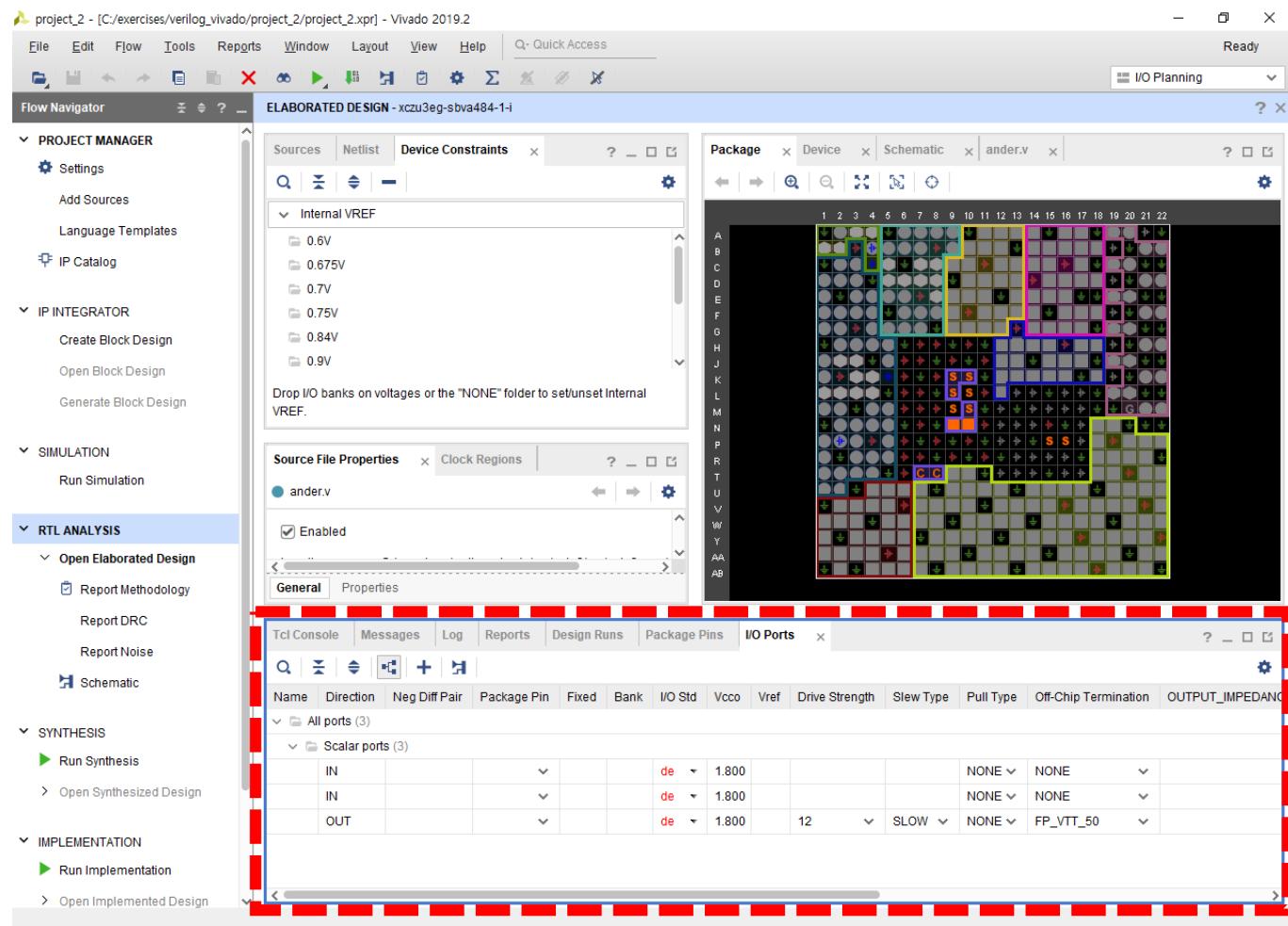


❖ Flow Navigator ⇒ Project Manager ⇒ RTL Analysis ⇒ Open Elaborated Design ⇒ Schematic을 클릭하면 Elaboration이 진행된 후 Schematic을 보여준다.

Step 5 Pin Constraints 1

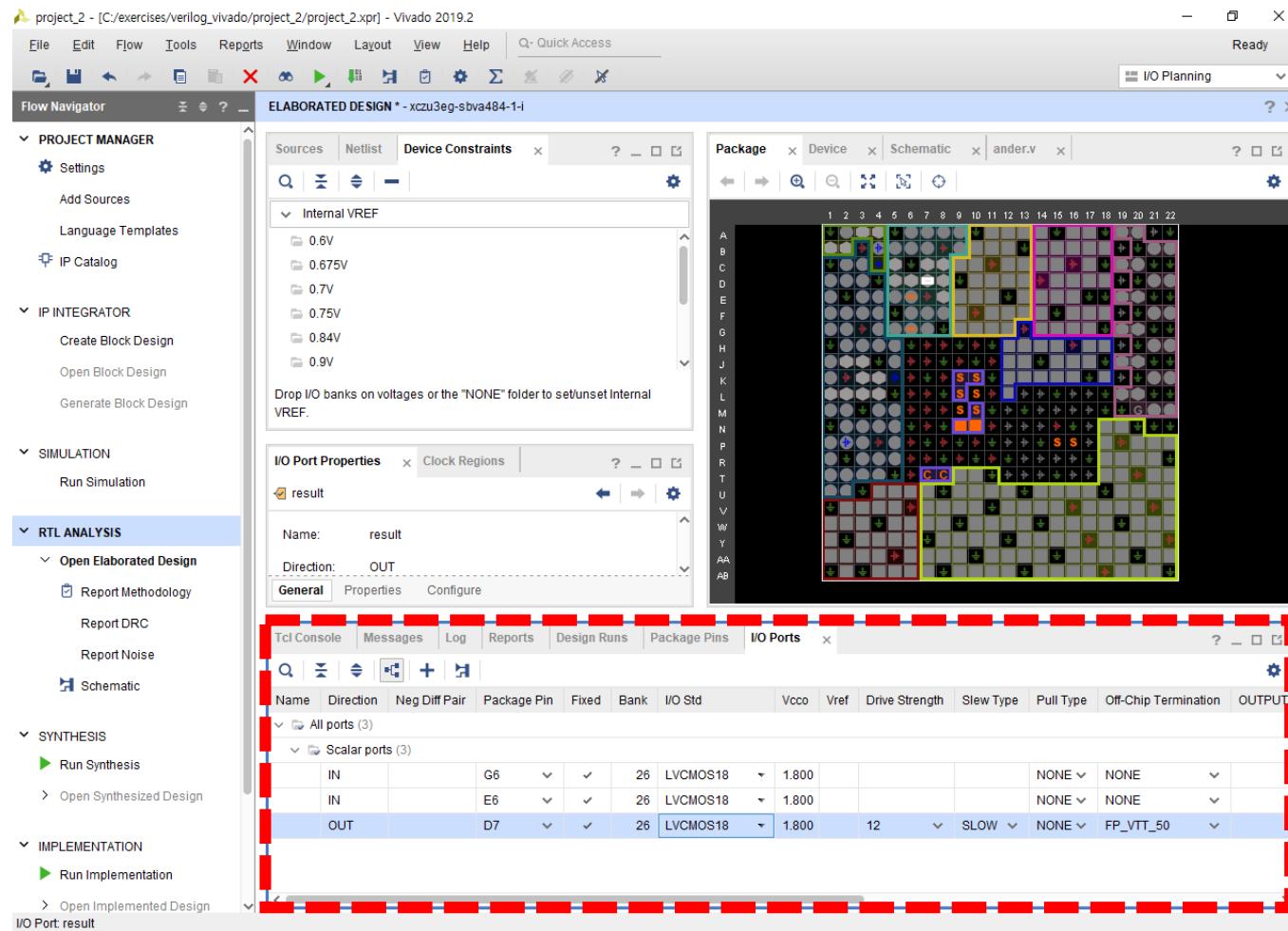
- ❖ Constraints는 사전적 의미로는 제한하다 라는 말로써 FPGA 설계에 필요한 제한조건을 만들기 위한 것이다.
- ❖ 예를 들어 가장 기본적인 Constraints에는 Pin Constraints가 있는데 Pin Constraints란 FPGA Device에 설계한 하드웨어의 입출력 포트가 FPGA Device의 어떤 Pin을 사용할지 지정하는 것이다.
- ❖ 보드에 다운로드 하지 않고 합성 또는 시뮬레이션만 한다면 Pin Constraints를 할 필요가 없지만 여기서는 Ultra96 보드에 다운로딩하여 구현할 예정이니 Ultra96 보드에 맞는 Pin Constraints를 추가해 주어야 한다.
- ❖ 본 교육에서 사용하는 Ultra96 Training Kit는 Ultra96보드에 Pmod96보드를 장착한 후, Pmod96보드에 있는 PMOD_A, PMOD_B, PMOD_C커넥터에 적절한 Pmod모듈을 장착하여 사용하도록 되어 있다.
- ❖ Pmod96보드에 장착된 Pmod 커넥터는 상하 각 6핀씩 총 12핀을 가지고 있는 커넥터로 보드를 위해서 바라봤을 때, 우측부터 좌측방향으로 번호가 매겨지고, 각 줄의 가장 좌측인 6,12번 핀과 5,11번 핀은 각각 VCC와 GND핀으로 데이터 핀은 1~4번, 7~10번 핀이다.
- ❖ Pmod모듈의 핀들도 같은 방식으로 번호가 매겨져 있으므로, Pmod모듈을 Pmod96보드에 장착할 때는 각 핀이 어긋나지 않게 장착해야 한다.
- ❖ 또한 PMOD_A, PMOD_B커넥터에 연결된 FPGA I/O핀은 1.8V (LVCMOS18)로, PMOD_C와 연결된 FPGA I/O핀은 1.2V (LVCMOS12)로 설정해줘야 한다.
- ❖ FPGA L2핀을 통해 40MHz 외부 클럭이 공급되며 L2핀은 1.2V (LVCMOS12)로 설정해줘야 한다

Step 5 Pin Constraints 2



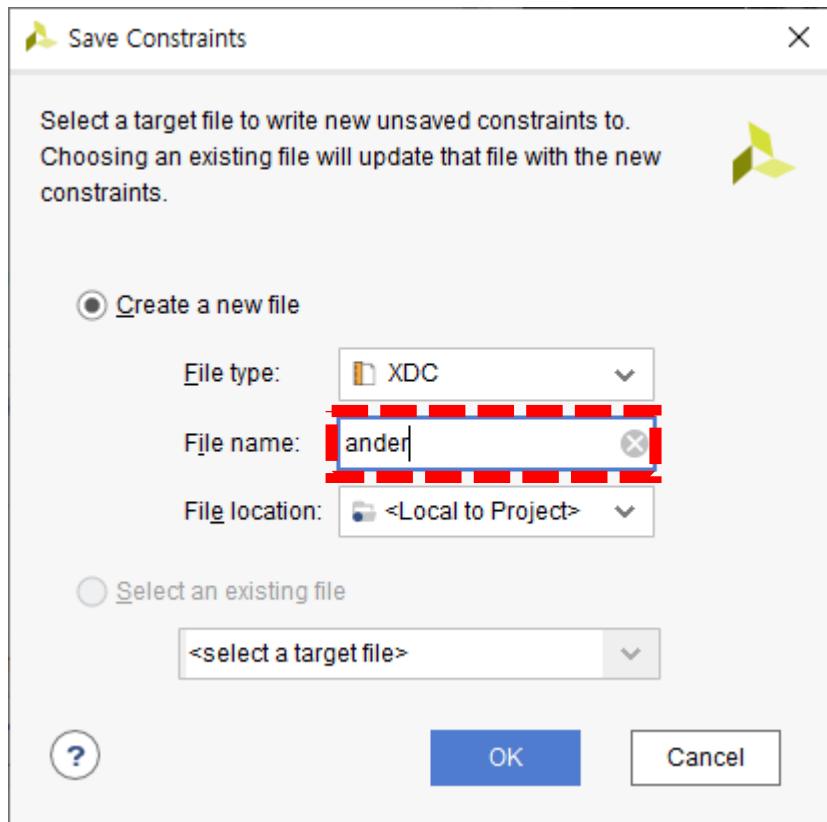
- ❖ Layout ⇒ I/O Planning을 클릭하면 하단에 I/O Ports 창이 나온다.
- ❖ I/O Ports 창 안에 Scalar ports를 열어보면 Ander 모듈의 포트들이 있는 것을 확인할 수 있다.

Step 5 Pin Constraints 3



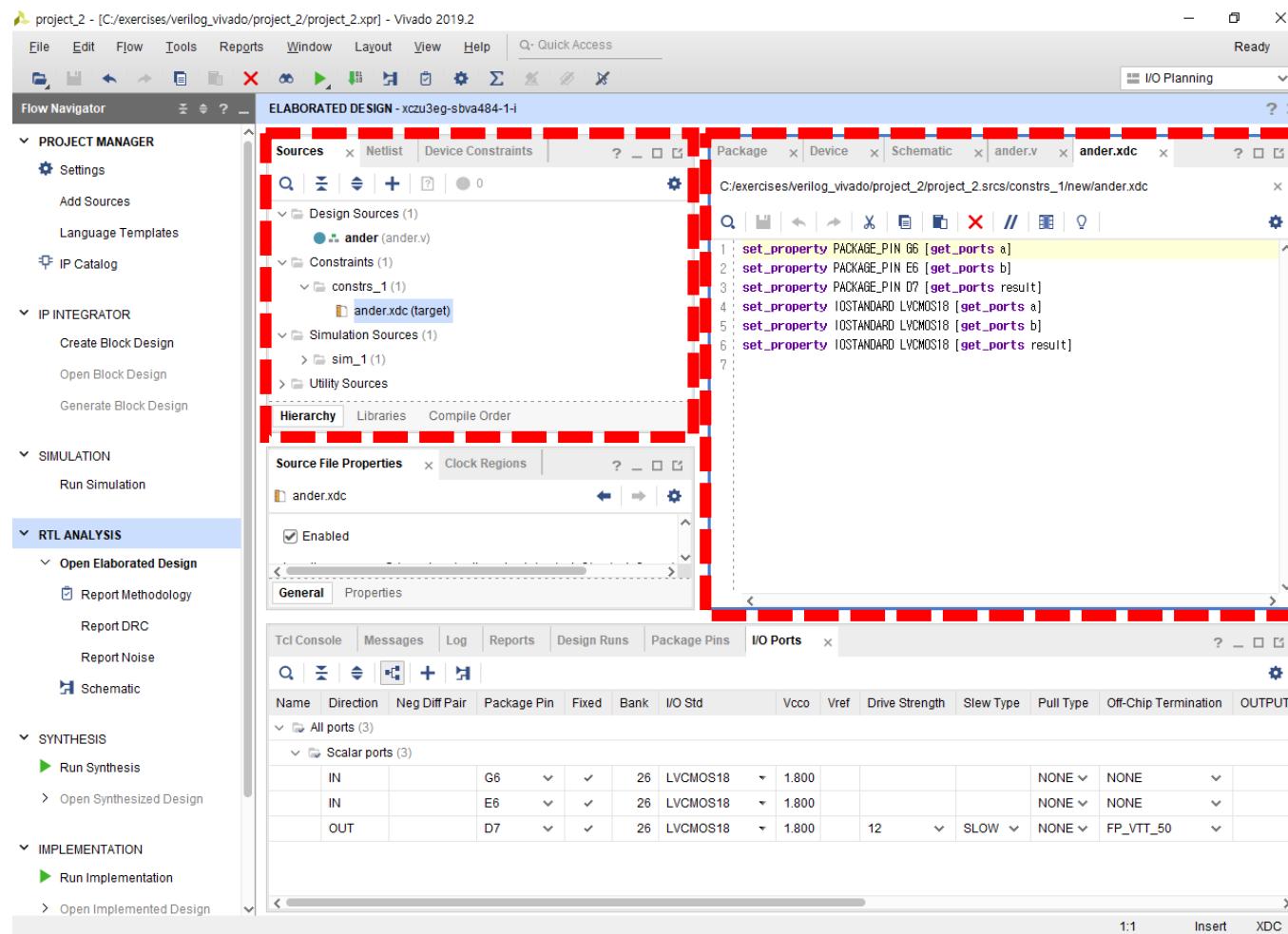
- ❖ Package Pin 열에 핀 번호를 입력하면 a, b, result 포트에 Pin Constraints를 할 수 있다.
- ❖ a → G6, b → E6, result → D7로 Pin Constraints를 하고 I/O Std열은 모두 LVCMS18을 선택한다.

Step 5 Pin Constraints 4



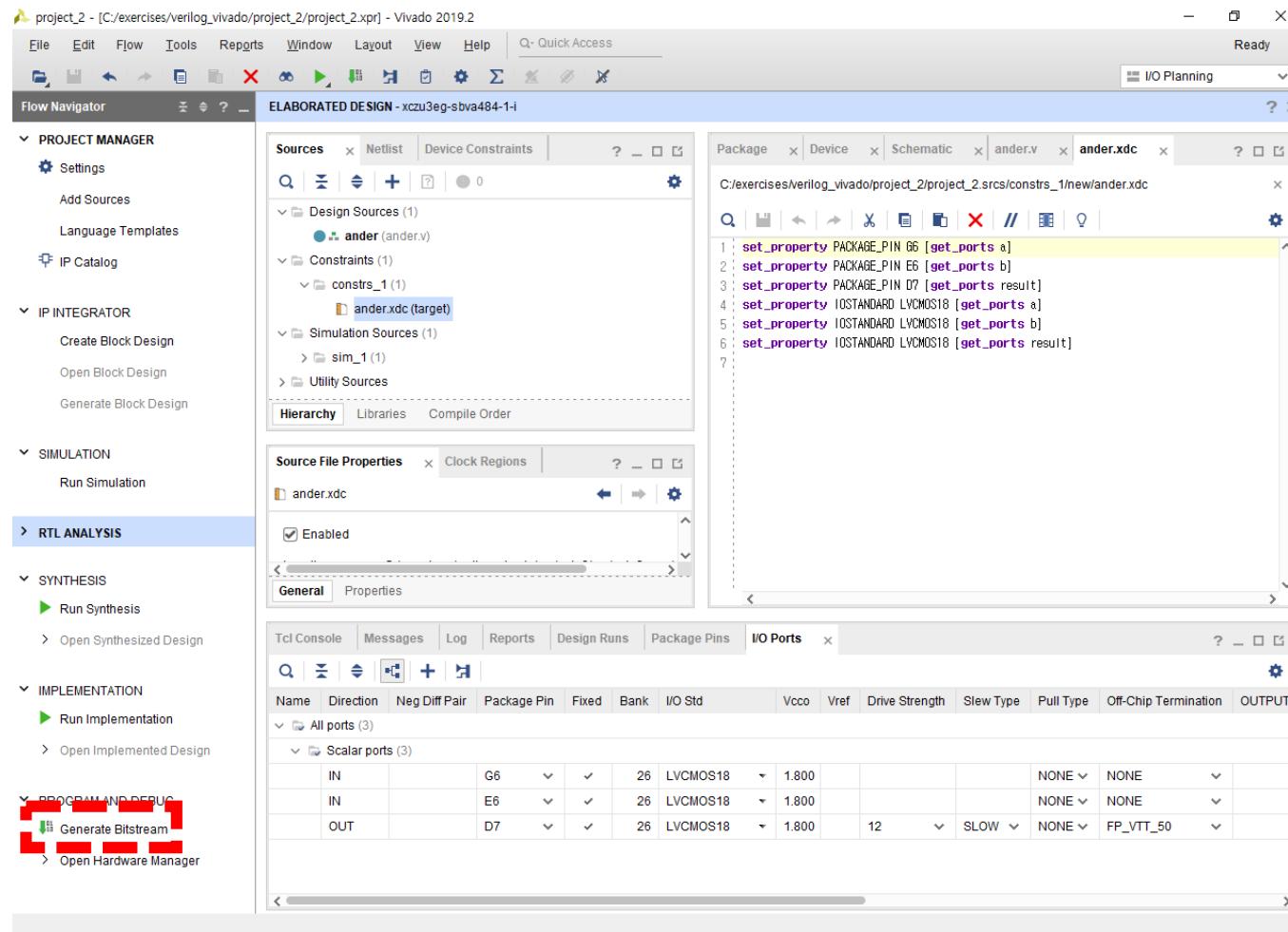
- ❖ File ⇒ Constraints ⇒ Save 메뉴 또는 Ctrl+S키를 사용하면 Constraints를 저장할 수 있다.
- ❖ Constraints를 저장할 파일 이름을 입력하는 창이 나오면 ander를 타이핑한 후 OK버튼을 클릭한다.

Step 5 Pin Constraints 5



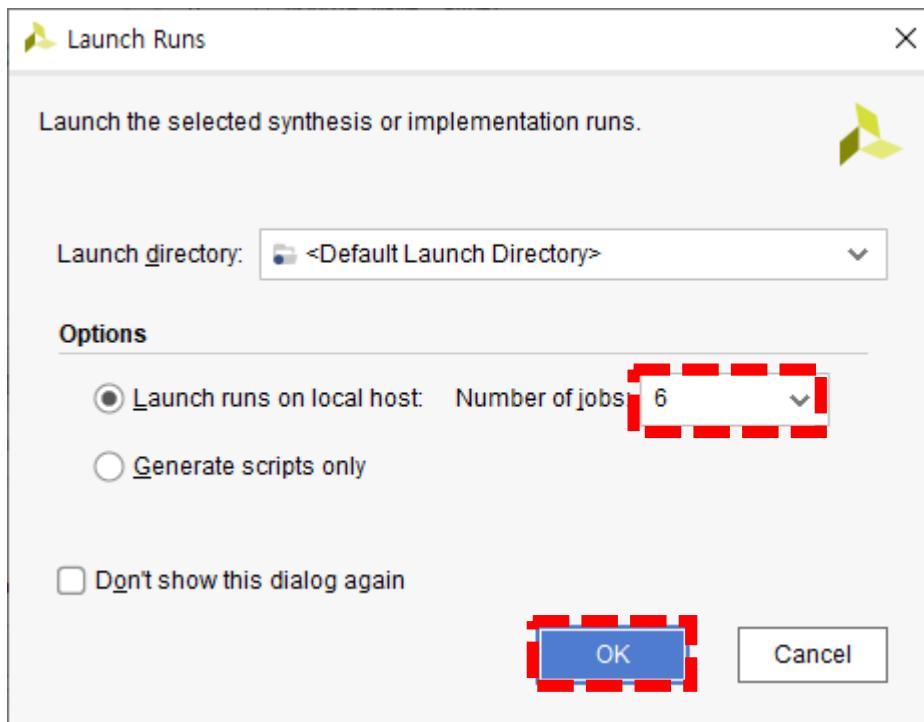
- ❖ Sources 탭을 클릭하면 Constraints 목록에 ex02.xdc 파일이 추가되어 있는 것을 확인할 수 있다.
- ❖ ex02.xdc 파일을 더블 클릭하면 Editor Window에서 Constraints 내용을 확인할 수 있다.

Step 6 Generate Bitstream 1



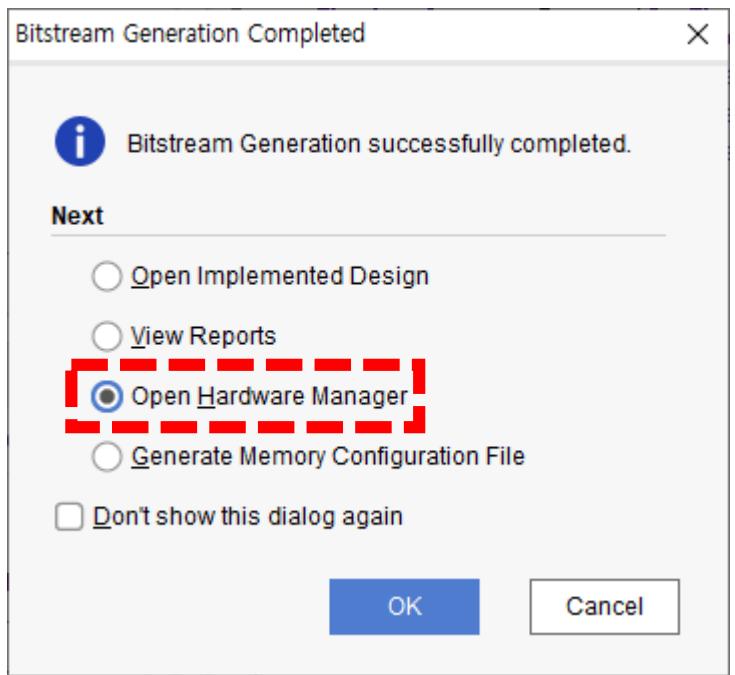
❖ Bitstream 생성을 위해 Flow Navigator ⇒ PROGRAM AND DEBUG ⇒ Generate Bitstream 을 클릭한다.

Step 6 Generate Bitstream 2



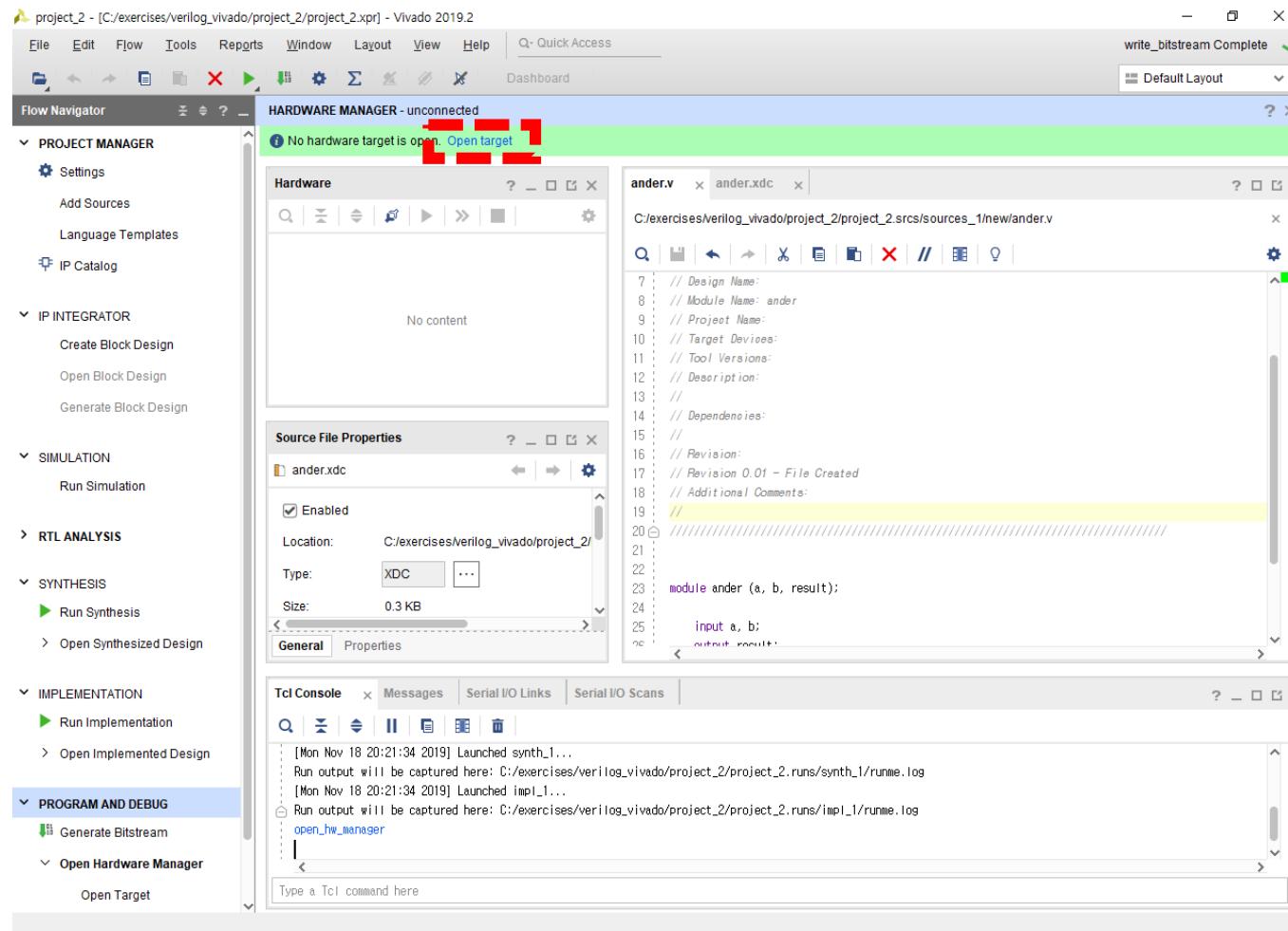
- ❖ Launch Runs 창은 Bitstream 을 생성하기 위해 CPU코어를 몇 개 사용할 지 선택하는 창이다.
- ❖ 적당한 개수를 선택한 후 OK 버튼을 클릭한다.

Step 7 Programming Device 1



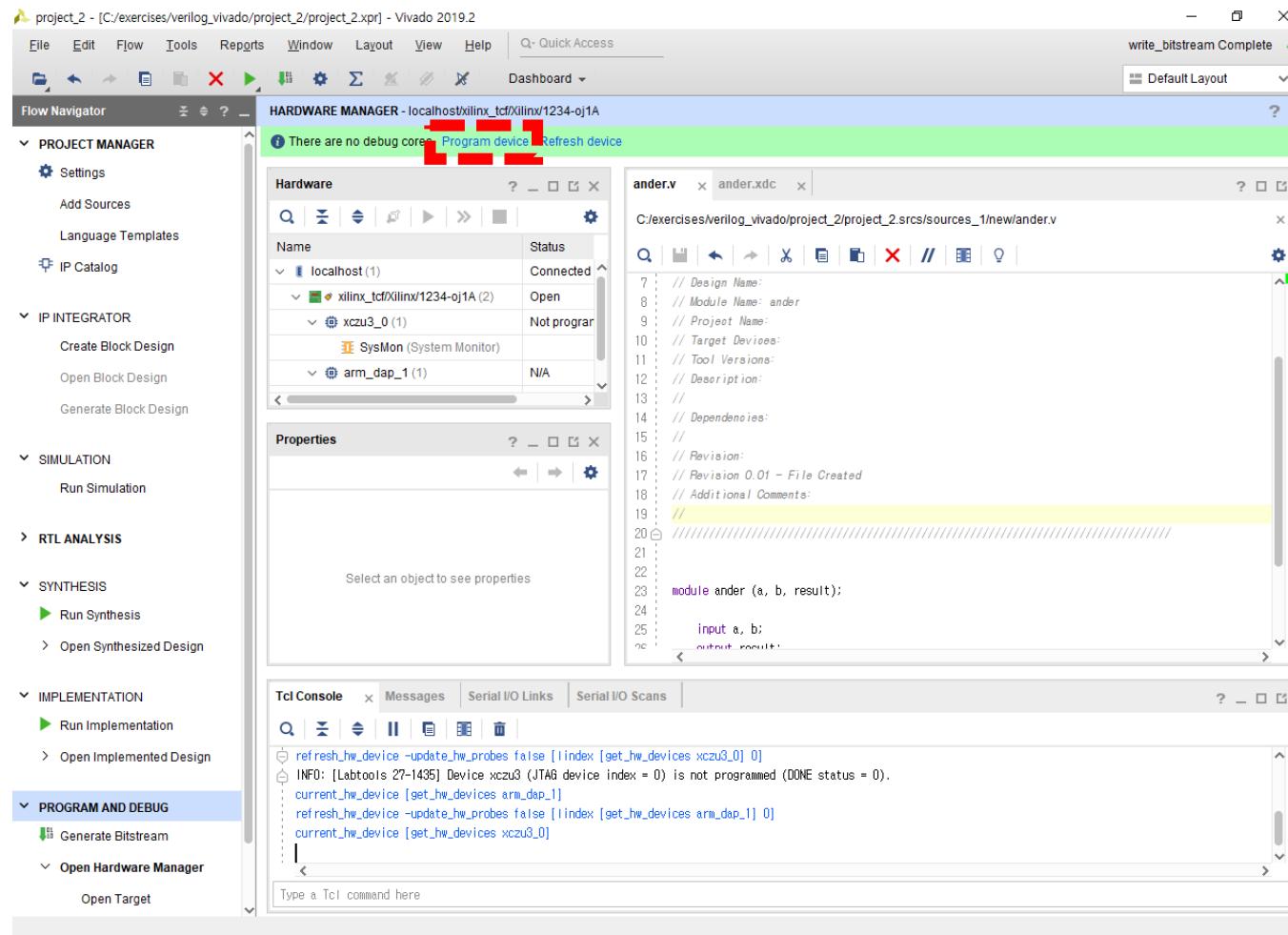
- ❖ Generate Bitstream이 정상적으로 완료되면 생성된 Bitstream을 디바이스에 프로그래밍하기 위해 하드웨어 매니저를 열 것인지 묻는 창이 나온다.
- ❖ Open Hardware Manager를 선택한 후 OK 버튼을 클릭한다.
(※ Flow Navigator ⇒ PROGRAM AND DEBUG ⇒ Open Hardware Manager를 클릭해도 된다.)

Step 7 Programming Device 2



- ❖ Ultra96 보드 위에 Pmod96 보드를 연결하고 Power Supply와 Ultra96 USB-to-JTAG/UART Pod를 Ultra96 보드에 연결하고 PMOD_A 커넥터에 Pmod8LD, PMOD_B 커넥터의 윗줄에 PmodBTN을 연결한 후 POWER 버튼을 누른다.
- ❖ Hardware Window 위에 Open Target이라고 되어 있는 부분을 클릭한다.
- ❖ 풀 다운 메뉴가 나오면 Auto Connect를 선택한다.

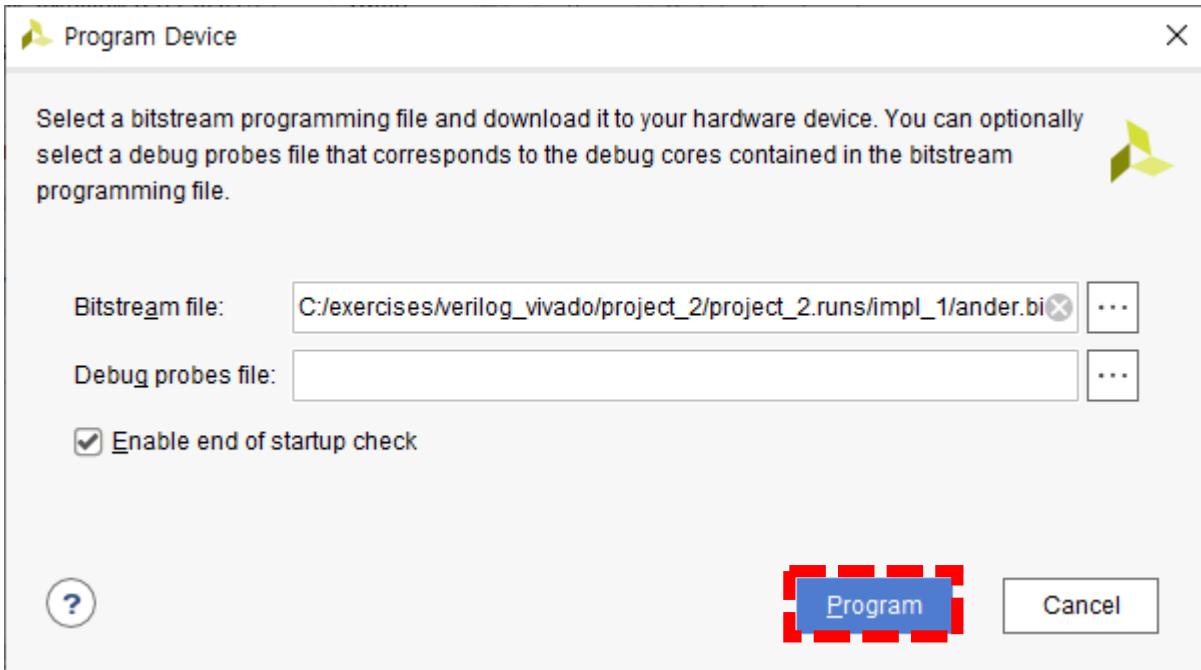
Step 7 Programming Device 3



❖ Ultra96 USB-to-JTAG/UART
Pod와 USB Cable가 연결된 상태에서 Auto Connect를 클릭하면 JTAG 인터페이스를 통해 JTAG Chain에 연결되어 있는 디바이스를 찾아서 보여준다.

❖ Hardware Window 위에 Program device 라고 되어 있는 부분을 클릭한다.

Step 7 Programming Device 4



- ❖ Program Device 창이 나오면 디폴트로 앞 과정에서 생성한 Bitstream file이 선택되어 있다.
- ❖ Program 버튼을 클릭하면 FPGA Device에 프로그래밍이 진행된다.
- ❖ Ultra96 보드에 프로그래밍이 완료된 후 PmodBTN의 BTN0와 BTN1을 동시에 누르면 Pmod8LD의 LD0가 켜지는 것을 확인할 수 있다.
- ❖ 동작을 확인한 후 POWER 버튼을 10초 정도 눌러서 시스템을 종료한다.

Chapter 3 Verilog HDL Basic Syntax

- Data Types & Parameter
- Number Representations
- Operators
- Ultra96 Training Kit Exercises 1

Net Data Types 1

- ❖ Net Data Type은 하드웨어들의 연결에 사용되는 와이어 또는 넷을 표현한 것으로 wire Data Type 을 가장 많이 사용한다.
- ❖ wire와 tri는 기본적으로 동일한 데이터 타입이지만 readability를 위해서 둘로 나누어서 사용한다.
- ❖ wire는 하나의 신호가 인가될 때 사용하고 tri는 한 개 이상의 신호가 인가되어 질 때 tri-state buffer와 같이 enable신호가 필요한 회로에서 사용한다.
- ❖ input 또는 output으로 선언된 포트는 wire가 생략된 것으로 wire data type으로 선언된 것과 같다.

```
`timescale 1 ns / 1 ps

module ander2 (a, b, result);

    input a, b;
    output result;
    wire c;

    assign c = a & b;
    assign result = c;

endmodule
```

Net Data Types 2

- ❖ Verilog에서 Array는 [width-1:0]와 같은 형태로 작성한다.
- ❖ Data Type을 wire로 Array를 [width-1:0]으로 선언하면 width-bit size의 wire Data Type을 Array로 선언한 것이다.

```
`timescale 1 ns / 1 ps

module ander3 (a, b, result);

    input [15:0] a,b;
    output [15:0] result;
    wire [15:0] c;

    assign c[15:8] = a[15:8] & b[15:8];
    assign c[7:0] = a[7:0] & b[7:0];
    assign result = c;

endmodule
```

Register Data Types

- ❖ Register Data Type은 데이터를 저장할 소자를 의미하는 것으로 사용되며 reg Data Type을 가장 많이 사용한다.
- ❖ Register Data Type에는 reg, integer, time, real과 같은 Data Type이 있다.
- ❖ reg는 1-bit data type이고 integer는 32-bit data type이다.
- ❖ reg는 wire와 같이 size를 설정할 수 있지만 integer는 32-bit size로 고정되어 있다.
- ❖ Register Data Type 선언 예
 - ❖ **reg** [7:0] a; // 8-bit size reg Data Type
 - ❖ **integer** i; // 32-bit size integer Data Type
- ❖ 그 밖의 Register Data Type은 time, real이 있다.
 - ❖ time은 64-bit size를 가지며 시뮬레이션을 할 때 시간을 저장하고 비교하고 확인하는데 사용하는 Data Type이다.
 - ❖ real은 64-bit size를 가지며 floating-point 값을 저장하는데 사용하는 Data Type이다.
 - ❖ 이 두 개의 Data Type은 합성이 되지 않는 Data Type이다.
 - ❖ time data type은 시간을 저장하고 비교 확인하는데 필요한 시뮬레이션을 할 때만 사용되고
 - ❖ floating-point 값을 저장하고 계산하는 real data type은 Verilog 합성 툴이 floating-point 연산을 지원하지 않아서 합성이 되지 않고 시뮬레이션만 가능하다.

Parameter

- ❖ Parameter는 주로 데이터의 width와 같은 값을 특정하기 위한 상수 또는 Net Data Type이나 Register Data Type에 속하지 않는 고유한 상수로 사용된다.
- ❖ parameter 선언 예
 - ❖ **parameter** width = 8;
 - ❖ **input** [width-1:0] d;
- ❖ Parameter는 모듈이나 IP의 재사용을 위해 사용된다.

Chapter 3 Verilog HDL Basic Syntax

- Data Types & Parameter
- Number Representations
- Operators
- Ultra96 Training Kit Exercises 1

Basic Number Representations

- ❖ Verilog의 숫자 표현법에는 sized와 unsized 두 가지가 있다.
 - ❖ sized는 숫자의 bit size를 정하고 표현하는 것이고 unsized는 숫자의 bit size를 정하지 않고 표현하는 것이다.
 - ❖ unsized로 표현하여 bit size를 정하지 않으면 디폴트로 32-bit size가 된다.
 - ❖ **sized** : 8'b11101010 (8-bit size)
 - ❖ **unsized** : 425 (32-bit size)
 - ❖ sized는 bit size를 정한 표현법으로 제일 앞에 숫자가 size를 의미하며 뒤에 따라오는 'b는 뒤에 따로 숫자의 진법을 의미한다.
 - ❖ 'b는 binary(2진법)를 표현한 것이고 그 밖에 'o는 octal(8진법)을, 'd는 decimal(10진법)을, 'h는 hexadecimal(16진법)을 각각 표현한 것이다.
 - ❖ 'b0010 (32-bit size, binary)
 - ❖ 6'o47 (6-bit size, octal)
 - ❖ 8'd38 (8-bit size, decimal)
 - ❖ 16'ha09f (16-bit size, hexadecimal)

Other Number Representations

- ❖ 음수 표현
 - ❖ -8'd38 (8-bit size)
- ❖ Verilog에서는 0과 1값 이외에 unknown을 의미하는 x값과 high impedance를 의미하는 z값이 존재한다.
 - ❖ 'b0x10 (32-bit size, binary)
 - ❖ 16'hax9z (16-bit size, hexadecimal)
- ❖ readability를 위해 _를 사용할 수 있다.
 - ❖ 32'b1010_0001_0010_1100_1010_1011_0101_0010
- ❖ bit extension
 - ❖ 8'b0010 = 8'b00000010
 - ❖ 7'bx1010 = 7'bxxx1010
 - ❖ 3'bz = 3'bzzz
 - ❖ 6'b1100 = 6'b001100
 - ❖ 4'd23 = 4'b0111 = 4'd7

Chapter 3 Verilog HDL Basic Syntax

- Data Types & Parameter
- Number Representations
- Operators
- Ultra96 Training Kit Exercises 1

Arithmetic Operators

연산자	설명	예 (a=4'b1010, b=4'b0101)
+	더하기	$a + b = 12$
-	빼기	$a - b = 6$
*	곱하기	$a * b = 27$
/	나누기	$a / b = 3$
%	나머지	$a \% b = 0$

❖ 산술 연산하는 값에 X값이나 Z값이 포함되어 있으면 결과 값은 unknown 값이 된다.

Relational Operators

연산자	설명	예 (a=4'b1010, b=4'b0101)
<	작다	a < b // false (0)
>	크다	a > b // true (1)
<=	작거나 같다	a <= b // false (0)
>=	크거나 같다	a >= b // true (1)

❖ Verilog 관계 연산자는 C에서 사용하는 연산자와 동일하다.

Logical Operators

연산자	설명	예 (a=1, b=0)
!	Not	<code>!a // false (0)</code>
<code>&&</code>	And	<code>a && b // false (0)</code>
<code> </code>	Or	<code>a b // true (1)</code>

- ❖ 연산하는 값에 X값이나 Z값이 포함되어 있으면 결과 값은 unknown 값이 된다.

Bitwise Operators and Reduction Operators

연산자	설명	예 (a=4'b0011, b=4'b1010)
<code>~</code>	Not	$\sim a = 4'b1100$
<code>&</code>	And	$a \& b = 4'b0010$
<code> </code>	Or	$a b = 4'b1011$
<code>^</code>	Xor	$a ^ b = 4'b1001$
<code>~^ or ^~</code>	Xnor	$a \sim^ b = 4'b0110$

연산자	설명	예 (a=4'b0011)
<code>&</code>	And	$\&a = 1'b0$
<code>~&</code>	Nand	$\sim\&a = 1'b1$
<code> </code>	Or	$ a = 1'b1$
<code>~ </code>	Nor	$\sim a = 1'b0$
<code>^</code>	Xor	$^a = 1'b0$
<code>~^ or ^~</code>	Xnor	$\sim^a = 1'b1$

Shift Operators and Conditional Operators

연산자	설명	예 (a=4'b1011)
>>	shift right	a >> 1 = 4'b0101
<<	shift left	a << 3 = 4'b1000

- ❖ 조건(enable)이 참이면 식1(data)이 out에 인가되고 거짓이면 식2(32'hz)가 out에 인가된다.
- ❖ 조건 ? 식1 : 식2;
 - ❖ assign out = (enable) ? data : 32'hz;

Concatenation Operator

- ❖ 표현된 식을 연결해주는 연산자이다.
- ❖ {식1,식2}

❖ $\{4'b0010,3'b101\} = 7'b0010101$

```
`timescale 1 ns / 1 ps

module ander4 (a, b, result);

    input [15:0] a, b;
    output [15:0] result;
    wire [7:0] c,d;

    assign c = a[15:8] & b[15:8];
    assign d = a[7:0] & b[7:0];
    assign result = {c,d};

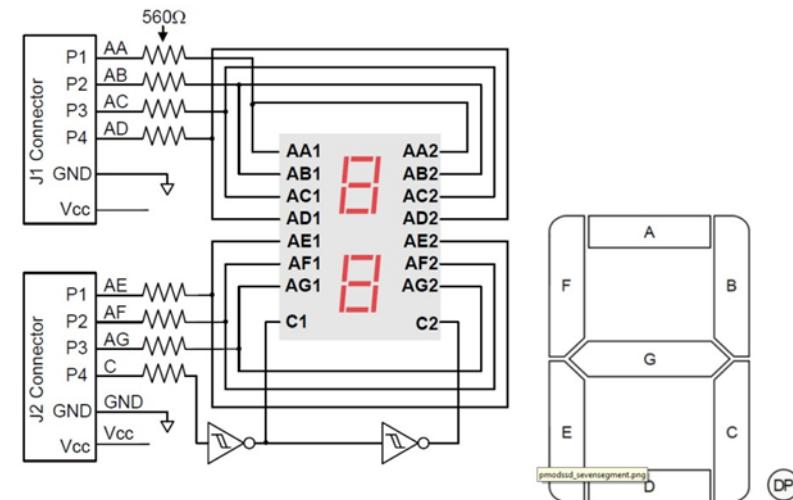
endmodule
```

Chapter 3 Verilog HDL Basic Syntax

- Data Types & Parameter
- Number Representations
- Operators
- Ultra96 Training Kit Exercises 1

Ultra96 Training Kit Exercises 1

- ❖ EX1-1 Pmod8LD와 PmodBTN을 활용하여 4개의 버튼이 각각 하나의 LED를 제어하도록 프로그래밍 하시오.
- ❖ EX1-2 PmodBTN의 버튼으로부터 2-bit size 두 개의 입력 신호를 산술연산자 또는 비트연산자로 연산한 결과를 LED에 출력하도록 프로그래밍 하시오.
- ❖ EX1-3 Ultra96 Training Kit에는 7-Segment 실습을 위해 PmodSSD가 2개 포함되어 있다. 그림 3-4의 7-Segment에서 A, B, C, D, E, F를 켜고 G를 끄면 숫자 0을 표시할 수 있고 A, B, C, D, G를 켜고 E, F를 끄면 숫자 3을 표시할 수 있다. pmod_a와 pmod_b 커넥터에 PmodSSD 한 개를 연결하여 7-Segment에 특정 숫자 또는 문자가 표시되도록 프로그래밍 하시오.

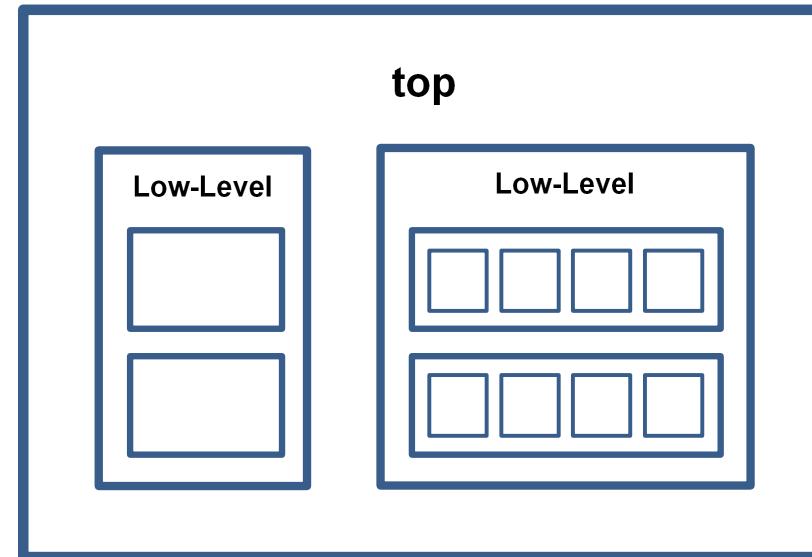


Chapter 4 Instantiation

- Hierarchical Design
- Instantiation
- IP Core Instantiation using Vivado
- Clocking Wizard
- Ultra96 Training Kit Exercises 2

Hierarchical Design

- ❖ 디지털 회로를 설계하는 기본적인 방식은 모듈, 컴포넌트, 소자라고 불리는 부품들을 연결하여 전체 하드웨어를 만드는 Structural Modeling 방식이다.
- ❖ 하나의 모듈은 하위 모듈들을 연결하여 만들고 하위 모듈들은 또 다른 하위 모듈들을 연결하여 만들어진다.
- ❖ 전체 모듈은 하위 모듈로 구성되고 하위 모듈은 그 모듈의 하위 모듈로 구성되게 되어 위에서부터 아래로 계층이 생긴다고 하여 계층적인 설계(Hierarchical Design)라고 한다.
- ❖ Hierarchical Design을 하면 부분적으로 나누어서 검증이 가능하여 디버깅이 용이하다는 장점이 있어서 Hierarchical Design을 추천한다.



Chapter 4 Instantiation

- Hierarchical Design
- Instantiation
- IP Core Instantiation using Vivado
- Clocking Wizard
- Ultra96 Training Kit Exercises 2

Instantiation Syntax

```
Module Name    Instance Name    ( Port Connection );
```

- ❖ Hierarchical Design을 하려면 상위레벨의 모듈에서 하위레벨의 모듈을 추가해야 한다.
- ❖ 상위레벨에서 하위레벨의 모듈을 추가하는 것을 인스턴스 생성(Instantiation)이라고 한다.
- ❖ 인스턴스 이름은 여러 개의 인스턴스들을 구분해 주기 위한 것이다.
- ❖ 포트를 연결하는 방식은 순서에 의한 방식(connected by order)과 이름에 의한 방식(connected by name) 두 가지가 있다.

Port Connection by Order

- ❖ 순서에 의한 포트 연결 방식은 하위레벨 모듈인 ander의 포트 리스트 순서대로 포트가 연결되도록 표현한다.

```
'timescale 1 ns / 1 ps

module top_order (top_a, top_b, top_result);

    input [3:0] top_a, top_b;
    output [3:0] top_result;

    ander u1 (top_a[3], top_b[3], top_result[3]);
    ander u2 (top_a[2], top_b[2], top_result[2]);
    ander u3 (top_a[1], top_b[1], top_result[1]);
    ander u4 (top_a[0], top_b[0], top_result[0]);

endmodule
```

Port Connection by Name

- ❖ 이름에 의한 포트 연결 방식은 하위레벨 모듈의 포트이름과 상위레벨 모듈의 넷을 명시하여 각각이 연결되도록 표현한다.

```
`timescale 1 ns / 1 ps

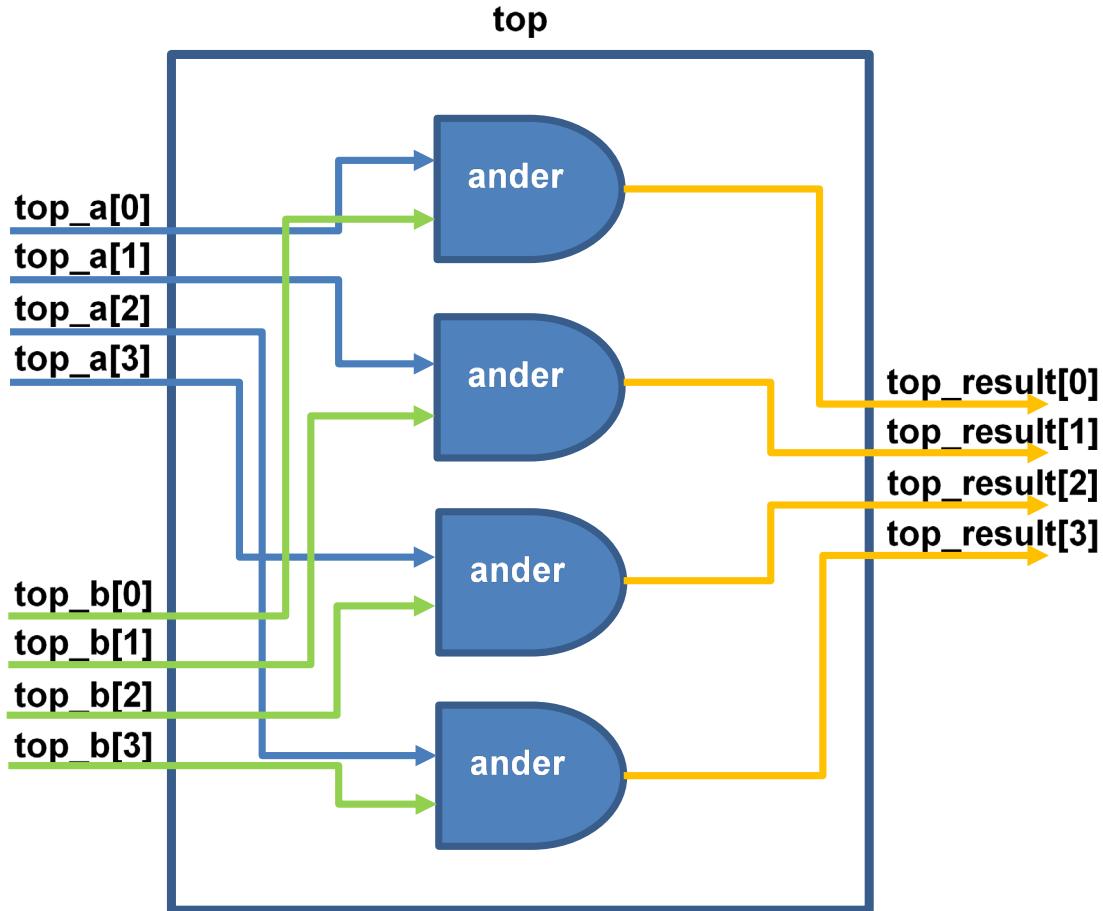
module top_name (top_a, top_b, top_result);

    input [3:0] top_a, top_b;
    output [3:0] top_result;

    ander u1( .a(top_a[3]), .b(top_b[3]), .result(top_result[3]) );
    ander u2( .a(top_a[2]), .b(top_b[2]), .result(top_result[2]) );
    ander u3( .a(top_a[1]), .b(top_b[1]), .result(top_result[1]) );
    ander u4( .a(top_a[0]), .b(top_b[0]), .result(top_result[0]) );

endmodule
```

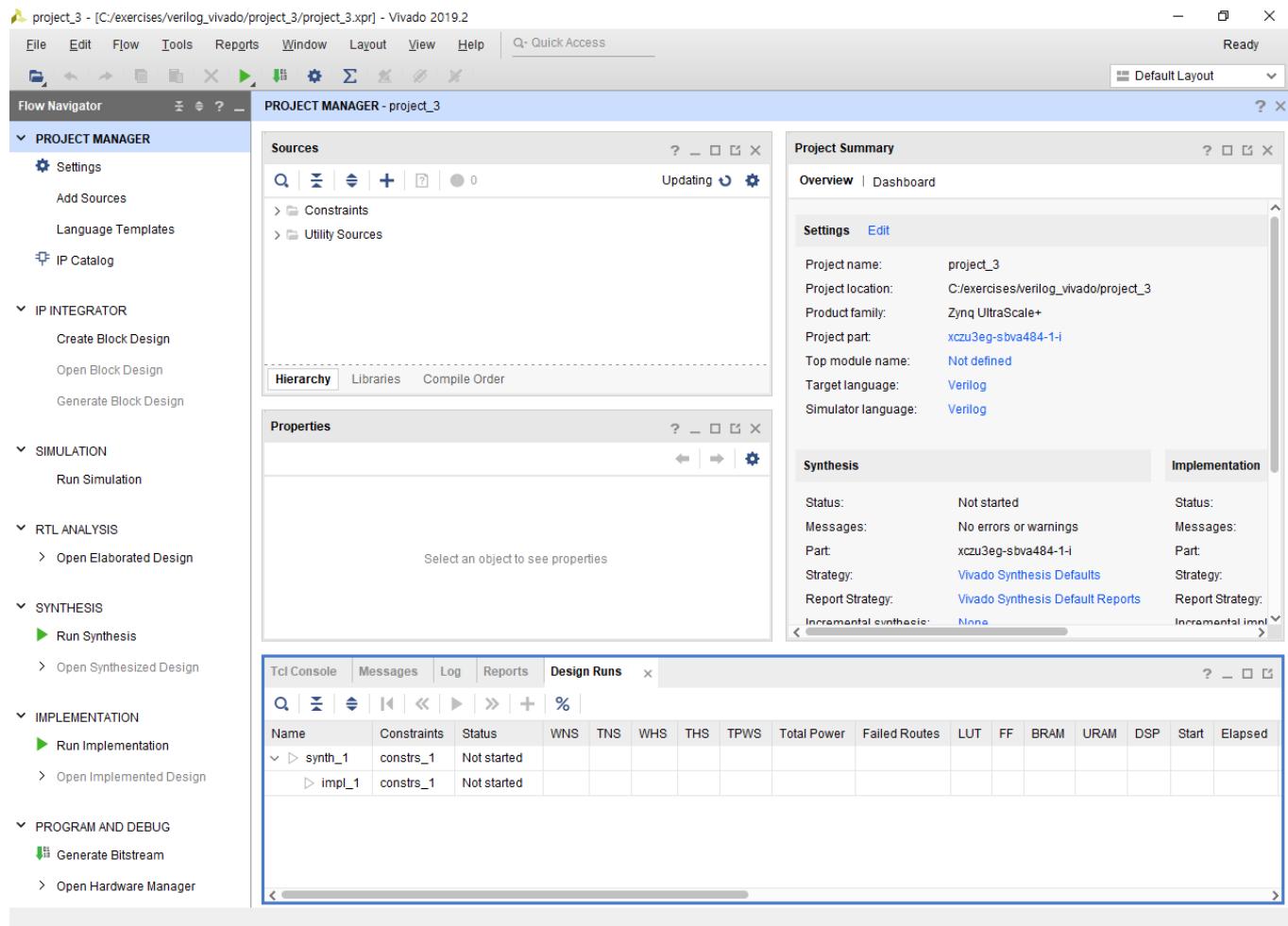
Schematic for Instantiation Example



Chapter 4 Instantiation

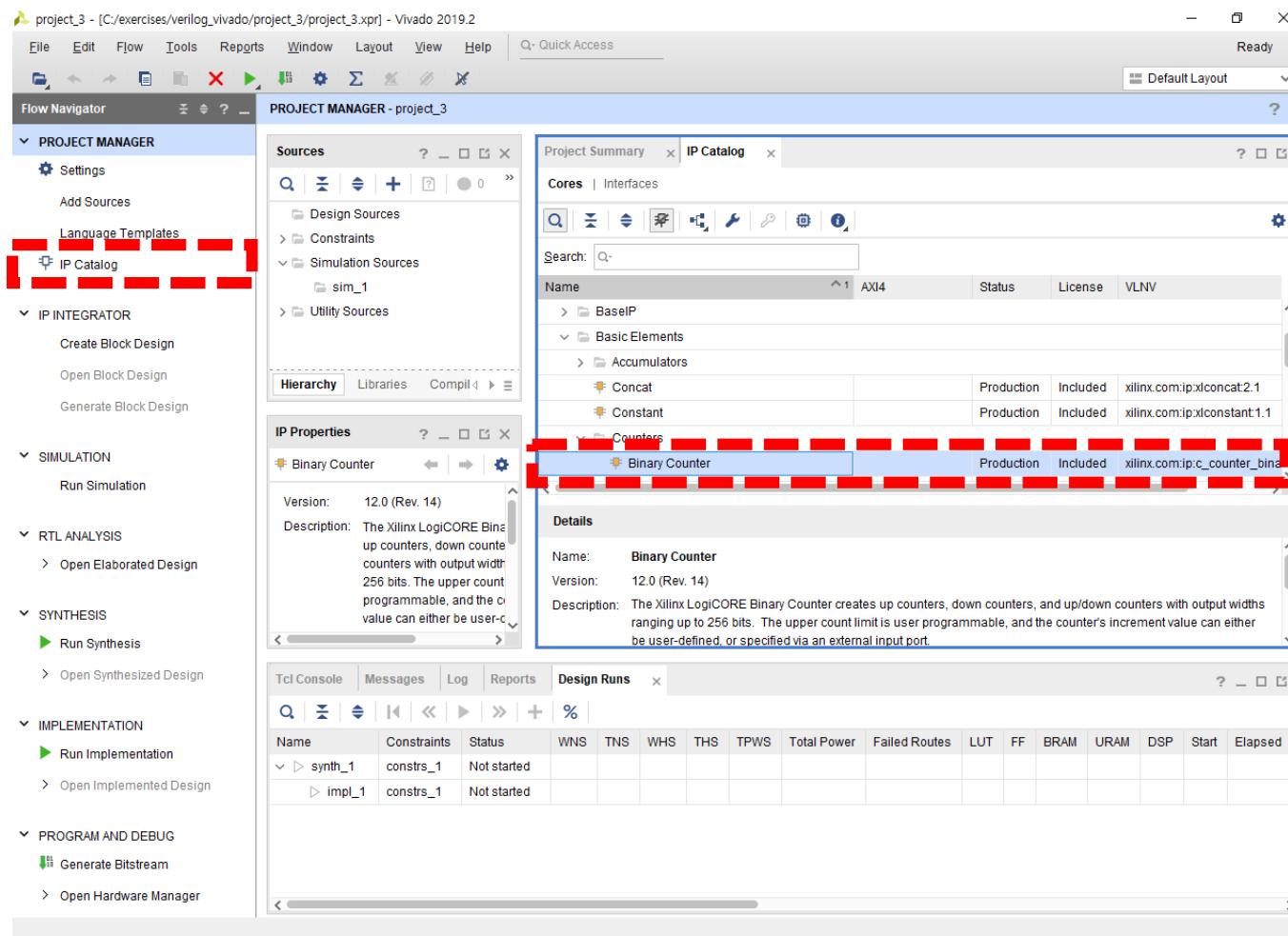
- Hierarchical Design
- Instantiation
- IP Core Instantiation using Vivado
- Clocking Wizard
- Ultra96 Training Kit Exercises 2

Step 1 Creating Vivado Project



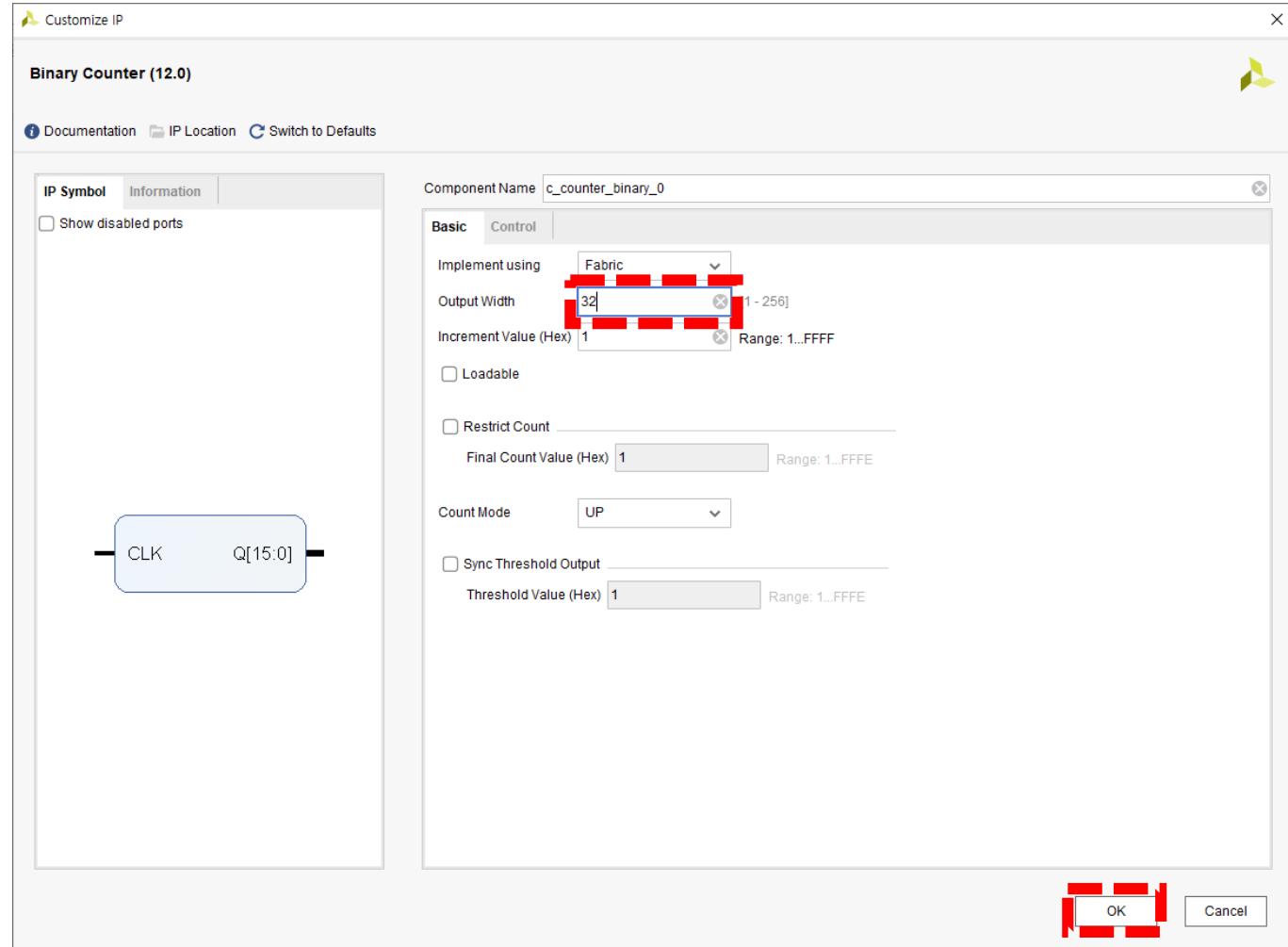
❖ Chapter 1의 Vivado 프로젝트 만들기를 참조하여 project_3 프로젝트를 만든다.

Step 2 Creating IP Core 1



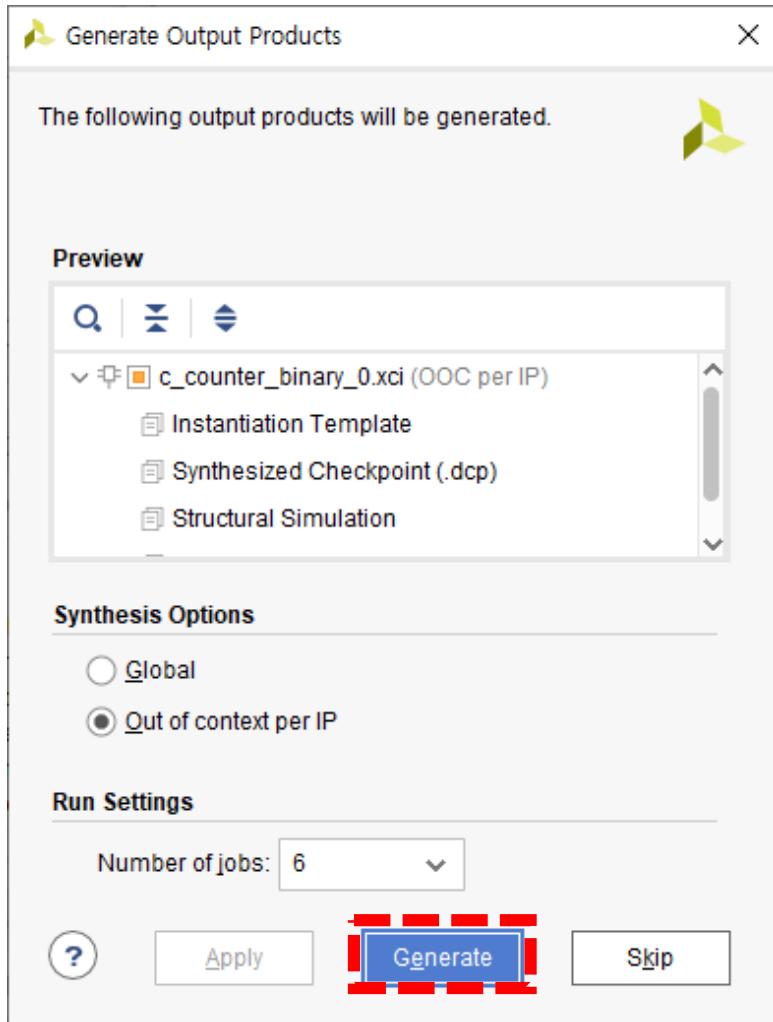
- ❖ Flow Navigator안에 Project Manager 아래 IP Catalog를 클릭한다.
- ❖ IP Catalog가 나오면 IP Catalog안에 Basic Elements ⇒ Counters ⇒ Binary Counter를 더블클릭 한다.

Step 2 Creating IP Core 2



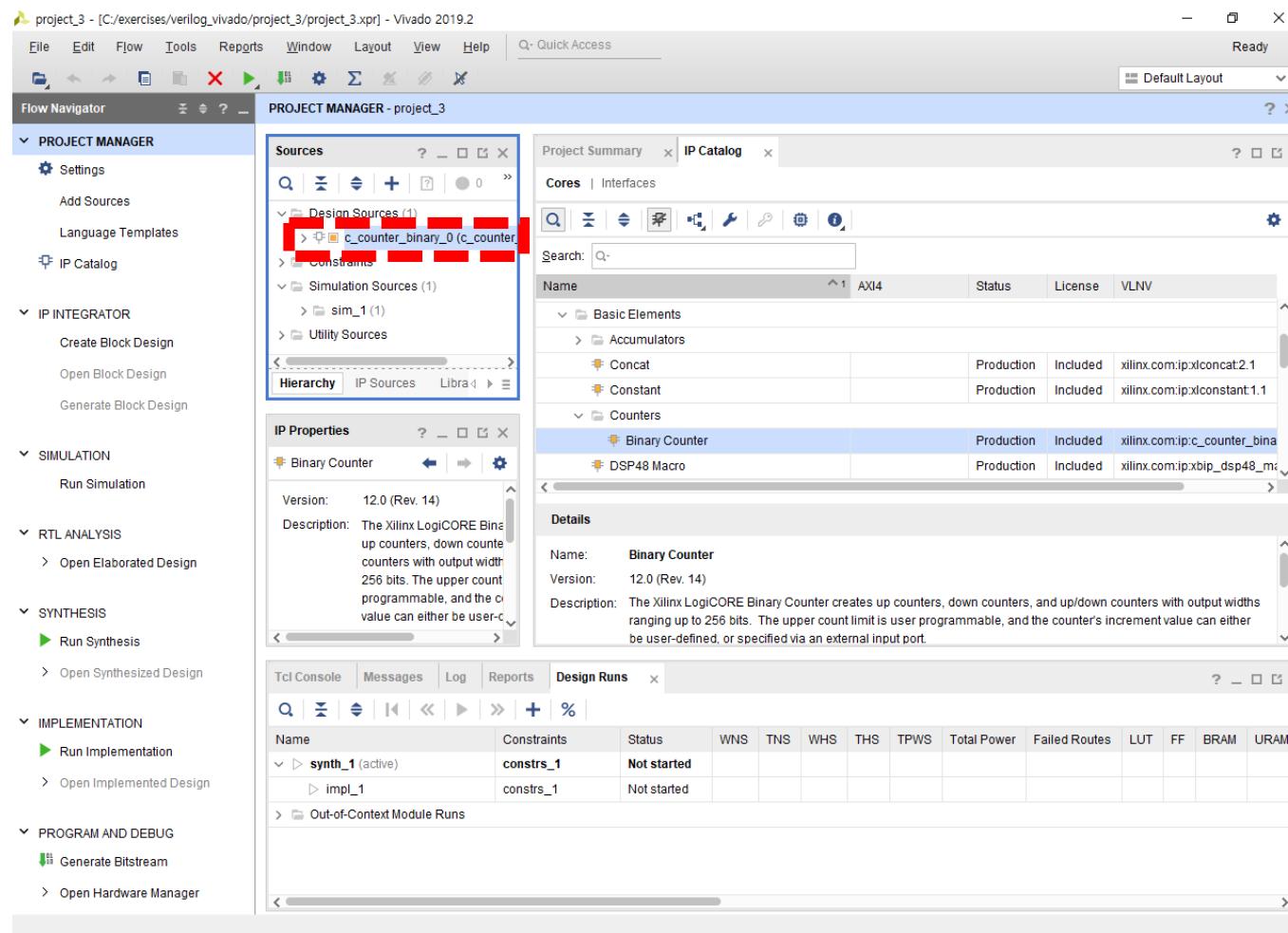
❖ Binary Counter IP를 Customize하는 Window가 나으면 Output Width를 32로 수정하고 OK버튼을 클릭한다.

Step 2 Creating IP Core 3



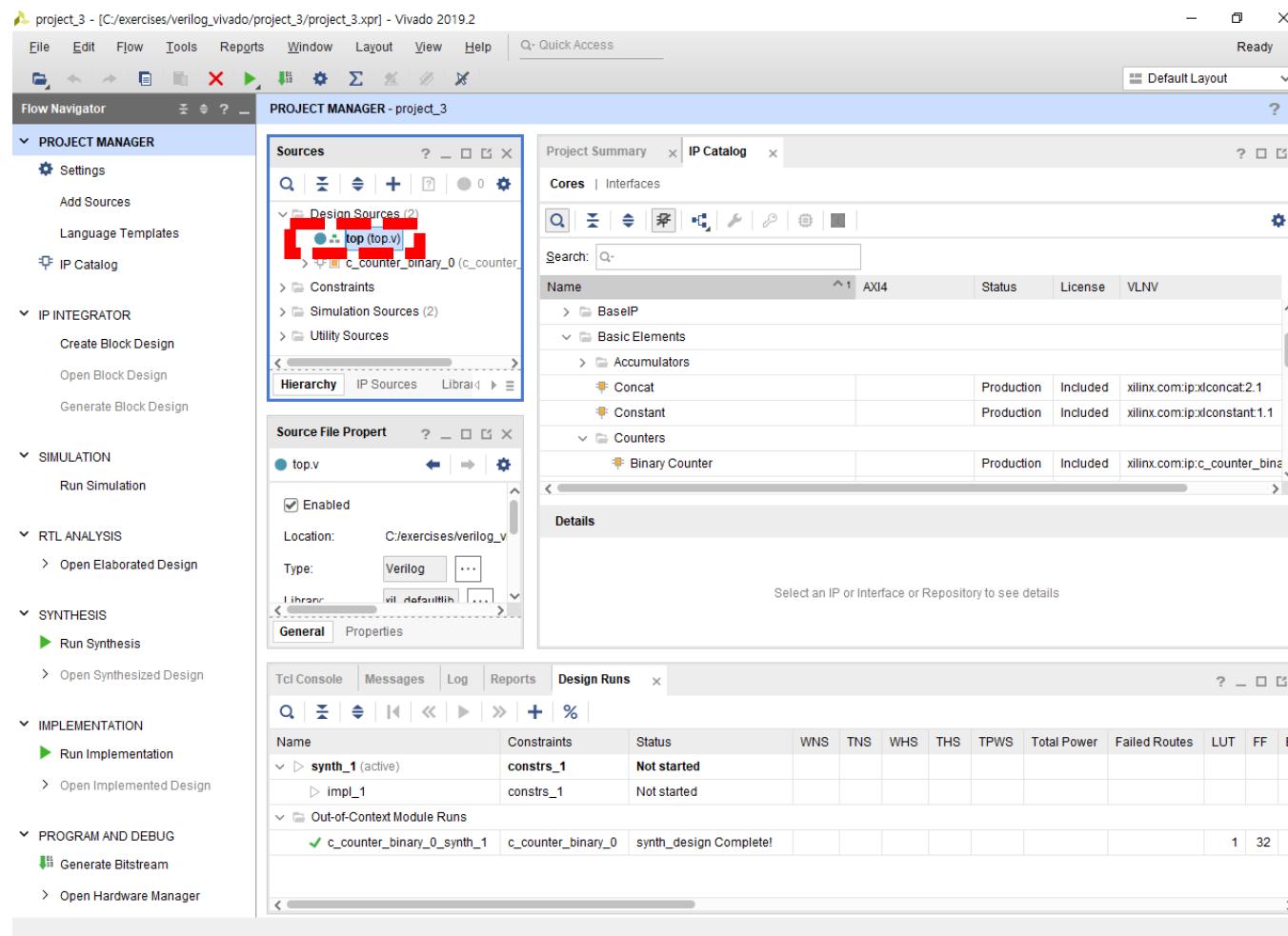
❖ Generate Output Products
Window가 나오면 디폴트 그대로 두고 Generate 버튼을 클릭한다.

Step 2 Creating IP Core 4



❖ Sources Window 안에 Design Sources를 보면
c_counter_binary_0로 Binary Counter IP가 생성된 것을 확인할 수 있다.

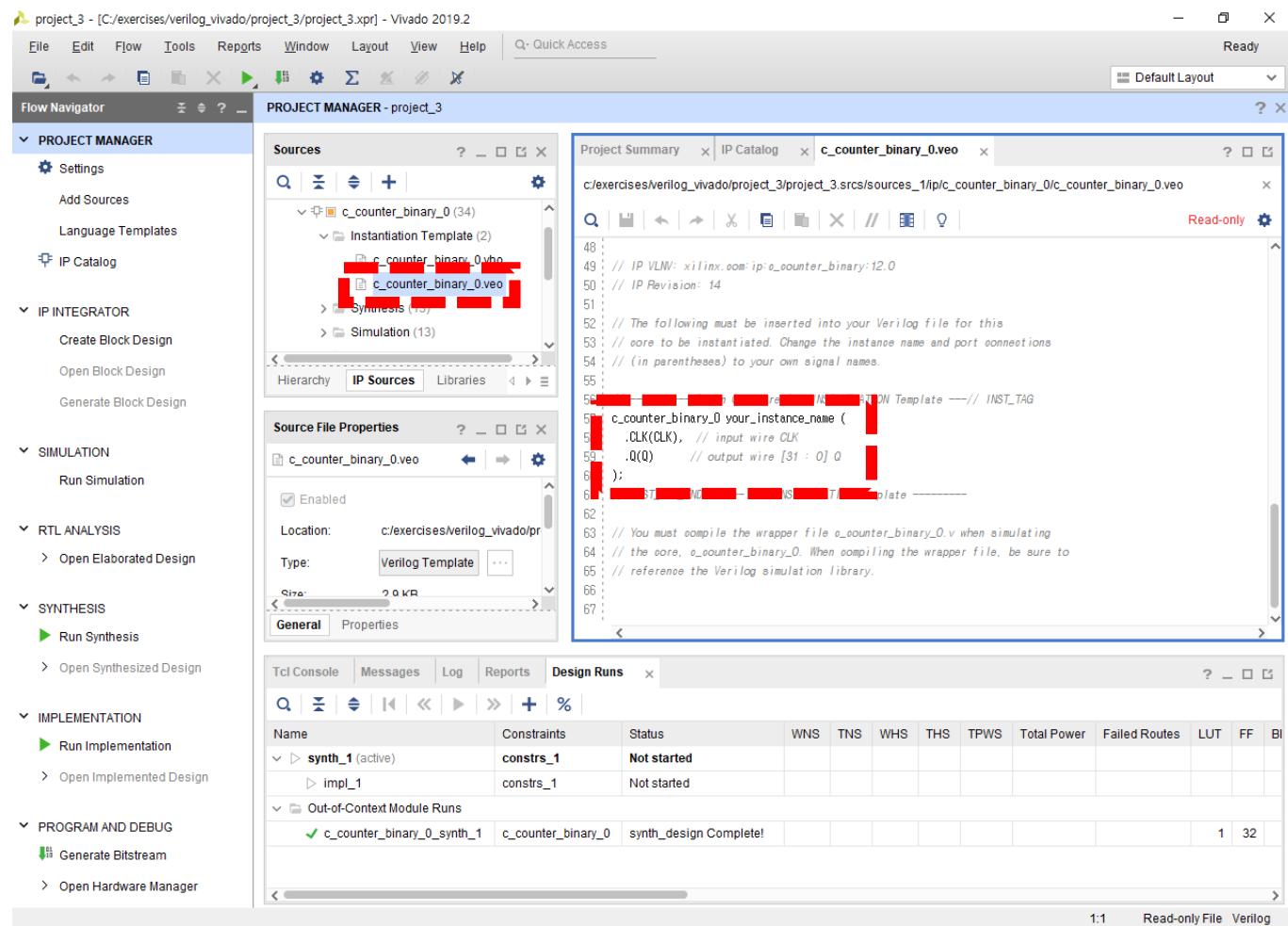
Step 3 Add Top Module



❖ Chapter 2 Vivado Design

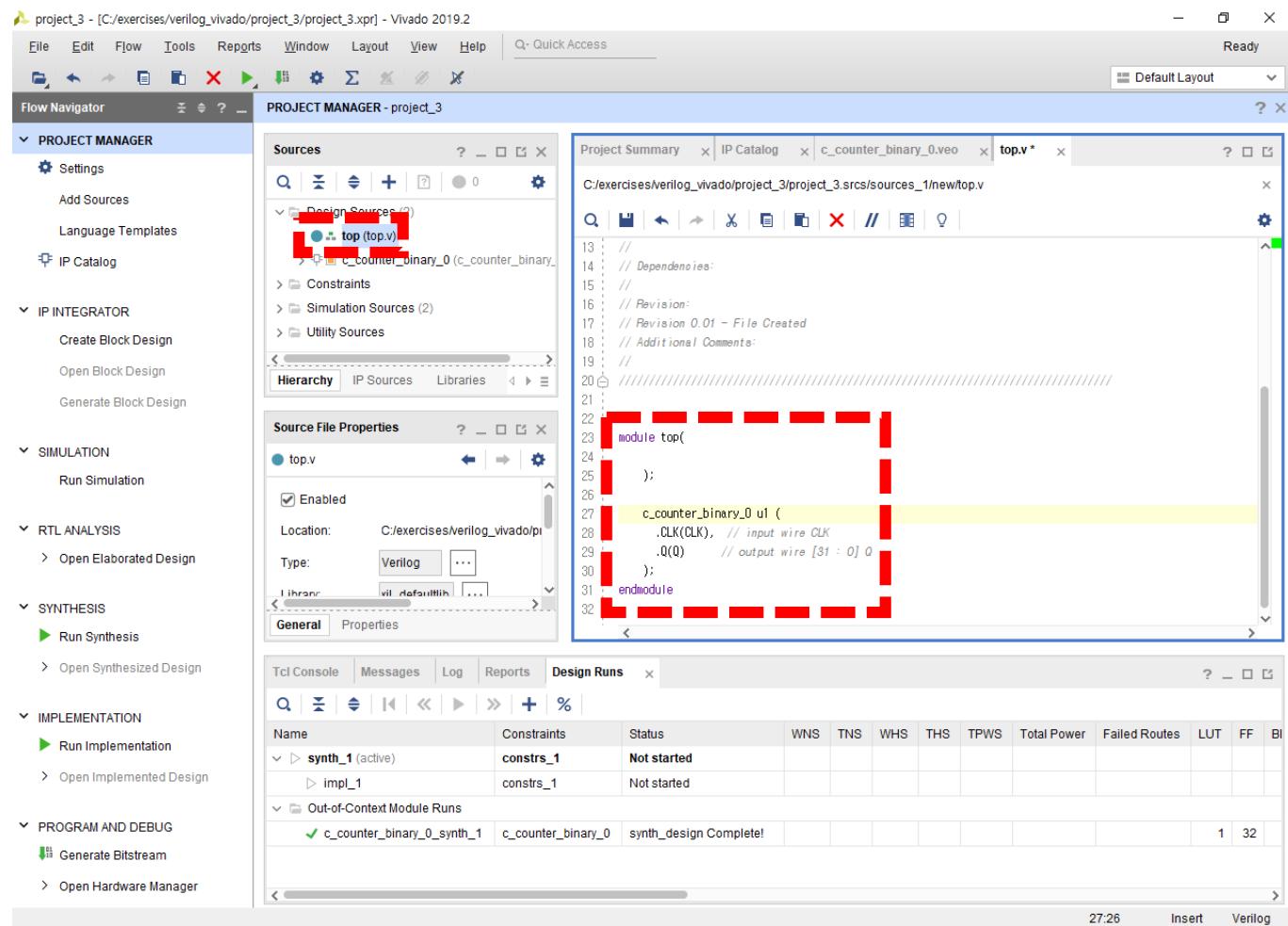
Flow의 Step 2에서 디자인 소스를 추가한 과정을 참고하여 top.v 소스파일을 추가한다.

Step 4 IP Core Instantiation 1



- ❖ Sources 창안에 IP Sources 탭을 클릭한 후 IP아래 $c_counter_binary_0 \Rightarrow$ Instantiation Template \Rightarrow $c_counter_binary_0.veo$ 파일을 더블 클릭한다.
- ❖ $c_counter_binary_0.veo$ 파일 안에 주석처리 되지 않은 부분을 복사하여 top.v파일에 붙여 넣기 한다.

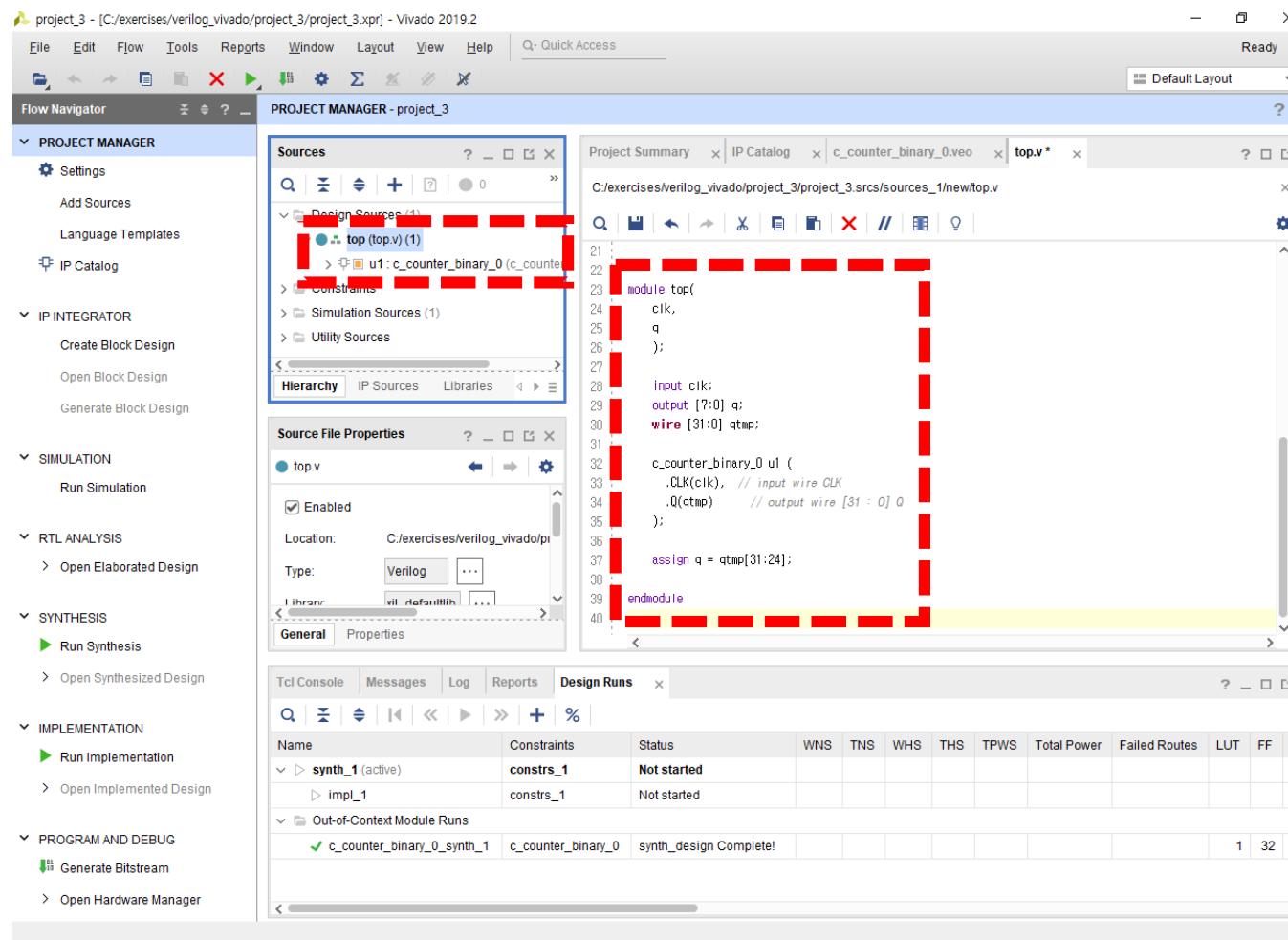
Step 4 IP Core Instantiation 2



❖ 붙여 넣기 한 내용 중

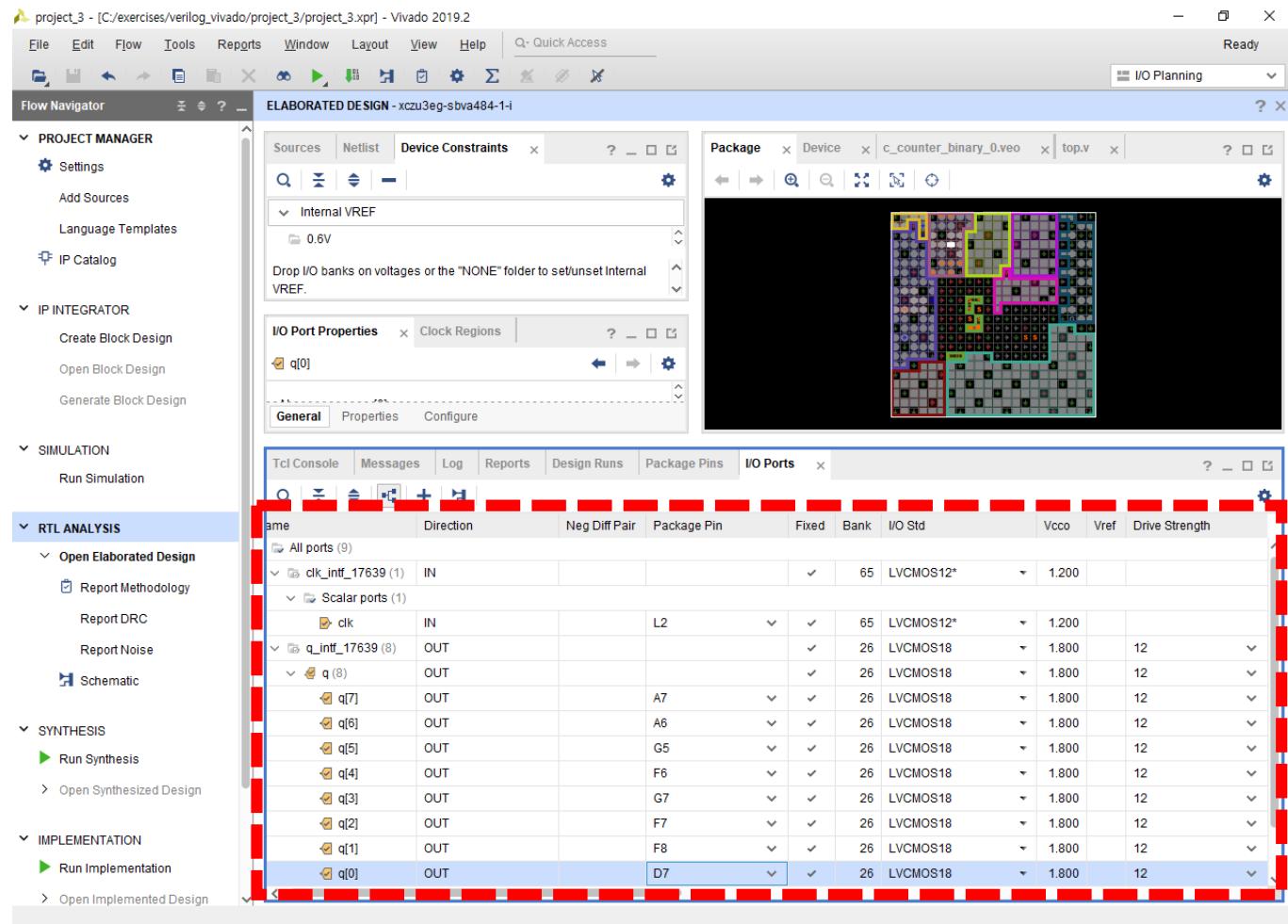
your_instance_name을 u1로 수정한다. (※ 해당 인스턴스의 기능에 따라서 인스턴스 이름을 만들어주는 것이 가독성을 위해서 좋지만 여기서는 편의상 인스턴스 이름을 u1으로 수정하였다.)

Step 5 Write Top Module Source Code



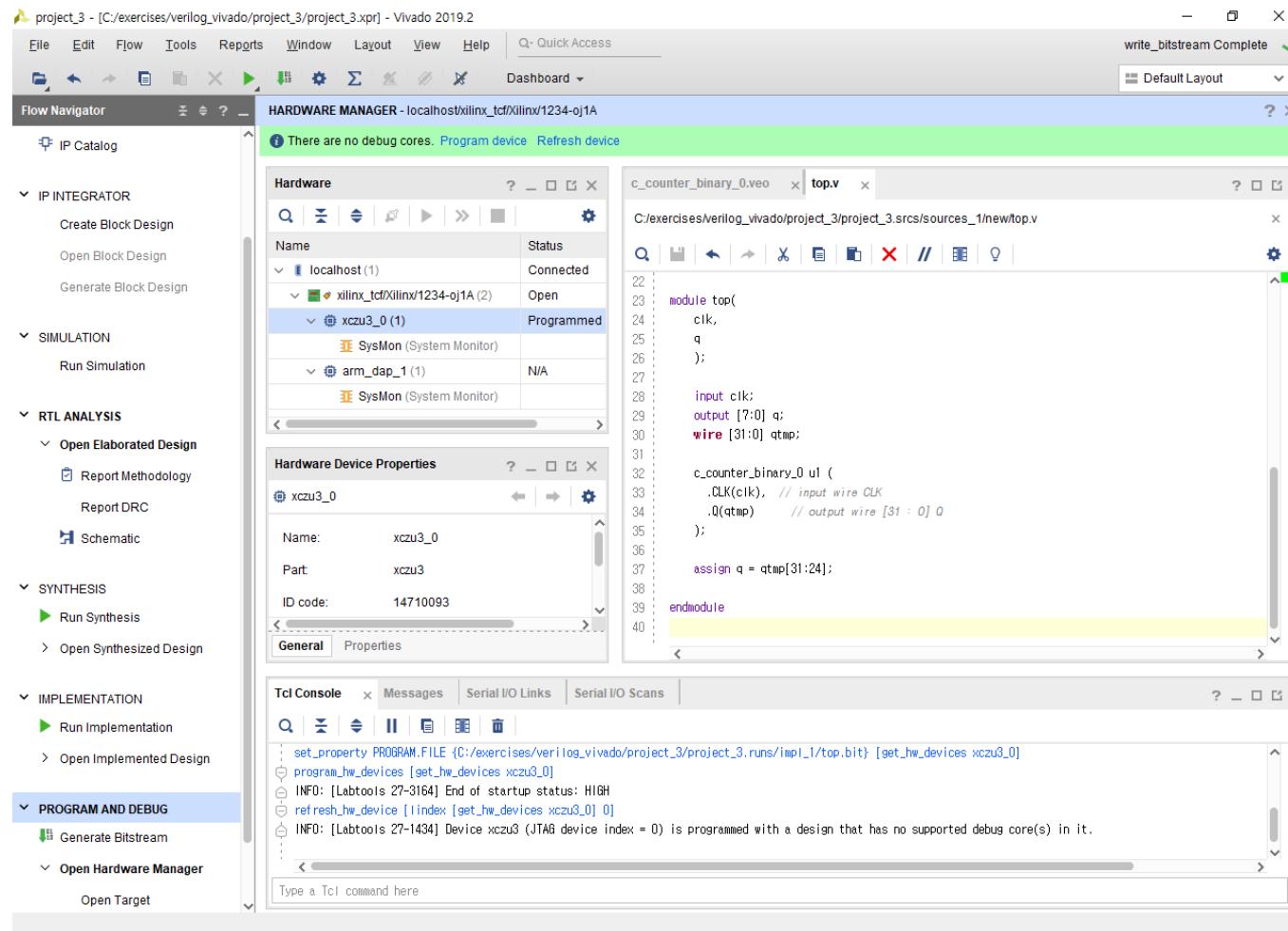
- ❖ top모듈의 포트리스트에 clk와 q를 추가한다.
- ❖ 포트선언부에 input clk; 와 output [7:0] q; 를 추가하고 데이터타입 선언부에 wire [31:0] qtmp를 선언한다.
- ❖ Counter IP Core Instantiation을 위해 포트를 clk와 qtmp로 수정하여 연결하고 qtmp의 상위 8-bit를 q에 assign 한다.
- ❖ top모듈을 저장하면 Sources 창에 생성한 Binary Counter IP가 top 모듈 아래로 들어온 것을 확인할 수 있다. (※ Binary Counter IP가 top 모듈의 하위 모듈임을 표현하기 위한 것이다.)

Step 6 Pin Constraints



- ❖ Flow Navigator ⇒ RTL ANALYSIS
⇒ Open Elaborated Design을 클릭하면 Elaboration이 진행된다.
- ❖ 하단 I/O Ports Windows에서 Pin Constraints를 한다. (※ clk은 L2 pin과 LVCMS12를 q[7:0]는 A7, A6, G5, F6, G7, F7, F8, D7과 LVCMS18을 선택하여 Pin Constraints 한다.)
- ❖ Ctrl+S키를 타이핑한 후 name에 counter를 입력하여 counter.xdc파일에 저장한다.

Step 7 Generating Bitstream and Programming Device



- ❖ Chapter 2 Vivado Design Flow의 Step 6를 참고하여 Bitstream을 생성한다.
- ❖ Chapter 2 Vivado Design Flow의 Step 7을 참고하여 디바이스를 프로그래밍한다.
- ❖ Pmod 8LD를 PMOD_A 커넥터에 연결한 후 디바이스 프로그래밍이 완료되면 Pmod 8LD의 LED가 카운팅 되는 값에 맞추어 계속 변하는 것을 확인할 수 있다.

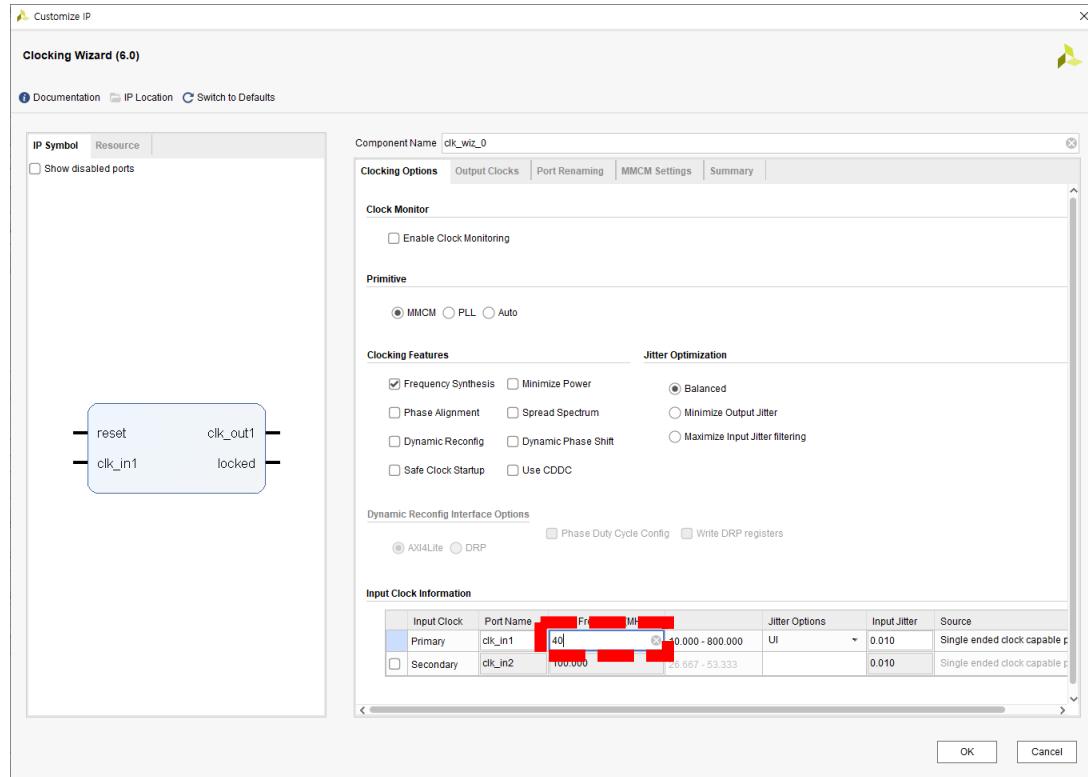
Chapter 4 Instantiation

- Hierarchical Design
- Instantiation
- IP Core Instantiation using Vivado
- Clocking Wizard
- Ultra96 Training Kit Exercises 2

Clocking Wizard 1

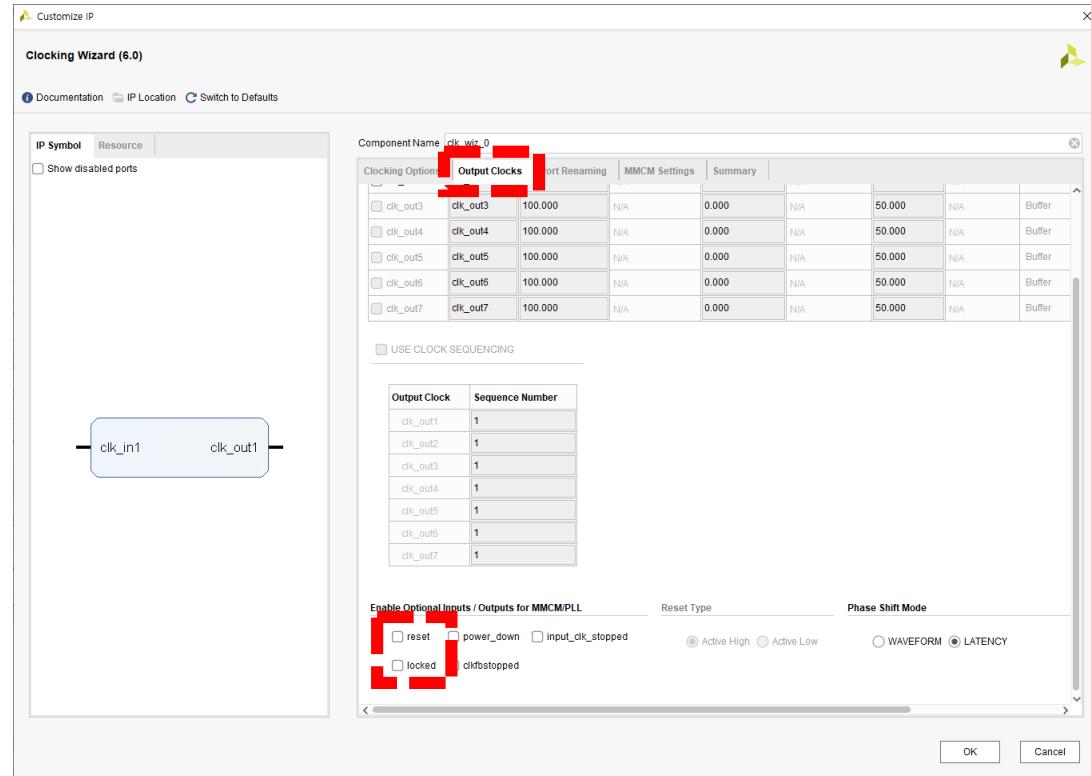
- ❖ 디지털 회로를 설계할 때 필요한 clock signal을 모두 FPGA 외부에서 공급받아 사용할 수는 없다.
- ❖ Xilinx에서 제공하는 Clocking Wizard IP Core를 사용하면 다양한 주파수, 위상, duty cycle을 가지는 clock signal을 클럭을 만들어 낼 수 있다.
- ❖ Clocking Wizard는 Mixed-mode clock manager(MMCM)나 phase-locked loop(PLL) primitive를 사용하여 클럭 생성 회로를 만들어낸다.
- ❖ Xilinx 디바이스를 사용할 때 clock signal을 직접 사용하기 보다는 Clocking Wizard IP Core를 사용하는 것을 추천한다.

Clocking Wizard 2



- ❖ IP Catalog에서 FPGA Features and Design ⇒ Clocking ⇒ Clocking Wizard를 더블 클릭하면 Clocking Wizard를 Customize할 수 있는 Window가 나온다.
- ❖ Ultra96 Training Kit에서는 외부클럭으로 40MHz 오실레이터를 사용하고 MPSoC Device의 L2핀에 연결되어 있다.
- ❖ Clocking Wizard를 Customize하기 위해선 먼저 Input Frequency(MHz) 입력란에 40을 입력해야 한다.

Clocking Wizard 3



- ❖ Output Clocks 탭을 클릭하면 출력할 clock signal에 대해서 설정할 수 있다.
- ❖ 디폴트로 100MHz 출력이 나가도록 되어 있는 것을 확인할 수 있다.
- ❖ 스크롤을 내려서 하단 부분에 reset과 locked가 디폴트로 체크되어 있는 것을 확인할 수 있다.
- ❖ reset signal은 Clocking Wizard를 리셋 할 수 있는 신호이다. (※ reset에 계속 High값이 들어오면 계속 리셋 되어 Clocking Wizard가 동작하지 않으니 주의 해야한다.)
- ❖ locked signal은 Low signal을 출력하다가 Clocking Wizard IP Core에서 출력하는 clock signal이 안정되면 High signal을 출력한다

Chapter 4 Instantiation

- Hierarchical Design
- Instantiation
- IP Core Instantiation using Vivado
- Clocking Wizard
- Ultra96 Training Kit Exercises 2

Ultra96 Training Kit Exercises 2

- ❖ EX2-1 Chapter 2 Vivado Design Flow에서 생성한 ander 모듈을 Instantiation하여 2개의 ander모듈을 가진 top모듈을 코딩하여 2개의 ander모듈이 동작하도록 프로그래밍 하시오. (※ PMOD_A 커넥터에 Pmod8LD를 PMOD_B 커넥터의 윗줄에 PmodBTN을 각각 연결하여 2개의 ander모듈이 구현되는지 확인할 수 있다.)
- ❖ EX2-2 Chapter 4 IP Core Instantiation using Vivado에서는 외부에서 입력되는 clock signal을 clk핀에 연결하여 바로 사용하였다. 여기에 Clocking Wizard를 추가하여 외부에서 입력되는 clock signal을 Clocking Wizard IP Core에 입력하고 Clocking Wizard에서 출력되는 clock signal을 Counter IP Core에 입력되도록 수정하여 프로그래밍 하시오. (※ 외부에서 입력되는 clock signal은 40MHz이다. 출력되는 clock signal의 주파수를 바꾸었을 때 카운터의 카운팅 속도가 변하는지 확인하면 정상적으로 구현된 것을 확인할 수 있다.)

Chapter 5 Behavioral Modeling

- Procedural Block
- Behavioral Statements
- Procedural Assignment
- Ultra96 Training Kit Exercises 3

Concurrent Statement vs Sequential Statement

- ❖ Structural Modeling으로 설계했던 Statement는 Concurrent Statement라고 부르고 Behavioral Modeling으로 설계하는 Statement는 Sequential Statement라고 부른다.
- ❖ Synthesis Tool은 Concurrent Statement를 동시에 실행되는 것으로 해석하고 Sequential Statement는 순차적으로 실행되는 것으로 해석하여 디지털 회로를 생성한다.
- ❖ Concurrent Statement들은 모두 동시에 실행되는 것으로 해석되므로 Statement들의 순서가 바뀌어도 똑같은 회로가 생성되지만 Sequential Statement는 순차적으로 실행되는 것으로 해석되므로 Statement들의 순서가 바뀌면 다른 회로가 생성된다.
- ❖ Procedural Block밖에 있는 Statement들은 Concurrent Statement이고 Procedural Block 안에 있는 Statement들은 Sequential Statement이다.

Procedural Block

- ❖ Procedural Block에는 initial과 always 두 가지가 있다.
- ❖ Synthesis Tool은 initial Block을 초기값을 주는 Block으로만 해석하고 always Block을 Sequential Statement들로 디지털 회로를 표현한 Block으로 해석한다.
- ❖ Simulation Tool은 initial Block과 always Block을 순차적으로 Stimulus를 주기위한 Block으로 해석한다. (※ initial Block은 Block안의 Statement를 한 번만 실행하는 것으로 해석하고 always Block은 Block안의 Statement를 반복적으로 실행하는 것으로 해석한다.)

Initial Block

- ❖ 합성되는 모듈에서 initial block은 초기 값 을 주기 위해 사용한다.

```
module ex_rom(  
    input [2:0] addr,  
    output [7:0] d_out );  
  
reg [7:0] data [0:7];  
  
initial  
begin  
    data[0] = 8'b10100010;  
    data[1] = 8'b00000000;  
    data[2] = 8'b11010101;  
    data[3] = 8'b11111011;  
    data[4] = 8'b10100010;  
    data[5] = 8'b10111011;  
    data[6] = 8'b01111010;  
    data[7] = 8'b11010101;  
end  
assign d_out = data[addr];  
endmodule
```

always Block

- ❖ addr신호를 입력받아서 addr의 값을 디코딩하여 d_out으로 출력하는 모듈을 표현한 것이다.
- ❖ always block은 always @ (-) 으로 시작하는데 여기서 괄호 안에 들어가는 신호를 Sensitivity List라고 부른다.
- ❖ Sensitivity List는 이 리스트에 포함된 신호가 변화한 것을 감지하면 always block의 내용을 실행한다는 의미이다.
- ❖ always block 안에서 사용하는 신호 중 모든 입력 신호들을 넣어주거나 모든 신호를 표현하는 "*"을 넣어준다.
 - ❖ always @ (*))
- ❖ 여러 개의 신호를 Sensitivity List에 넣는 때에는 그림 "," 또는 "or" 로 분리하여 넣어준다.
 - ❖ always @ (a,b,c)
 - ❖ always @ (a or b or c)

```
always @ (addr)
begin
    case (addr)
        3'b000 : d_out <= 8'b10100010;
        3'b001 : d_out <= 8'b00000000;
        3'b010 : d_out <= 8'b11010101;
        3'b011 : d_out <= 8'b11111011;
        3'b100 : d_out <= 8'b10100010;
        3'b101 : d_out <= 8'b10111011;
        3'b110 : d_out <= 8'b01111010;
        3'b111 : d_out <= 8'b11010101;
    default : d_out <= 8'b00000000;
    endcase
end
```

Chapter 5 Behavioral Modeling

- Procedural Block
- Behavioral Statements
- Procedural Assignment
- Ultra96 Training Kit Exercises 3

Behavioral Statements

- ❖ Behavioral Modeling을 위해 순차적으로 해석되어 지는 Procedural Block안에 코딩하는 Statement를 의미하며 앞서 이야기한 Sequential Statement와 같은 의미이다.
- ❖ Verilog에는 If, Case, Loop와 같은 Behavioral Statements가 있다.
- ❖ If, Case, Loop와 같은 Behavioral Statements의 문법은 상당 부분 C문법과 비슷하다. (※ C에서 중괄호({ })를 사용하는 것과 달리 Verilog에서는 begin, end를 사용하는 것이 대표적으로 다른 부분이고 대부분의 문법은 비슷하거나 같다.)

if Statement

- ❖ if Statement는 if안에 if를 중첩하여 사용하는 것이 가능하다.
- ❖ 여러 개의 표현을 if의 조건으로 사용할 때 && 또는 || 논리 연산자를 사용한다.
- ❖ if Statement안에 쓰여진 Statement가 2개 이상이면 begin end로 열고 닫아야 하며 1개일 때는 생략 가능하다.
- ❖ if Statement를 사용할 때는 else Statement를 사용하여 모든 조건에 대해서 기술해 주는 것이 좋다.

```
always @ (a or b or c or d)
begin
    if (a)
        begin
            e <= c;
            f <= d;
        end
    else if (b)
        begin
            g <= c;
            h <= d;
        end
    else
        i <= c;
end
```

case Statement

❖ case Statement는 여러 개의 값을 한 번에 조건을 비교하여 해당 Statement를 실행 한다. (※ if Statement는 if Statement가 쓰여진 순서대로 조건을 비교하기 때문에 하드웨어적으로 중첩된 Multiplexer가 되지만 case Statement는 한 번에 조건을 비교하기 때문에 하나의 Multiplexer가 필요하여 하드웨어 성능면에서 좋은 결과를 가져올 수 있다.)

```
always @ (a)
begin
    case (a)
        2'b00 : begin
            d_out <= 8'b10100010;
            d_out2 <= 8'b00000000;
        end
        2'b01 : begin
            d_out <= 8'b10100010;
            d_out2 <= 8'b00000000;
        end
        2'b10 : begin
            d_out <= 8'b10100010;
            d_out2 <= 8'b00000000;
        end
        2'b11 : begin
            d_out <= 8'b10100010;
            d_out2 <= 8'b00000000;
        end
        default : begin
            d_out <= 8'b00000000;
            d_out2 <= 8'b00000000;
        end
    endcase
end
```

casez Statement

- ❖ 'a'의 값이 1'b0, 1'b1 또는 1'bx이면 해당 Statement가 실행된다.
- ❖ 'a'의 값이 1'bz이면 1'b0에 해당하는 첫 번째 Statement가 실행된다. (※ casez는 1'bz를 don't care 값으로 취급한다.)
- ❖ 이 예제에서 1'bz에 해당하는 Statement는 실행되지 않는다.

```
always @ (a)
begin
    casez (a)
        1'b0 : d_out <= 8'b00000000;
        1'b1 : d_out <= 8'b00001111;
        1'bx : d_out <= 8'b11110000;
        1'bz : d_out <= 8'b11111111;
    endcase
end
```

casex Statement

- ❖ 'a'의 값이 1'b0 또는 1'b1이면 해당 Statement가 실행된다.
- ❖ 'a'의 값이 1'bx 또는 1'bz이면 1'b0에 해당하는 첫번째 Statement가 실행된다. (※ casex는 1'bx와 1'bz를 don't care 값으로 취급한다.)
- ❖ 이 예제에서 1'bx와 1'bz에 해당하는 Statement는 실행되지 않는다.

```
always @ (a)
begin
    casex (a)
        1'b0 : d_out <= 8'b00000000;
        1'b1 : d_out <= 8'b00001111;
        1'bx : d_out <= 8'b11110000;
        1'bz : d_out <= 8'b11111111;
    endcase
end
```

Loop Statement

- ❖ Verilog에는 forever, repeat, while, for와 같은 4가지의 Loop Statements가 있다.
- ❖ forever Statement는 무한루프로 계속해서 반복되는 구문으로 주로 시뮬레이션에서 클럭과 같이 반복되는 입력신호를 만들 때 사용한다.
- ❖ repeat Statement는 일정 횟수만큼 반복하는 구문이다.
- ❖ while Statement는 expression이 참일 때는 계속 반복하다가 거짓이 되면 끝내는 구문이다.
- ❖ for Statement는 인덱스 값이 증가하거나 감소하게 하고 인덱스가 조건이 거짓이 되면 끝내는 구문이다.

```
forever #50 cnt = cnt + 1;  
  
repeat(8) begin  
    i = i + 1;  
    d_out[i+8] = d_in[i];  
end  
  
while(i<9)  
begin  
    i = i + 1;  
    d_out[i+8] = d_in[i];  
end  
  
for(i=0;i<9;i=i+1)  
begin  
    d_out[i+8] = d_in[i];  
end
```

Chapter 5 Behavioral Modeling

- Procedural Block
- Behavioral Statements
- **Procedural Assignment**
- Ultra96 Training Kit Exercises 3

Procedural Assignment

- ❖ Verilog의 Assignment는 Continuous Assignment와 Procedural Assignment 두 가지가 있다.
- ❖ Continuous Assignment는 Chapter 2에서 설명한 assign문으로 Procedural Block 밖에서 사용한 Continuous Statement이다.
- ❖ Procedural Assignment는 Procedural Block 안에서 사용한 Sequential Statement이다.
- ❖ Procedural Assignment는 Sequential Statement이어서 Sequential Assignment로 부르기도 한다.
- ❖ Procedural Assignment에서 왼쪽에 오는 신호는 Register Data Type이어야 하며 그 외 다른 신호들은 Net Data Type이어야 한다.

Blocking Assignment

❖ Blocking Assignment는 '=' 기호를 사용한다.

❖ Blocking Assignment는 순차적으로 실행되는 것을 막는다는 의미이다.

❖ 예를 보면 Blocking Assignment는 다음 라인이 실행되는 것을 막아서 시간이 지연된다.

```
initial begin  
    a = #50 0;  
    a = #100 1;  
    a = #200 2;  
    a = #300 3;  
end
```

시간	실행
50	a = 0
150 = 50 + 100	a = 1
350 = 150 + 200	a = 2
650 = 350 + 300	a = 3

Nonblocking Assignment

- ❖ Nonblocking Assignment는 ' $<=$ ' 기호를 사용한다.
- ❖ Nonblocking Assignment는 순차적으로 실행되는 것을 막지 않는다는 의미이다.
- ❖ 예를 보면 Nonblocking Assignment는 다음 라인이 실행되는 것을 막지 않아서 시간 지연 없이 바로 다음 라인으로 진행된다.

```
initial begin  
    a <= #50 0;  
    a <= #100 1;  
    a <= #200 2;  
    a <= #300 3;  
end
```

시간	실행
50	a = 0
100	a = 1
200	a = 2
300	a = 3

Chapter 5 Behavioral Modeling

- Procedural Block
- Behavioral Statements
- Procedural Assignment
- Ultra96 Training Kit Exercises 3

Ultra96 Training Kit Exercises 3

- ❖ EX3-1 If Statement를 사용하여 스위치 상태에 따라 7-Segment에 다른 숫자가 표시되도록 프로그래밍 하시오. (※ PMOD_A 커넥터에 2x6-pin to Dual 6-pin Pmod Splitter Cable의 2x6-pin를 연결하고 Dual 6-Pin에 Pmod SSD를 연결하면 7-Segment 동작을 확인할 수 있다. 연결시에는 항상 VCC와 GND를 주의하여 연결해야 한다.)
- ❖ EX3-2 Case Statement를 사용하여 스위치 상태에 따라 7-Segment에 다른 숫자가 표시되도록 프로그래밍 하시오. (※ PMOD_A 커넥터에 2x6-pin to Dual 6-pin Pmod Splitter Cable의 2x6-pin를 연결하고 Dual 6-Pin에 Pmod SSD를 연결하면 7-Segment 동작을 확인할 수 있다. 연결시에는 항상 VCC와 GND를 주의하여 연결해야 한다.)

Chapter 6 Simulation

- **Testbench**
- **Vivado Simulation**
- **Simulation Exercises**

Simulation

- ❖ 시뮬레이션은 우리말로 모의실험 또는 가상실험이라는 말이다.
- ❖ 실존하는 객체가 아닌 컴퓨터 안에 가상의 객체를 만들어서 그 객체에 여러 가지 실험을 하는 것을 의미한다.
- ❖ 여기서 객체는 우리가 검증하고자 하는 하드웨어 모듈을 말한다.
- ❖ 하드웨어 모듈에 적절한 입력 신호를 넣어주는 코드를 만든 후 시뮬레이션 툴을 통해 시뮬레이션을 실행하면 시뮬레이션 툴은 출력 신호를 보여주는데 출력되는 신호를 보고 하드웨어 모듈의 동작을 검증한다.
- ❖ 여기서 적절한 입력 신호를 넣어주는 코드를 Testbench라고 부른다.

Testbench

- ❖ Testbench 모듈이 이름은 관습적으로 검증하고자 하는 모듈의 이름 뒤에 _tb라는 이름을 붙여서 짓는다.
- ❖ Testbench는 검증하려는 모듈의 입력 신호에 적절한 입력만 넣어주면 되므로 외부와 인터페이스를 할 필요가 없어서 포트가 없다.
- ❖ 검증하고자 하는 모듈을 Instantiation하기 위해 필요한 신호들을 선언해 준다.
- ❖ 검증하고자 하는 모듈을 Instantiation 한다.
- ❖ 검증하고자 하는 모듈의 입력포트에 인가할 적절한 입력 신호들을 만들어 준다.
- ❖ `timescale에서 정의한 1 ns는 reference time unit으로 #뒤에 오는 숫자의 시간 단위를 의미한다.
- ❖ `timescale에서 정의한 1 ps는 resolution으로 시뮬레이션 툴이 데이터를 업데이트하는 시간 단위를 말한다.

```
`timescale 1 ns / 1 ps

module ander_tb;
    reg a, b;
    wire result;
    ander uut (.a(a),.b(b),.result(result));
    initial begin
        a=0;    b=0;
        #200;
        a=0;    b=1;
        #200;
        a=1;    b=1;
    end
endmodule
```

Stimulus

- ❖ Testbench에서 입력 신호들을 넣어 주는 것을 자극을 준다는 의미로 Stimulus라고 부른다.
- ❖ Stimulus를 줄 때 initial block 또는 always block을 사용할 수 있다.
- ❖ Initial block은 한 번 실행되고 always block은 반복 실행된다.
- ❖ 카운터를 사용하면 값이 계속 증가하는 Stimulus를 만들 수 있다.

```
module tb;  
    reg clk;  
    always begin  
        #50 clk = 1'b1;  
        #50 clk = 1'b0;  
    end  
endmodule
```

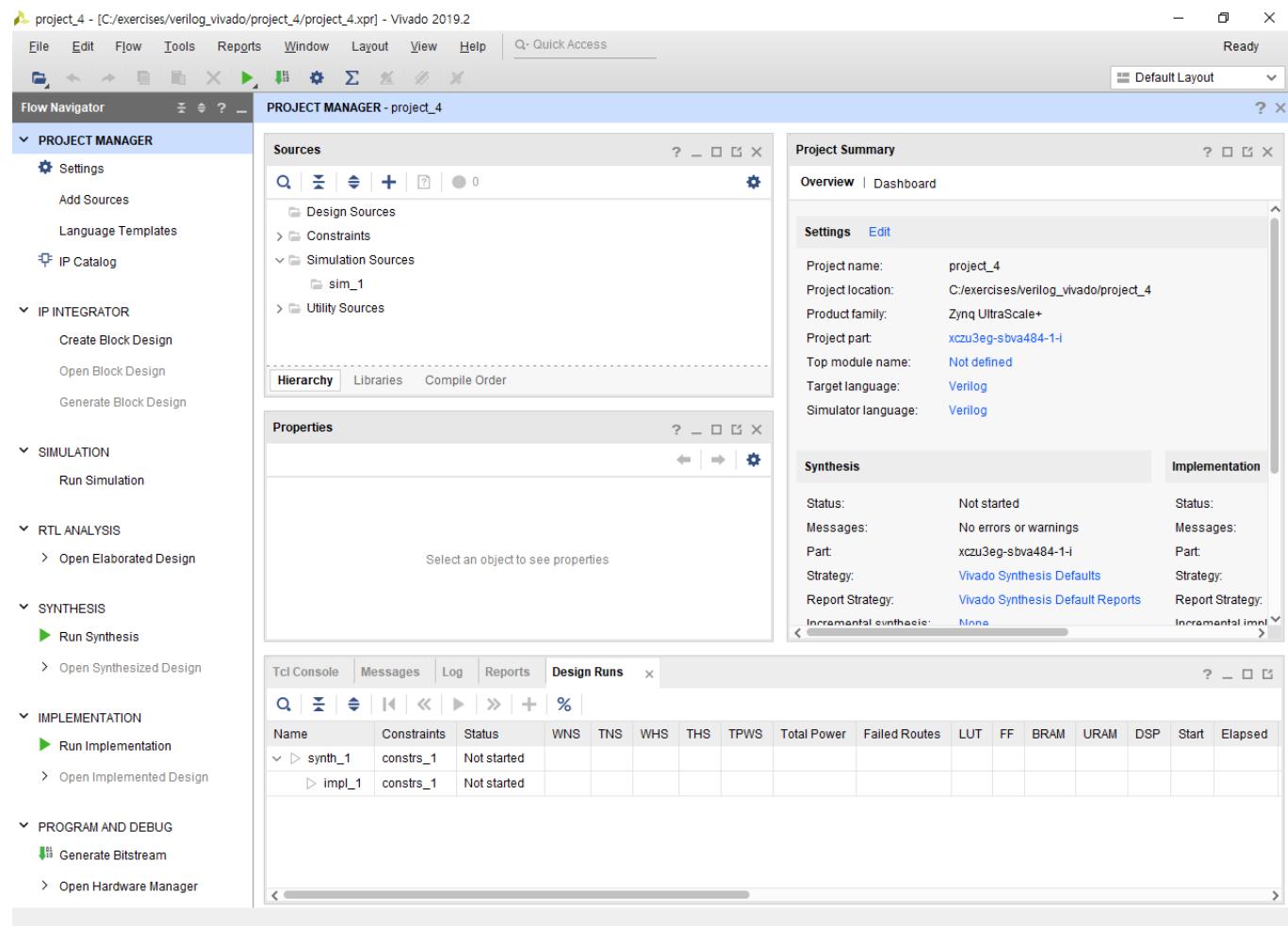
```
module tb;  
    reg clk = 1'b0;  
    always  
        #50 clk = ~clk;  
    endmodule
```

```
module tb;  
    reg [7:0] cnt = 8'h00;  
    always  
        #50 cnt = cnt + 1;  
    endmodule
```

Chapter 6 Simulation

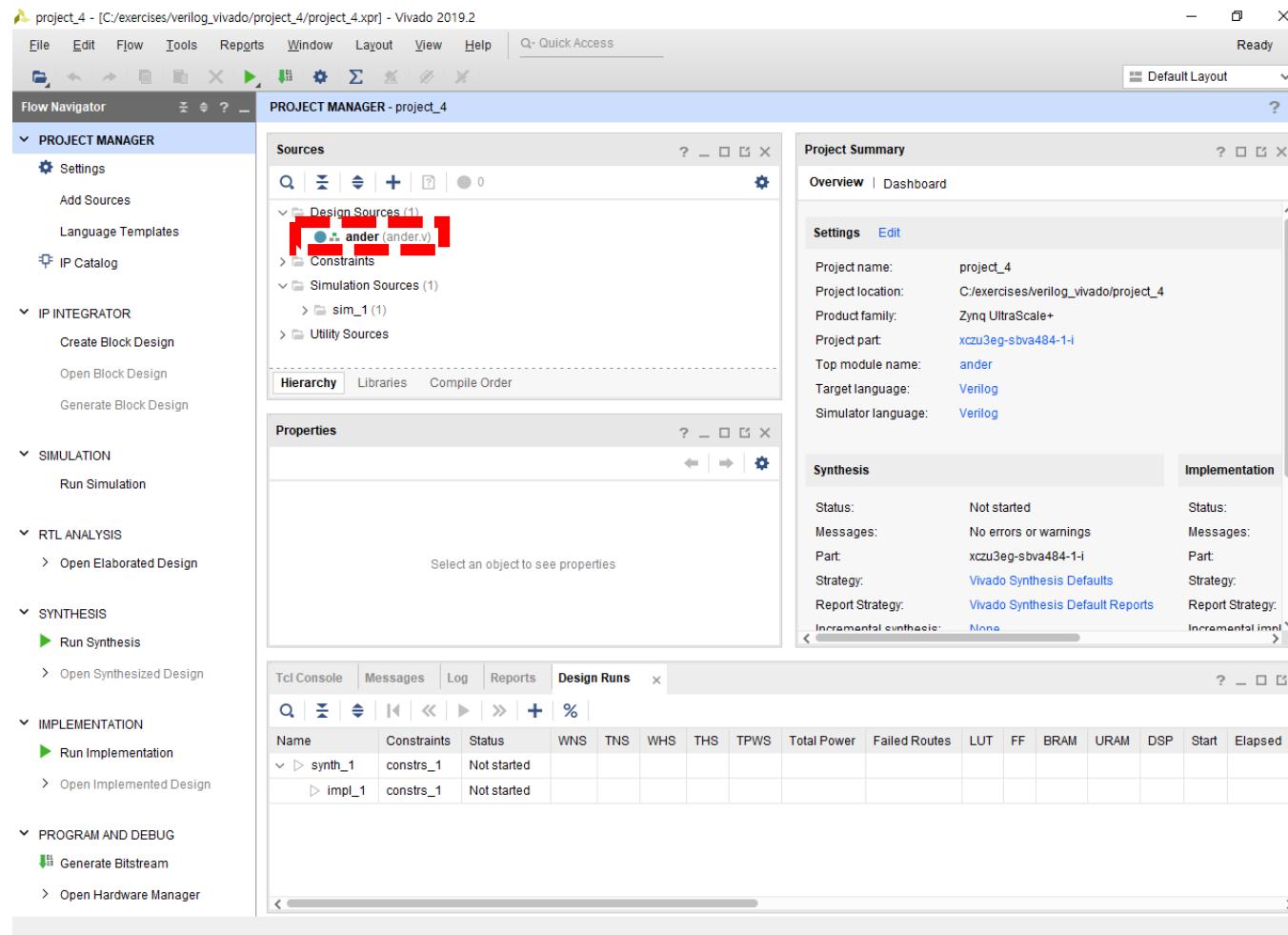
- Testbench
- Vivado Simulation
- Simulation Exercises

Step 1 Creating Vivado Project



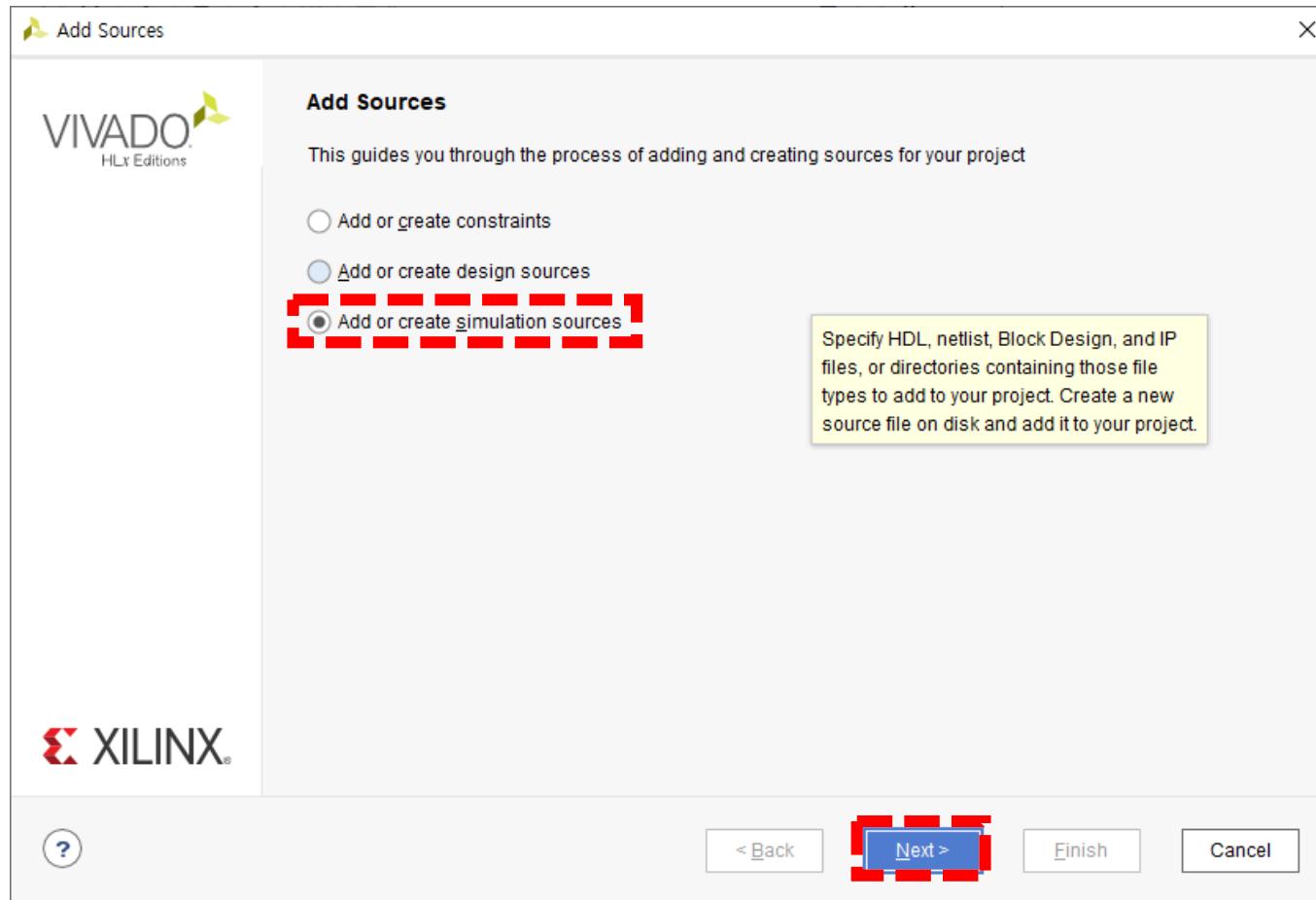
❖ Chapter 1의 Vivado 프로젝트 만들기를 참조하여 project_4 프로젝트를 만든다.

Step 2 Add Design Source in Vivado Project



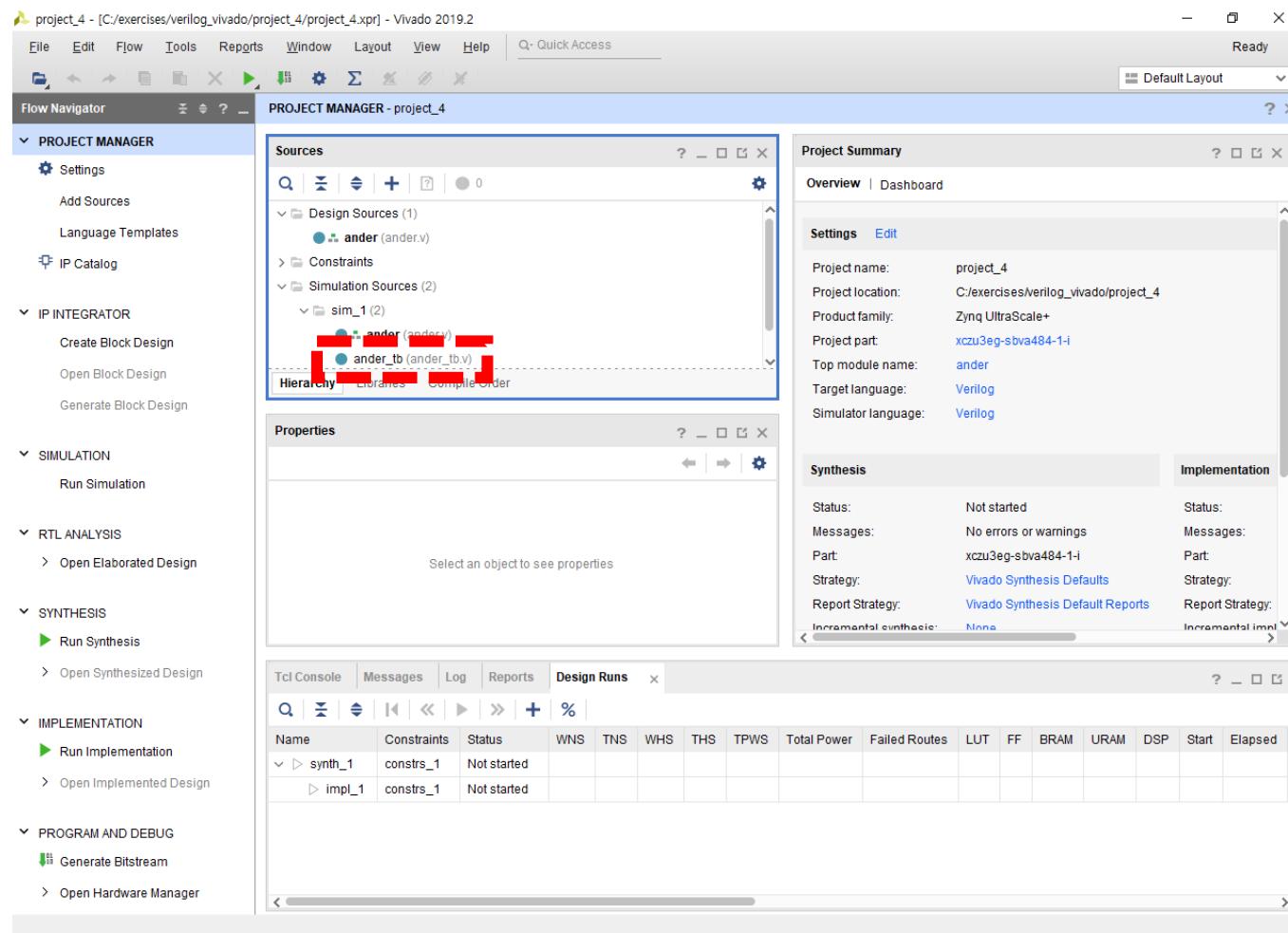
❖ Chapter 2 Vivado Design
Flow의 Step 2를 참고하여
ander.v 소스파일을 프로젝트
에 추가한다.

Step 2 Add Simulation Source in Vivado Project 1



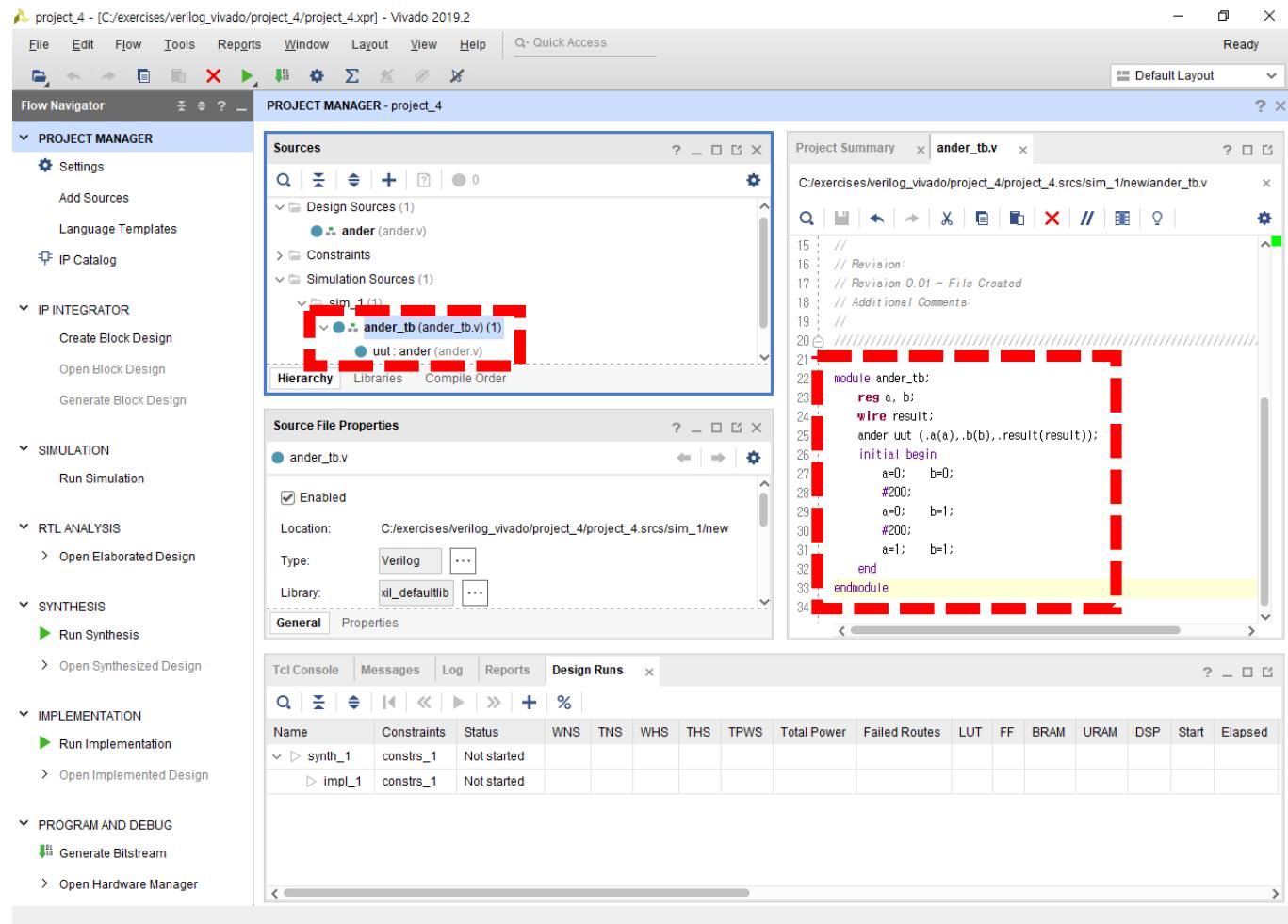
- ❖ Flow Navigator ⇒ Project Manager ⇒ Add Sources를 클릭한다.
- ❖ Add or create simulation sources를 클릭하여 선택한 후 Next 버튼을 클릭한다.

Step 2 Add Simulation Source in Vivado Project 3



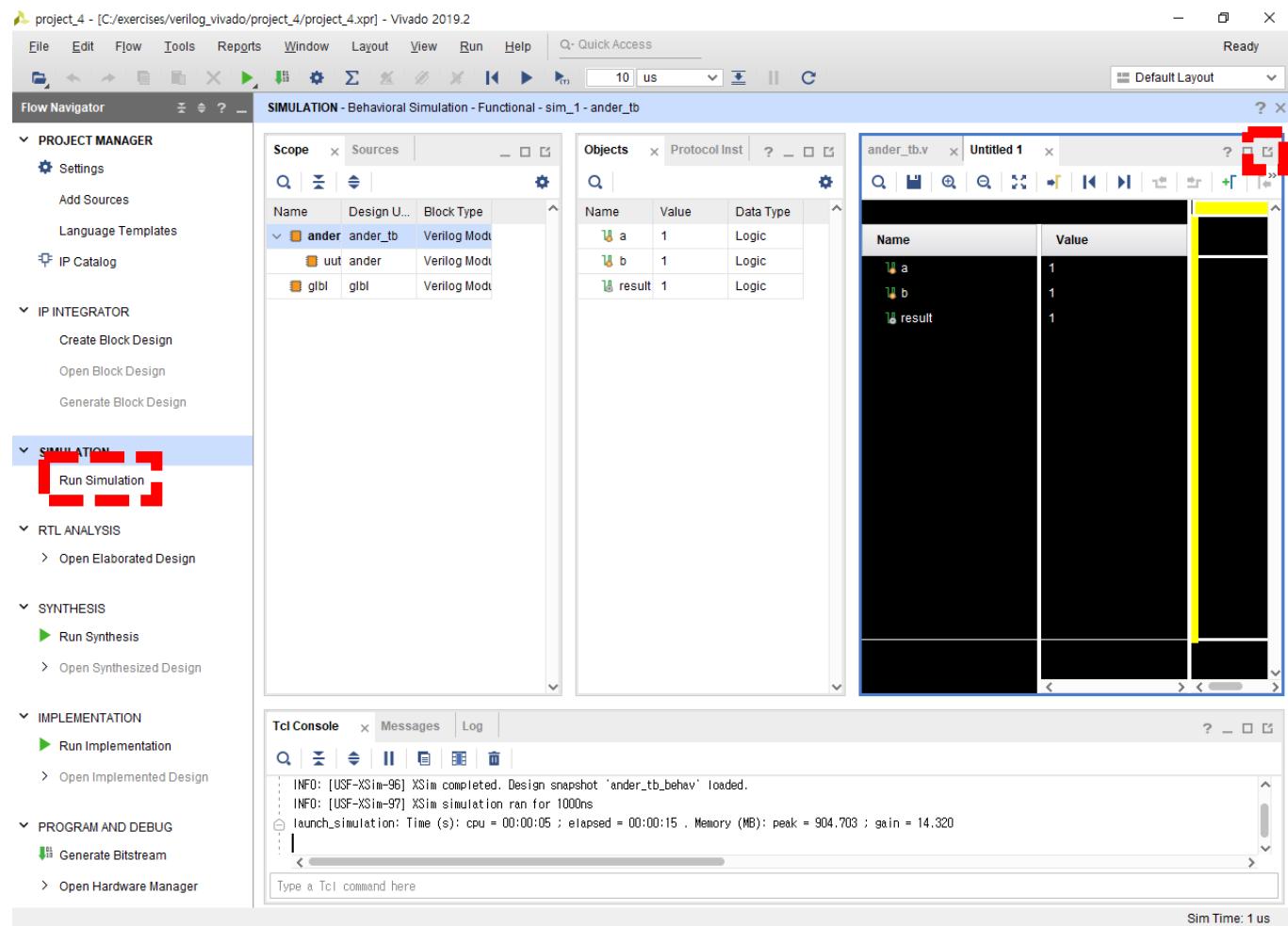
❖ Sources Windows 안에
Simulation Sources 아래에
ander_tb가 추가된 것을 확인
할 수 있다.

Step 3 Write Testbench Code



- ❖ `ander_tb`를 더블클릭하여 Editor Windows를 연다.
- ❖ Chapter 6 Testbench에서 예제로 사용한 `ander_tb` 모듈을 코딩한다.
- ❖ `ander_tb`가 `ander`를 instantiation하여 Sources Windows에 `ander_tb` 아래 `uut: ander`가 들어간 것을 확인할 수 있다.

Step 4 Run Simulation 1



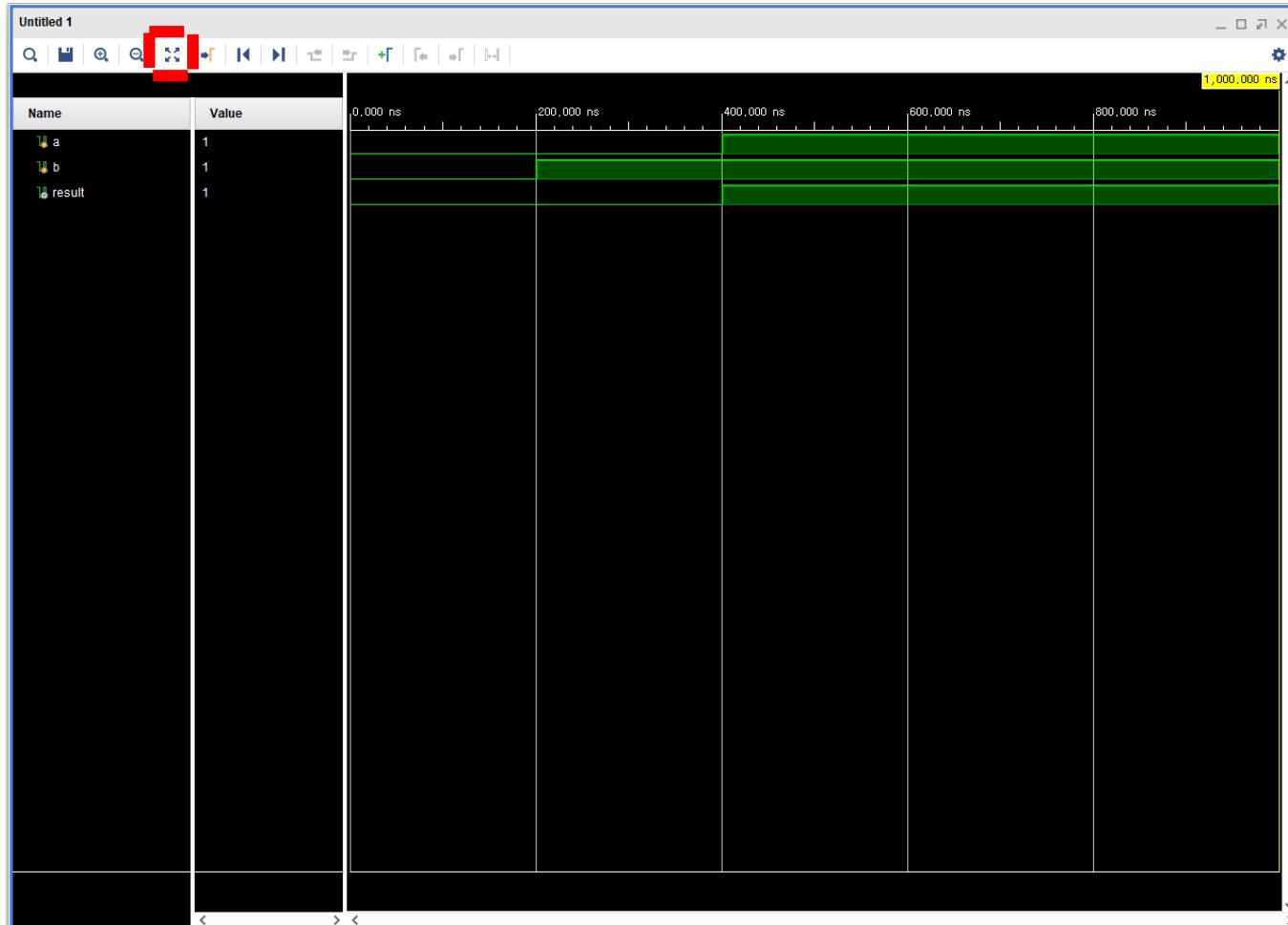
❖ Flow Navigator ⇒ Simulation

⇒ Run Simulation을 클릭한 후 Run Behavioral Simulation 을 클릭한다.

❖ Simulation 결과를 확인하기

쉽도록 우측에 있는 Simulation 결과 창의 우측 상단에 있는 Float 버튼을 클릭한다.

Step 4 Run Simulation 2



- ❖ Simulation Window 상단의 Zoom Fit 버튼을 클릭하여 Simulation을 실행한 시간에 맞추도록 한다.
- ❖ a, b신호는 Testbench에서 입력한 신호들인 것을 확인할 수 있다.
- ❖ a, b 입력신호가 변할 때 result 출력 신호를 표시해 준다.
- ❖ 출력신호가 설계 의도한 디지털 회로의 출력과 동일한 결과가 나오는지를 확인하여 디지털 회로를 검증을 할 수 있다.

Chapter 6 Simulation

- Testbench
- Vivado Simulation
- **Simulation Exercises**

Simulation Exercises

- ❖ EX4-1 Ultra96 Training Kit Exercises 1에서 만든 모듈 중 하나를 선택하여 그 모듈의 Testbench를 작성하고 시뮬레이션을 통해 검증하시오.
- ❖ EX4-2 Ultra96 Training Kit Exercises 2에서 만든 모듈 중 하나를 선택하여 그 모듈의 Testbench를 작성하고 시뮬레이션을 통해 검증하시오.
- ❖ EX4-3 Ultra96 Training Kit Exercises 3에서 만든 모듈 중 하나를 선택하여 그 모듈의 Testbench를 작성하고 시뮬레이션을 통해 검증하시오.

Chapter 7 Sequential Circuit

- Register
- Counter
- Ultra96 Training Kit Exercises 4

Flip-Flop

- ❖ Flip-Flop은 기본적으로 Clock(clk)과 Data(d) 입력 신호와 Data(q) 출력 신호를 가지고 있다.
- ❖ Flip-Flop은 기본적으로 Clock신호의 Rising Edge 또는 Falling Edge 순간에 입력 Data 신호를 출력하도록 동작한다.
- ❖ Verilog에서는 always Block의 Sensitivity List에 posedge 또는 negedge를 넣어서 Rising Edge 또는 Falling Edge에서 동작하는 것을 표현한다.
- ❖ 예제 코드를 보면 Clock의 Edge순간에 입력 신호 d가 출력 신호 q에 인가되도록 설계하였다.

```
module flipflop(clk,d,q);  
  
    input clk,d;  
    output reg q;  
  
    always @ (posedge clk)  
    begin  
        q <= d;  
    end  
  
endmodule
```

Register

- ❖ Flip-Flop을 데이터 크기에 따라서 16개 또는 32개의 Flip-Flop이 하나의 Clock 신호에 동기를 맞추어서 동작하는 디지털 회로를 Register라고 한다.
- ❖ 예제 코드를 보면 Clock의 Edge순간에 입력 신호 d가 출력 신호 q에 인가되도록 설계하였다. (※ 코드 상에서 Flip-Flop과 다른 점은 입출력신호의 크기만 다르고 다른 부분은 동일하다.)

```
module reg16_ex1(clk,d,q);  
  
    input clk;  
    input [15:0] d;  
    output reg [15:0] q;  
  
    always @ (posedge clk)  
    begin  
        q <= d;  
    end  
  
endmodule
```

RTL(Register Transfer Level) Circuit



- ❖ RTL(Register Transfer Level)은 Register에서 Register로 데이터가 이동한다는 의미이다.
- ❖ RTL로 설계한 디지털 회로를 RTL Circuit라고 하며 Data가 순차적으로 업데이트 된다고 하여 **Sequential Circuit**이라고도 한다.
- ❖ 대부분의 디지털 회로는 RTL Circuit로 설계되어 있다.
- ❖ RTL Circuit는 Clock의 Edge순간에 Register에 입력된 신호를 출력하고 출력된 신호는 Combinatorial Circuit로 입력되고 Combinatorial Circuit에서 출력된 신호는 다른 Register의 입력단에 도달하며 Clock의 Edge순간이 되면 다시 이 Register의 출력으로 나가도록 동작한다.

Register 2

- ❖ Clear 신호를 추가한 Register이다.
- ❖ Sensitivity List에 posedge clr이 추가되어 clr신호가 Rising Edge 순간에 always Block안의 내용이 실행되는데 clr신호가 Rising Edge는 '1'(High)이 되므로 출력 신호 q는 16'h0000으로 초기화 된다.
- ❖ Sensitivity List에 posedge clr가 생략되면 clr신호는 clk신호의 Rising Edge에서만 동작하여 Synchronous Circuit이 되고 예제 코드와 같이 그대로 posedge clr이 포함되어 있으면 clk신호의 Rising Edge와 상관없이 Clear를 하게 되어 Asynchronous Circuit이 된다.
- ❖ Synchronous Circuit과 Asynchronous Circuit 의 구분은 Data 신호가 Clock Edge에서만 변하는지에 따라서 구분된다. (※ Data 신호가 Clock Edge에서만 변하면 Synchronous Circuit이고 Clock Edge가 아닌 곳에서도 변하면 Asynchronous Circuit이다.)

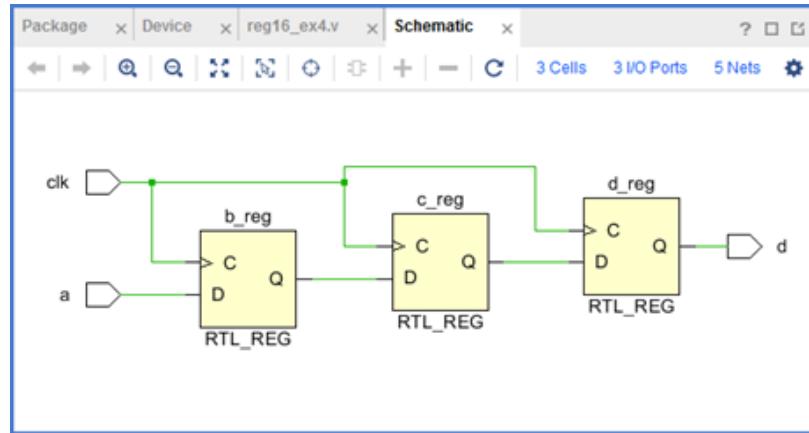
```
module reg16_ex2(clk,clr,d,q);  
  
    input clk,clr;  
    input [15:0] d;  
    output reg [15:0] q;  
  
    always @ (posedge clk or posedge clr)  
    begin  
        if(clr)  
            q <= 16'h0000;  
        else  
            q <= d;  
    end  
  
endmodule
```

Register 3

- ❖ Clock Enable 신호를 추가한 Register이다.
- ❖ 이 코드 예제에서 Clear 신호(clr)와 Clock Enable 신호(clken)들은 Synchronous하게 동작하도록 설계되어 있다.
- ❖ Clock Enable 신호(clken)가 '0'(Low)이면 입력 신호(d)가 출력 신호(q)에 인가되지 않고 '1'(High)일 때만 인가되도록 하여 Clock Enable을 설계하였다.

```
module reg16_ex3(clk,clr,clken,d,q);  
  
    input clk,clr,clken;  
    input [15:0] d;  
    output reg [15:0] q;  
  
    always @ (posedge clk)  
    begin  
        if(clr)  
            q <= 16'h0000;  
        else if(clken)  
            q <= d;  
    end  
  
endmodule
```

Register 4



- ❖ Nonblocking Assignment를 사용하여 Assign을 하면 위 Schematic 그림과 같이 Assign을 할 때마다 Register가 생성된다.
- ❖ Flow Navigator ⇒ RTL Analysis ⇒ Elaborated Design ⇒ Schematic을 클릭하면 위 Schematic 그림을 확인할 수 있다.

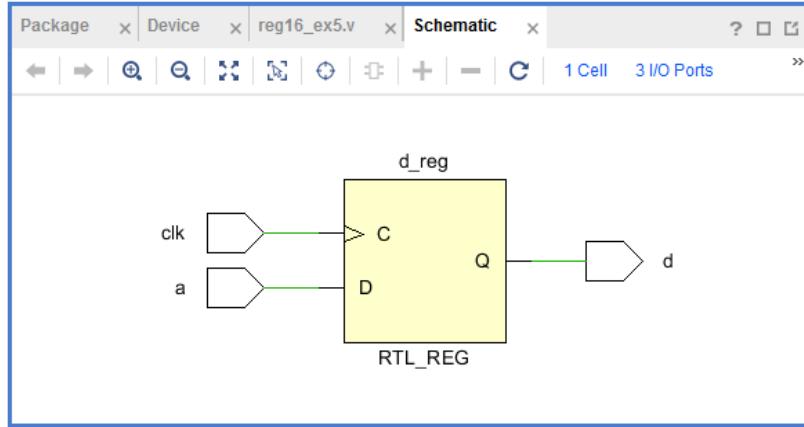
```
module reg16_ex4(clk,a,d);
```

```
input clk,a;  
output reg d;  
reg b,c;
```

```
always @ (posedge clk)  
begin  
    b <= a;  
    c <= b;  
    d <= c;  
end
```

```
endmodule
```

Register 5



❖ Blocking Assignment를 사용하여 Assign을 하면 위 Schematic 그림과 같이 Assign을 할 때마다 Register가 생성되지 않고 중복된 Net으로 간주되어 중간에 사용된 Net들인 b, c신호는 삭제되는 것을 알 수 있다.

```
module reg16_ex5(clk,a,d);
```

```
    input clk,a;  
    output reg d;  
    reg b,c;
```

```
    always @ (posedge clk)  
    begin  
        b = a;  
        c = b;  
        d = c;  
    end
```

```
endmodule
```

Chapter 7 Sequential Circuit

- Register
- Counter
- Ultra96 Training Kit Exercises 4

Counter 1

- ❖ Counter는 Clock Edge에 값을 증가 또는 감소하도록 동작하는 회로를 말한다.
- ❖ Counter는 일반적으로 Clock을 분주하여 사용하거나 시간을 재기 위한 용도로 사용된다.
- ❖ 이 코드 예제는 Clear 신호가 '1'(High)이면 16'h0000을 q포트로 출력하고 클리어 신호가 '0'(Low)이면 Clock Rising Edge에 q값을 1씩 증가한 후 q포트로 출력하는 기본적인 카운터이다.

```
module counter_ex1(clk,clr,q);  
  
    input clk,clr;  
    output reg [15:0] q = 16'h0000;  
  
    always @ (posedge clk)  
    begin  
        if(clr)  
            q <= 16'h0000;  
        else  
            q <= q + 1;  
    end  
  
endmodule
```

Counter 2

- ❖ 이 코드 예제와 같이 카운터를 만들면 출력포트 q는 0~999 값을 반복하게 되고 q의 값을 0으로 만드는 문장 아래에 해야 할 일을 넣어주면 특정 시간마다 한 번씩 실행되는 하드웨어를 만들 수 있다.
- ❖ 이런 Counter를 Modulo-N Counter라고 부른다.
- ❖ Modulo-N Counter는 Counter 값이 0~N-1을 반복한다.

```
module counter_ex2(clk,clr,q);  
  
    input clk,clr;  
    output reg [15:0] q = 16'h0000;  
  
    always @ (posedge clk)  
    begin  
        if(clr)  
            q <= 16'h0000;  
        else begin  
            q <= q + 1;  
            if(q==999) begin  
                q <= 16'h0000;  
                // 특정시간마다 해야 할 일  
            end  
        end  
    end  
  
endmodule
```

Counter 3

❖ cnten신호를 삽입하여 카운팅을 할지 말지 결정할 수 있는 컨트롤 신호를 삽입한 카운터 예제이다.

```
module counter_ex3(clk,clr,cnten,q);

    input clk,clr,cnten;
    output reg [15:0] q = 16'h0000;

    always @ (posedge clk)
    begin
        if(clr)
            q <= 16'h0000;
        else if(cnten) begin
            q <= q + 1;
            if(q==999) begin
                q <= 16'h0000;
                // 특정시간마다 해야 할 일
            end
        end
    end

endmodule
```

Counter 4

❖ updown신호를 삽입하여 카운트 값이 1씩
증가하는 카운트업을 할지 1씩 감소하는
카운트다운을 할지 결정하는 컨트롤 신호
를 삽입한 카운터 예제이다.

```
module counter_ex4(clk,clr,cnten,updown,q);  
  
    input clk,clr,cnten,updown;  
    output reg [15:0] q = 16'h0000;  
  
    always @ (posedge clk)  
    begin  
        if(clr)  
            q <= 16'h0000;  
        else if(cnten) begin  
            if(updown)  
                q <= q + 1;  
            else  
                q <= q - 1;  
        end  
    end  
end  
  
endmodule
```

Chapter 7 Sequential Circuit

- Register
- Counter
- Ultra96 Training Kit Exercises 4

Ultra96 Training Kit Exercises 4

- ❖ EX4-1 Chapter 7에서 Flip-Flop 또는 Register 예제 중 하나를 선택하여 그 모듈과 Testbench를 작성하고 시뮬레이션을 통해 검증하시오.
- ❖ EX4-2 Chapter 7에서 Counter 예제 중 하나를 선택하여 그 모듈과 Testbench를 작성하고 시뮬레이션을 통해 검증하시오.
- ❖ EX4-3 Chapter 4에서는 Vivado에서 제공하는 Counter IP를 Instantiation하여 Counter의 값이 LED에 출력되도록 구현하였다. 이 Counter IP를 사용하지 않고 Verilog로 Counter를 구현하여 LED에 출력되도록 구현하시오.

Chapter 8 Hardware Debugging

- **Timing Constraints**
- **Static Timing Analysis**
- **How to Debug Hardware Directly**
- **Debugging using ILA**
- **Ultra96 Training Kit Exercises 5**

Simulation

- ❖ Simulation은 Functional Simulation과 Timing Simulation으로 구분된다. (※ Chapter 6에서 실행한 Simulation은 Functional Simulation이다.)
- ❖ Functional Simulation은 실제 회로의 전기신호가 전달되는 과정에서 발생하는 Delay를 고려하지 않은 Simulation이다. (※ 기존 Simulation 결과를 보면 출력된 신호가 다른 소자의 입력단에 도착하는 시간이 0초로 해석되어지는 것을 알 수 있다.)
- ❖ Timing Simulation은 Delay 값들을 적용하여 실행하는 Simulation이다.
- ❖ 실제 회로상에서 발생하는 Timing 문제를 해결하기 위해 Timing Analysis가 필요하다.
- ❖ Timing Analysis는 Timing Simulation을 할 수도 있지만 디자인이 복잡해지면 Simulation 툴을 사용하는 것보다 Vivado 툴에서 제공하는 Timing Report와 같은 툴을 통해 Timing Analysis를 하는 것이 효과적이다.

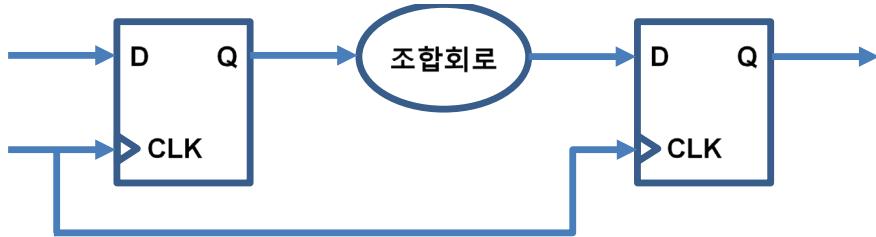
Synthesis & Implementation

- ❖ Vivado에서 Xilinx Device를 Programming하기 위해 Bitstream을 생성하려면 Synthesis와 Implementation 과정을 거쳐야 한다.
- ❖ Synthesis는 RTL로 설계된 회로를 Xilinx Device 안에 있는 Resource들로 재구현한 회로를 만들어 주는 작업이다.
- ❖ Implementation은 Xilinx Device 안에 있는 Resource들로 재구현한 회로를 Xilinx Device 안에 있는 많은 Resource들 중 어떤 Resource를 사용할 지 특정하고 어떤 Route Resource를 사용하여 연결하지를 결정하는 작업이다.
- ❖ Xilinx Device 안에 있는 Resource들 중 어떤 Resource를 사용할 지 특정하는 것을 Place한다고 하고 어떤 Route Resource를 사용하여 연결할지 결정하는 것을 Route라고 부른다. (※ Implementation 과정을 Place & Route(P&R)라고 부르기도 한다.)

Delay

- ❖ 실제 회로에는 전기 신호가 전달될 때 시간이 지연되는 현상이 발생한다.
- ❖ 특정 Resource에 입력된 후 출력될 때까지 시간이 지연되는 것을 Logic Delay라고 부르고 Wire를 통해 다른 Resource로 전기신호가 전달될 때까지 시간이 지연되는 것을 Route Delay라고 부른다.
- ❖ Xilinx Device에서 Logic Delay는 Xilinx가 실험적으로 측정된 값이 사용되고 Route Delay는 Route된 Wire 길이에 의해서 결정된다.
- ❖ 일반적으로 Wire로 전기가 전달되는 속도는 빛의 속도와 비슷하다. (※ 전기 전달 속도는 빛의 속도로 전달되는 것으로 간주한다.)
- ❖ 빛의 속도는 진공에서 $299,792,458 \text{ m/s}$ 이다. (※ 대략 $300,000,000 \text{ m/s}$ 로 계산한다.)
- ❖ 10ns 에 $0.3\text{m}(30\text{cm})$ 를 움직이는 속도로 계산된다. (※ 회로상에 전기신호가 안정된 후 30cm 도선 거리에 전달되려면 10 ns 가 걸린다.)

Timing Violation



- ❖ RTL Circuit에서 Clock Edge에 Register에 입력된 신호가 출력되어 Combinatorial Circuit를 지나서 다음 Register 입력단에 신호가 도착할 때까지 Delay가 발생한다.
- ❖ 100MHz Clock이 연결되어 있다면 Total Delay값이 10ns 보다 작아서 Register의 입력단에 먼저 도착해 있어야 새로 업데이트 된 값이 다음 Edge에 출력된다.
- ❖ Total Delay값이 10ns 보다 커서 다른 Edge에 출력되지 못하게 되는 것을 Timing Violation이 발생했다고 한다.
- ❖ Timing Violation이 발생하면 Vivado에서 제공하는 Report 기능을 이용하여 Violation이 발생한 경로를 확인할 수 있다.

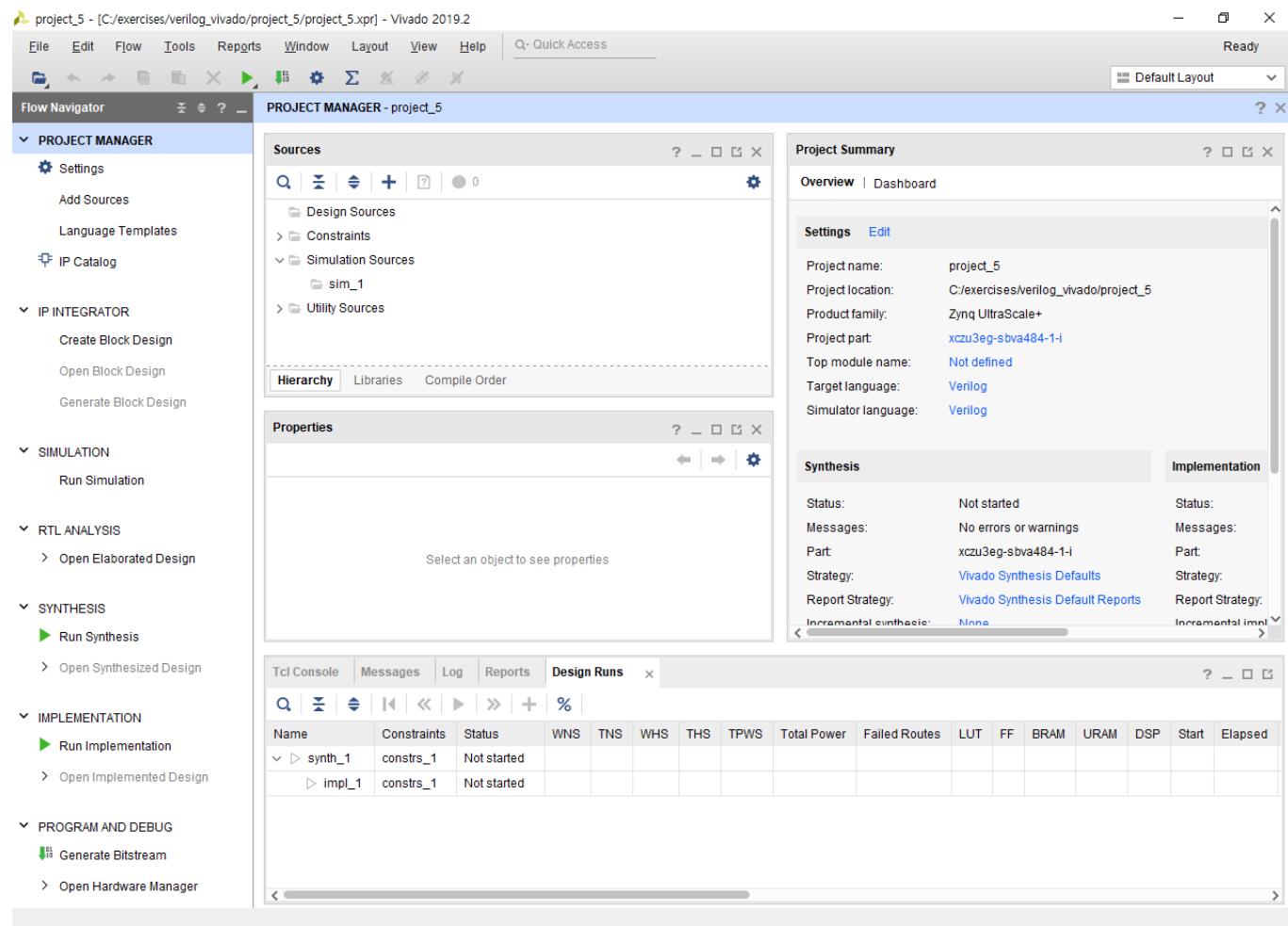
Timing Constraints

- ❖ Vivado Tool은 설계한 RTL Circuit의 Clock에 몇 MHz의 Clock이 연결되어 있는지 알 수 없다.
- ❖ RTL Circuit의 Clock에 몇 MHz의 Clock이 연결되어 있는지 알려주는 것이 Timing Constraints이다.
- ❖ Timing Constraints도 Pin Constraints가 입력된 .xdc 파일에 입력된다.
- ❖ Timing Constraints를 입력하는 것은 Clock의 속도를 제한하여 Timing Violation이 발생하는지 확인하기 이유도 있고 Vivado Tool이 P&R을 할 때 Timing Violation이 발생하지 않도록 노력하게 하기 위함 이기도 하다.

Chapter 8 Hardware Debugging

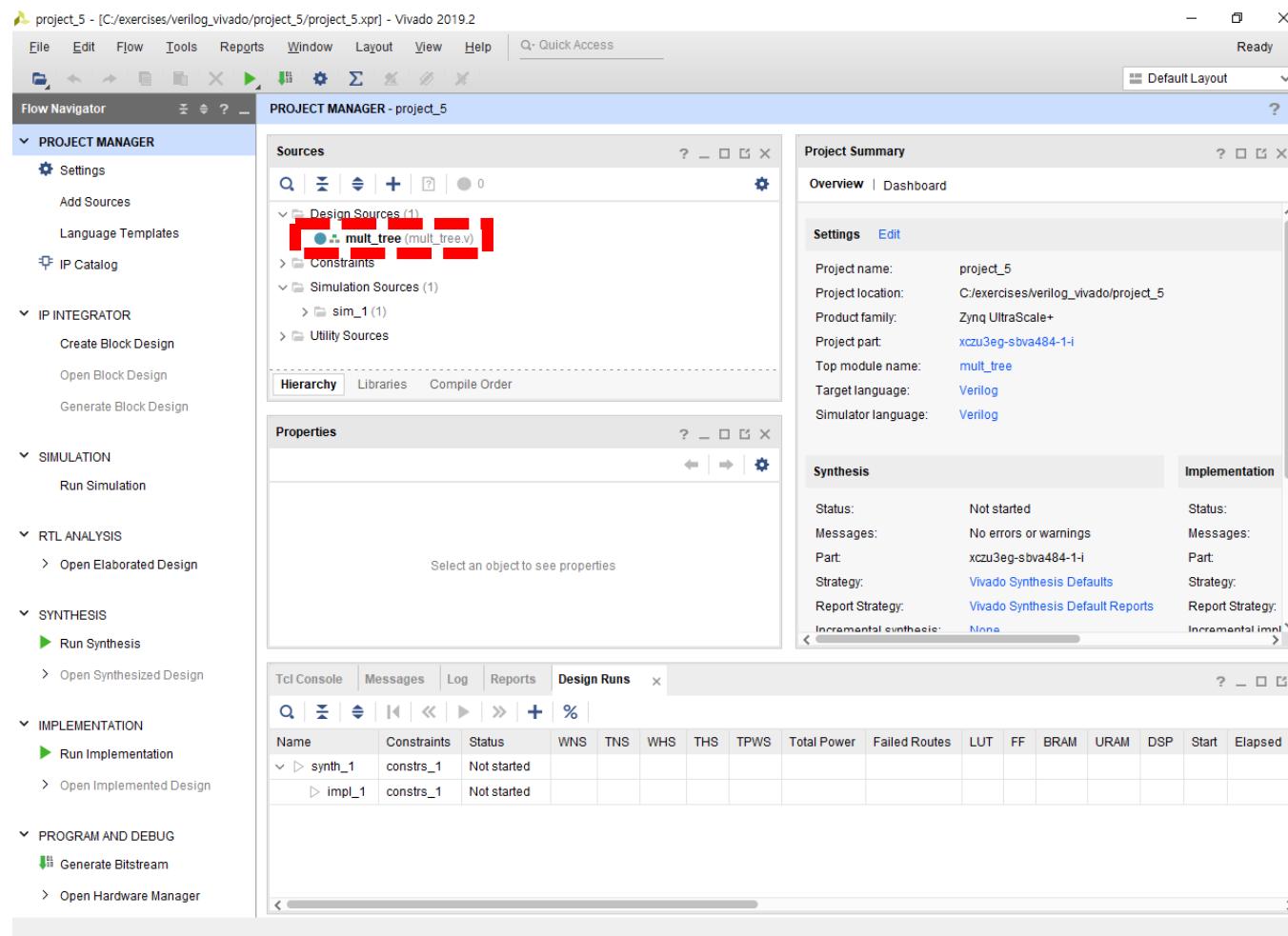
- Timing Constraints
- Static Timing Analysis
- How to Debug Hardware Directly
- Debugging using ILA
- Ultra96 Training Kit Exercises 5

Step 1 Creating Vivado Project



❖ Chapter 1의 Vivado 프로젝트 만들기를 참조하여 project_5 프로젝트를 만든다.

Step 2 Add Design Source in Vivado Project



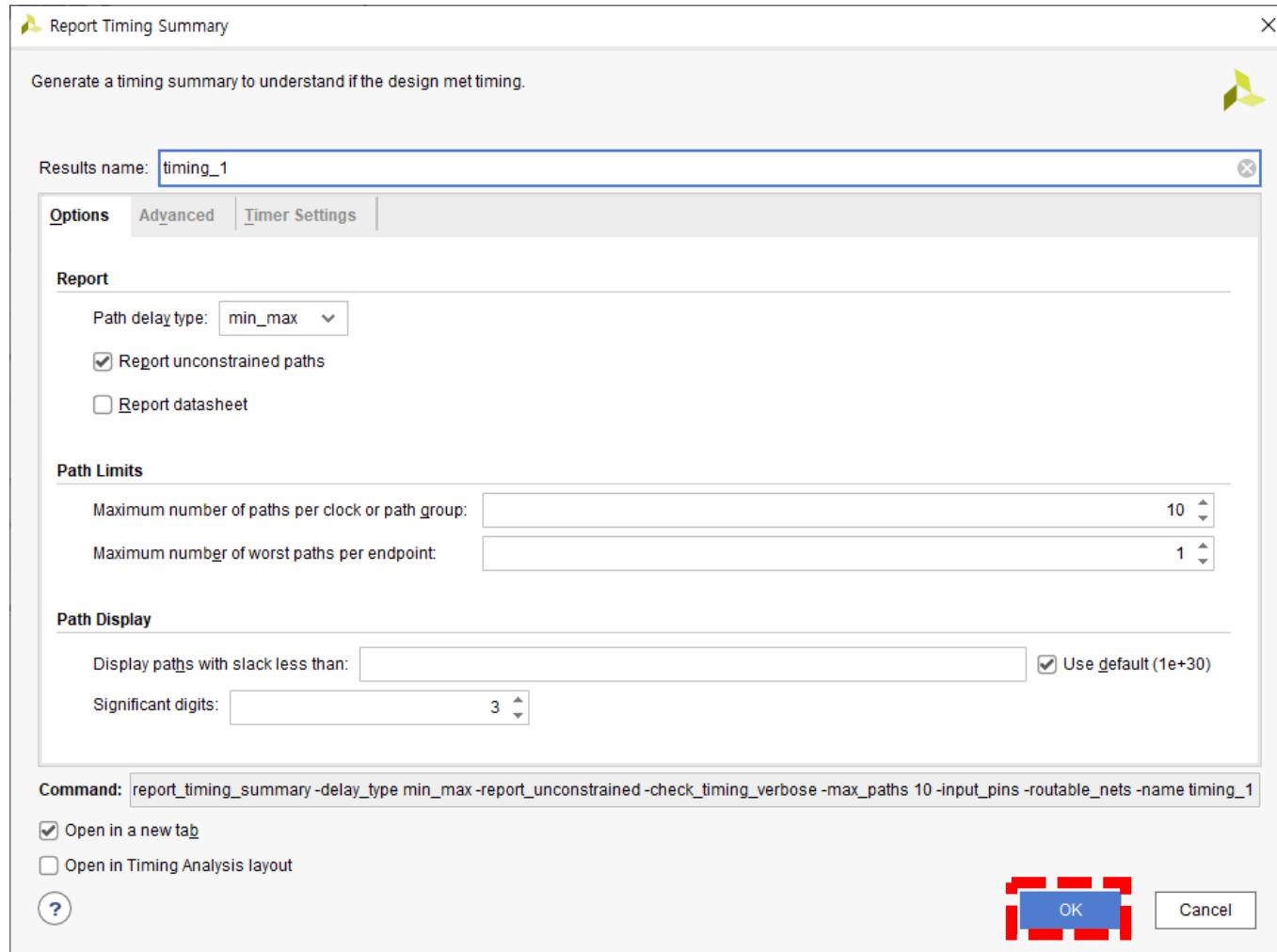
❖ Chapter 2 Vivado Design
Flow의 Step 2를 참고하여
mult_tree.v 파일을 프로젝트
에 추가한다.

Step 3 Write Design Source Code

- ❖ Source Windows 안의 mult_tree.v파일을 더블 클릭하여 Editor Window에 연 후 mult_tree 모듈 소스 코드를 입력한다.

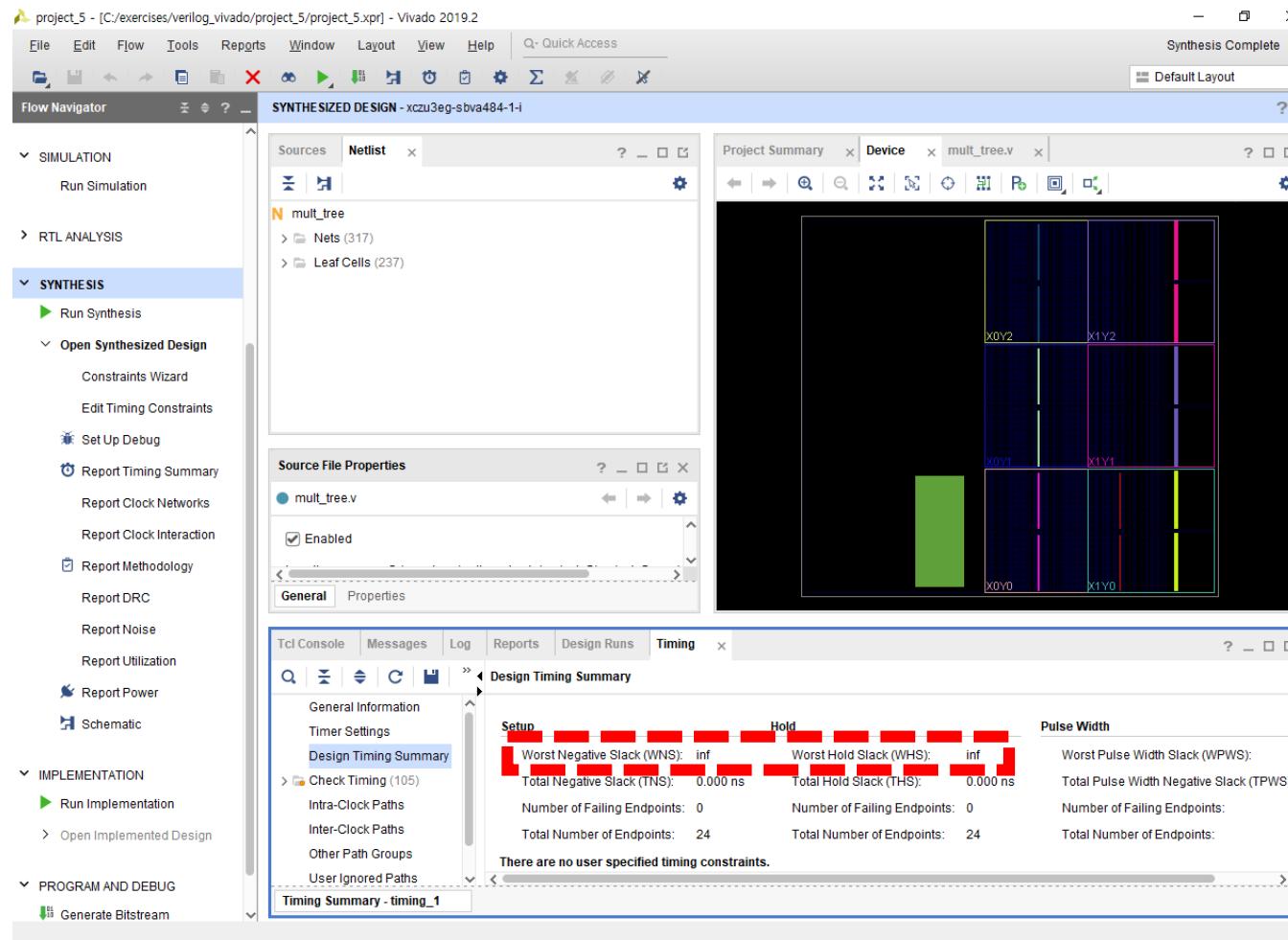
```
module mult_tree(  
    input clk,  
    input[3:0] i_a, i_b, i_c, i_d,  
    output[7:0] o_result  
>);  
  
reg[3:0] a_buf, b_buf, c_buf, d_buf;  
reg[23:0] result_buf;  
  
always @(posedge clk)  
begin  
    a_buf <= i_a;  
    b_buf <= i_b;  
    c_buf <= i_c;  
    d_buf <= i_d;  
    result_buf <= (  
        ((a_buf * b_buf) * b_buf) *  
        ((c_buf * d_buf) * d_buf)  
    );  
end  
  
assign o_result = result_buf[23:16];  
endmodule
```

Step 4 Add Timing Constraints 1



- ❖ Flow Navigator ⇒ Synthesis
⇒ Run Synthesis를 클릭하여 디자인을 합성한다.
- ❖ Flow Navigator ⇒ Synthesis
⇒ Open Synthesized Design
⇒ Report Timing Summary를 클릭하면 Report Timing Summary Window가 열린다.
- ❖ OK버튼을 클릭하여 Timing Summary를 생성한다.

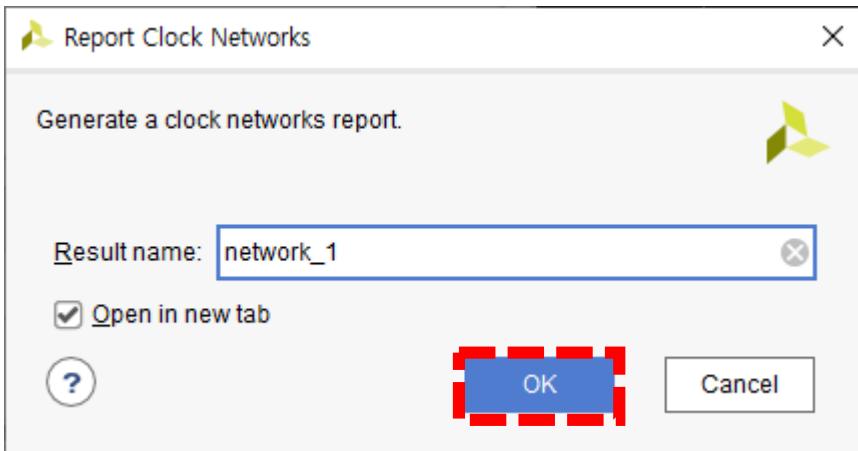
Step 4 Add Timing Constraints 2



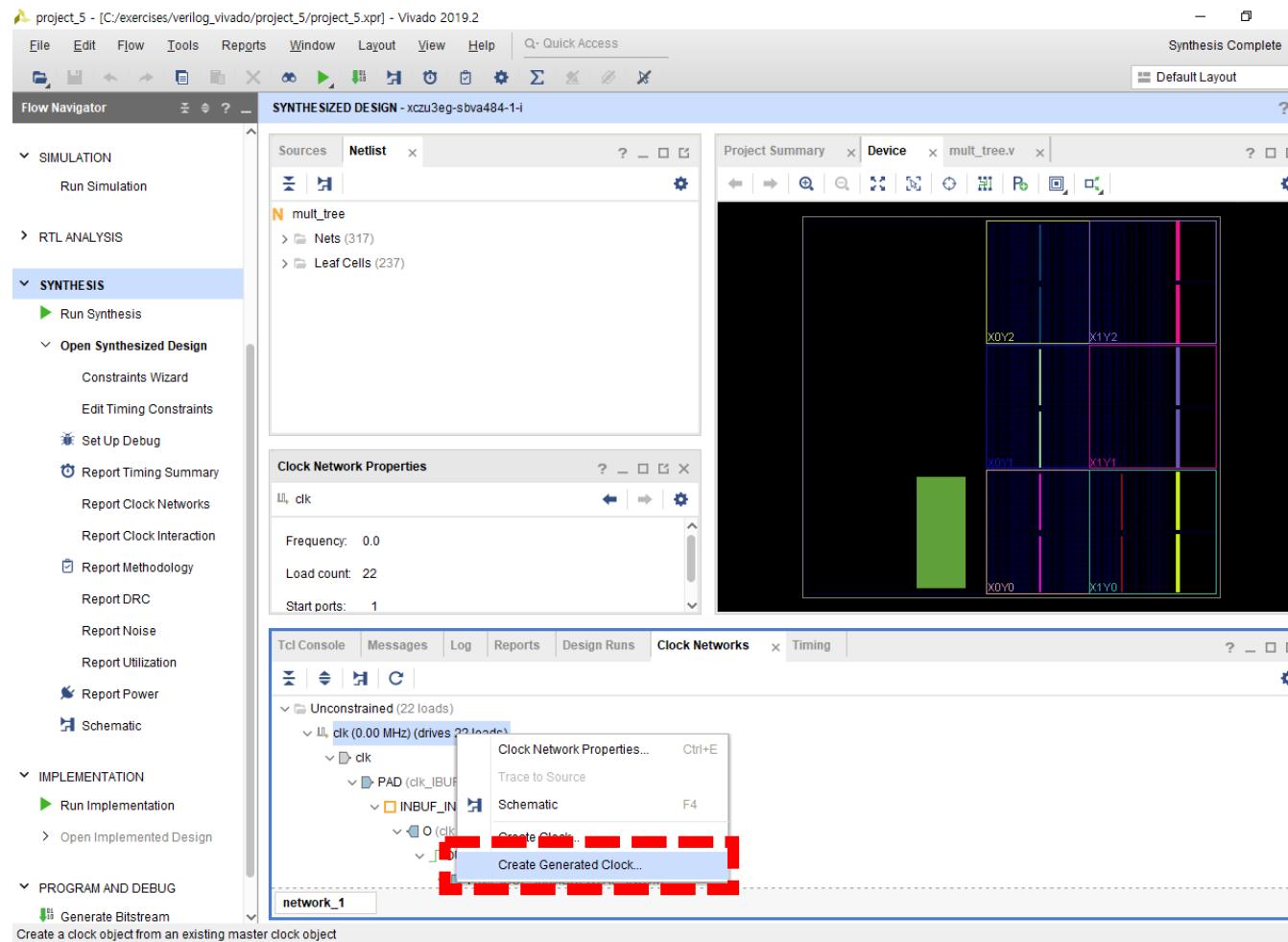
- ❖ Vivado 하단 Window의 Timing 탭에서 Design Timing Summary 메뉴를 선택하면 Setup과 Hold의 Worst Negative Slack이 inf로 뜨는 것을 확인할 수 있다.
- ❖ Inf로 나오는 이유는 Timing Constraints가 적용되지 않았기 때문이다.

Step 4 Add Timing Constraints 3

❖ Flow Navigator ⇒ Synthesis
⇒ Open Synthesized Design
⇒ Report Clock Network를
클릭하여 Report Clock
Network Window가 열리면
OK버튼을 클릭한다.

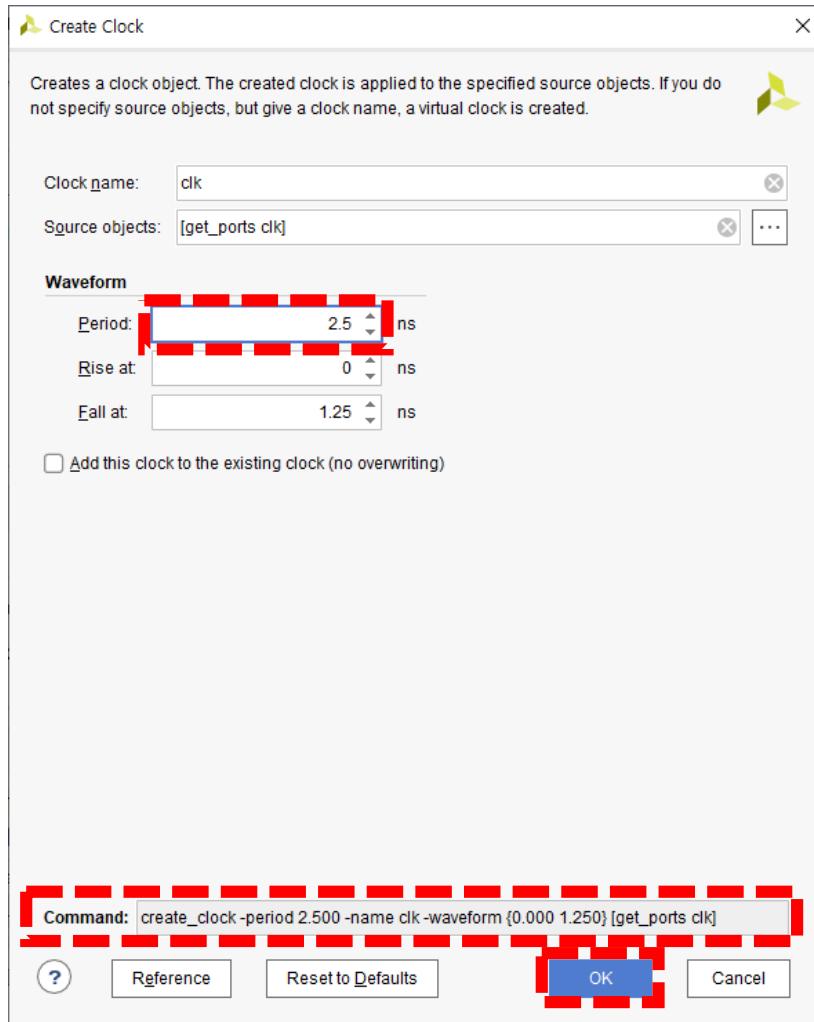


Step 4 Add Timing Constraints 4



❖ Vivado 하단 Window안에
Clock Networks탭에서
Unconstrained 항목 바로 아래
에 뜨는 clk위에서 마우스 오
른쪽 버튼을 클릭하고 Create
Clock메뉴를 클릭한다.

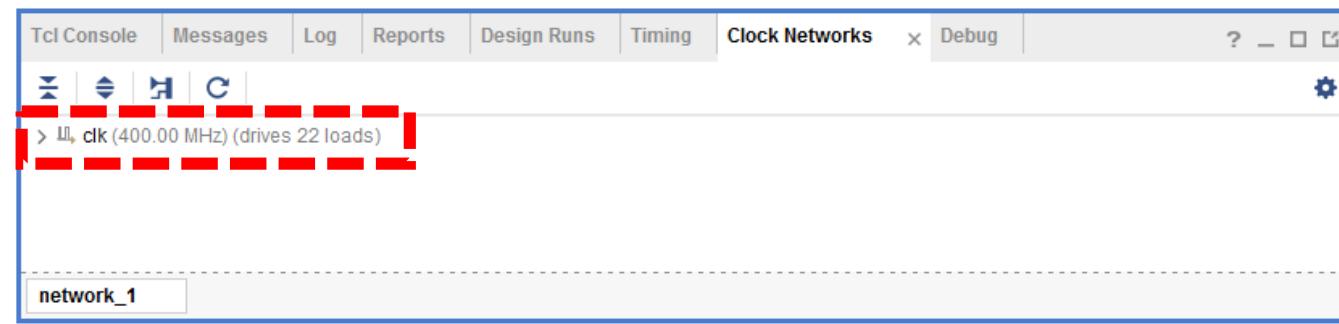
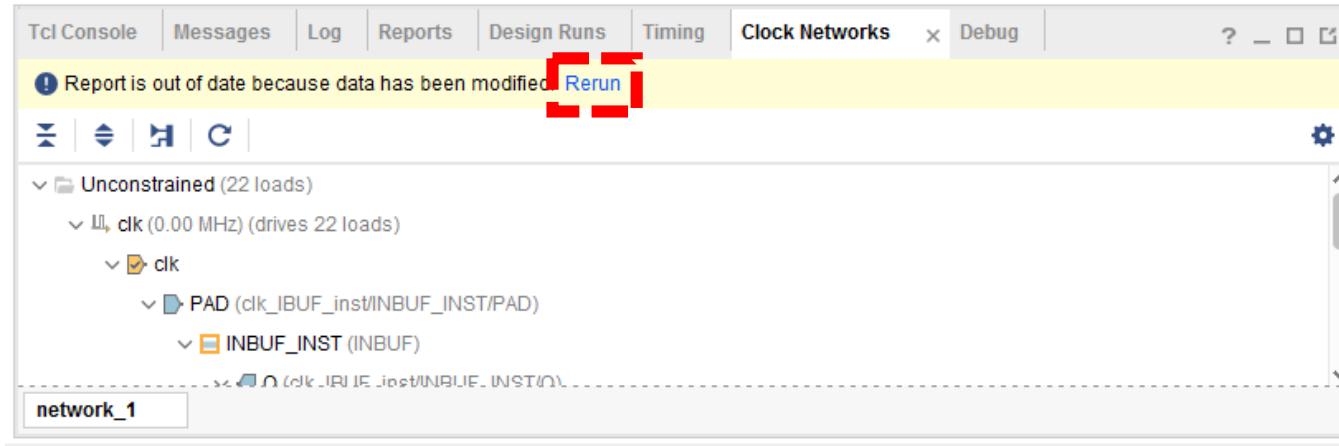
Step 4 Add Timing Constraints 5



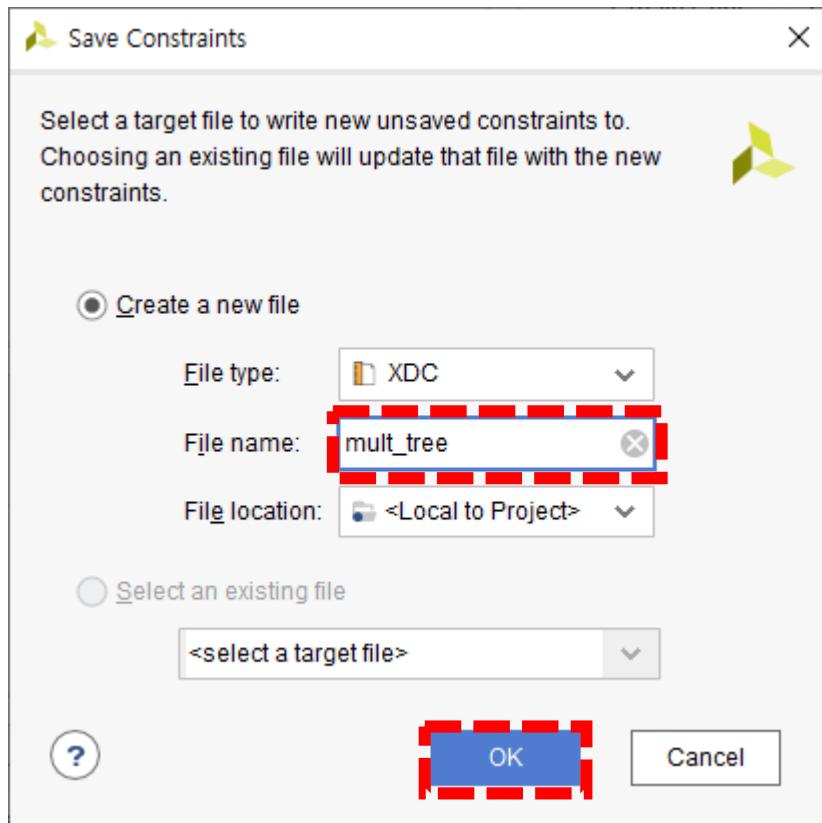
- ❖ Create Clock Windows가 나오면 Period에 2.5를 입력하여 400MHz타겟으로 Clock Constraints를 주고 OK버튼을 클릭한다.
- ❖ Command 품에 뜨는 내용은 GUI를 통해 설정한 Constraints에 대한 스크립트로 xdc파일에 저장될 코드이다.

Step 4 Add Timing Constraints 6

❖ Clock Networks 탭에 생긴 Rerun 메뉴를 클릭하면 clk에 대한 Timing Constraints가 추가되는 것을 확인할 수 있다.

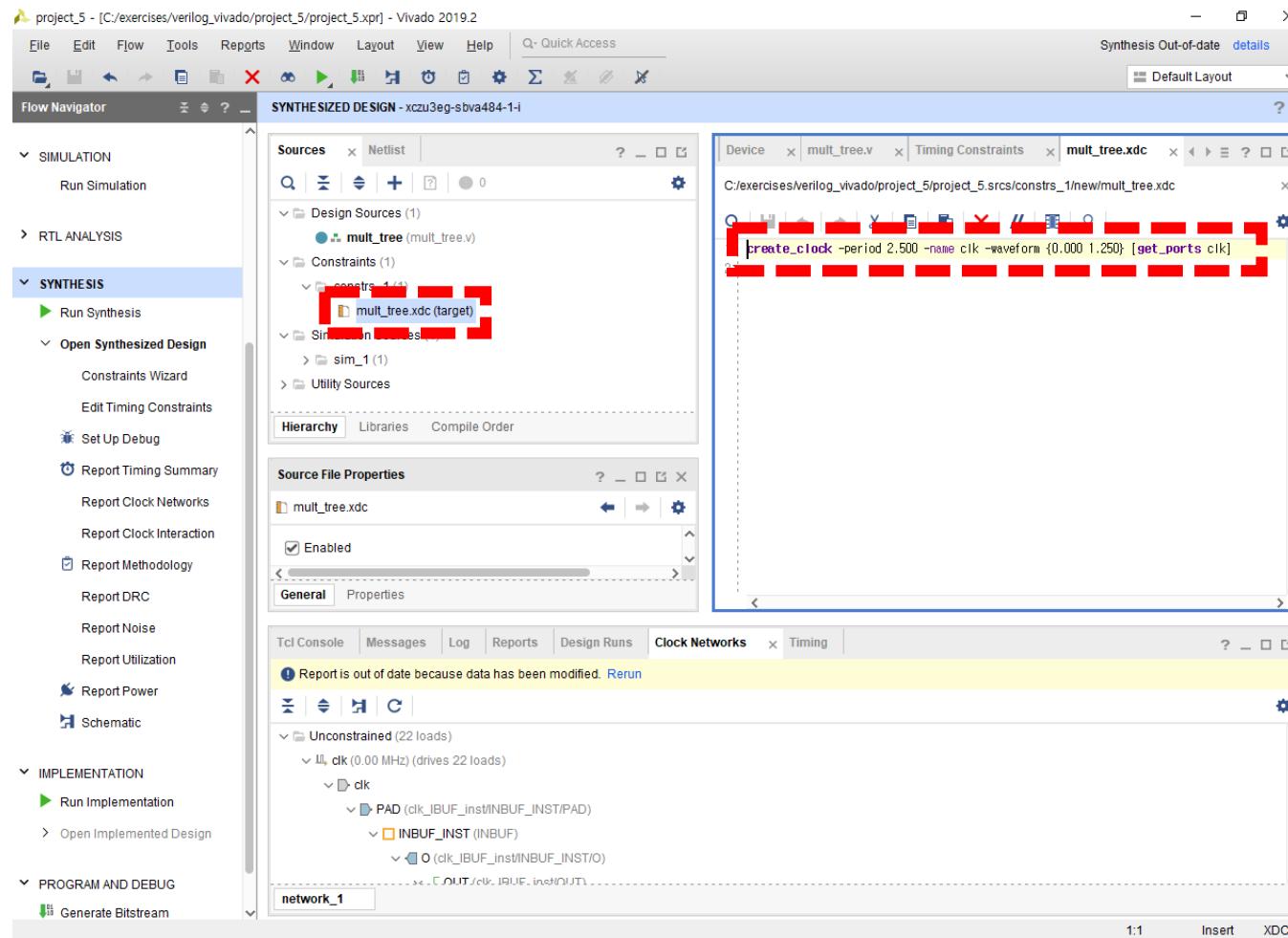


Step 4 Add Timing Constraints 7



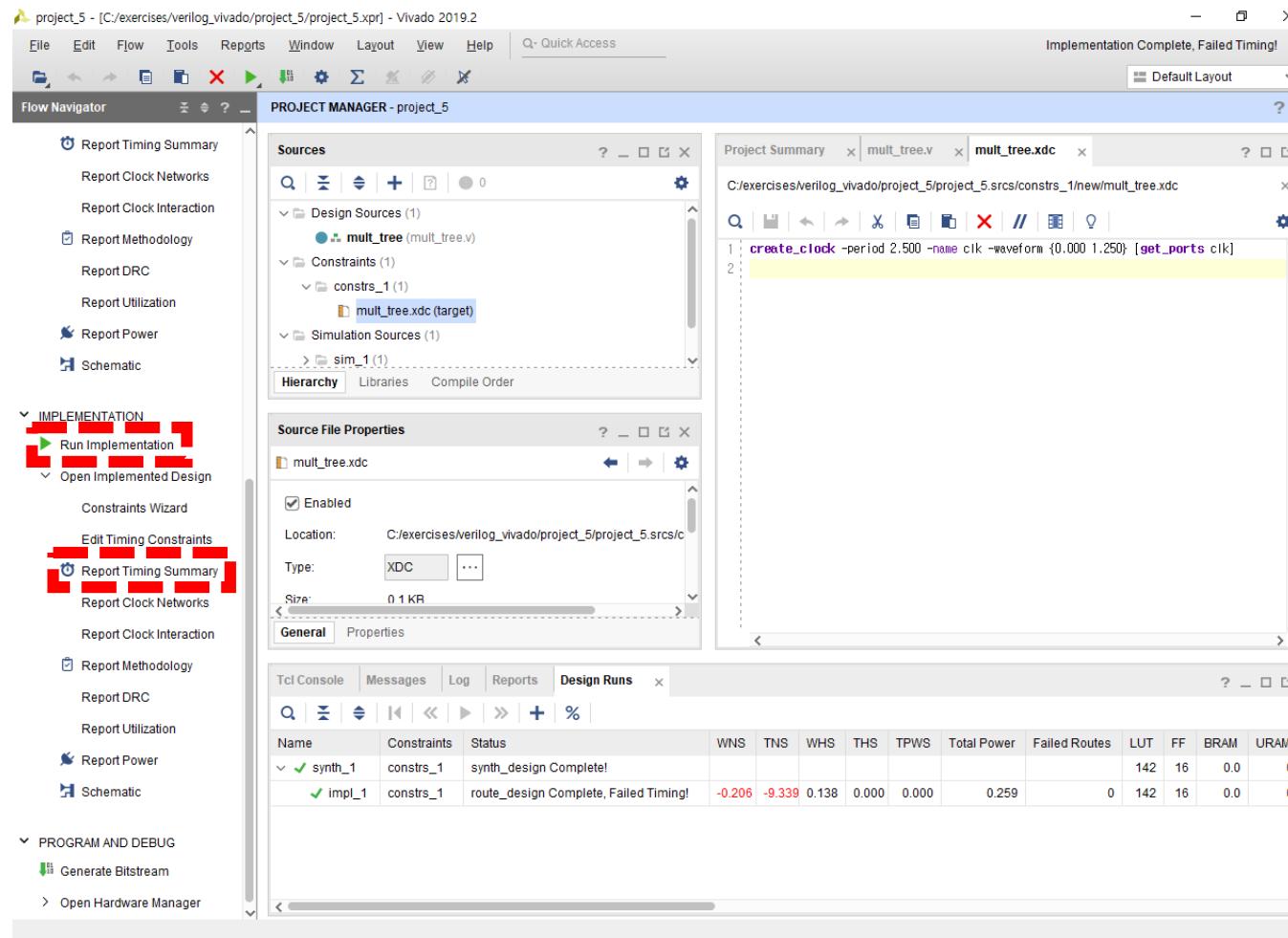
- ❖ File ⇒ Constraints ⇒ Save 메뉴를 클릭한다.
- ❖ Save Constraints Window가 나오면 File name에 mult_tree 를 입력하고 OK버튼을 클릭한다.

Step 4 Add Timing Constraints 8



❖ Sources Windows 안에
Constraints 아래 mult_tree.xdc
파일을 더블 클릭하면 Timing
Constraints 내용이 추가된 것
을 확인할 수 있다.

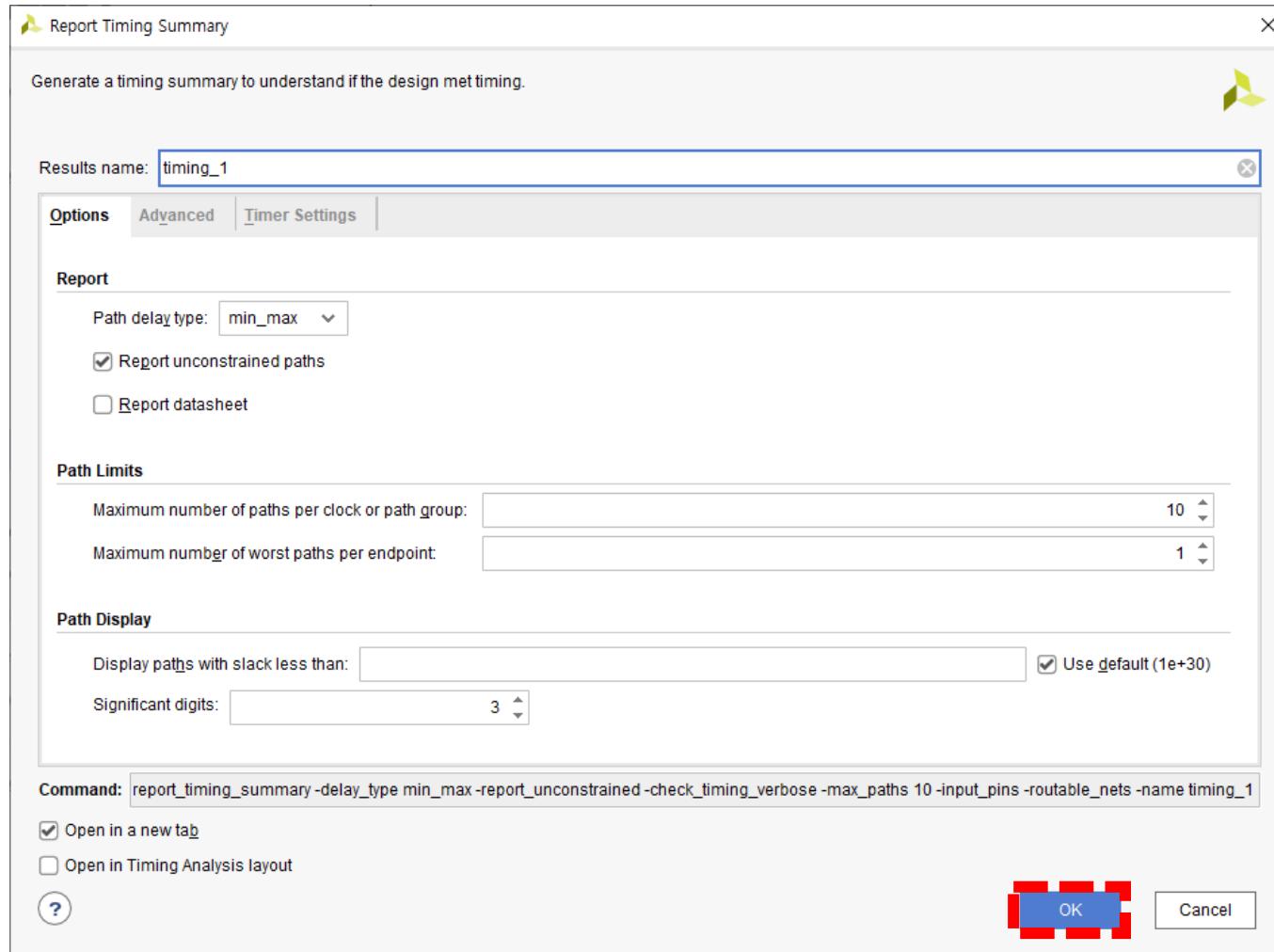
Step 5 Run Implementation



- ❖ Flow Navigator ⇒
Implementation ⇒ Run
Implementation을 클릭한다.

- ❖ Flow Navigator ⇒
Implementation ⇒ Open
Implemented Design ⇒
Report Timing Summary를 클릭한다.

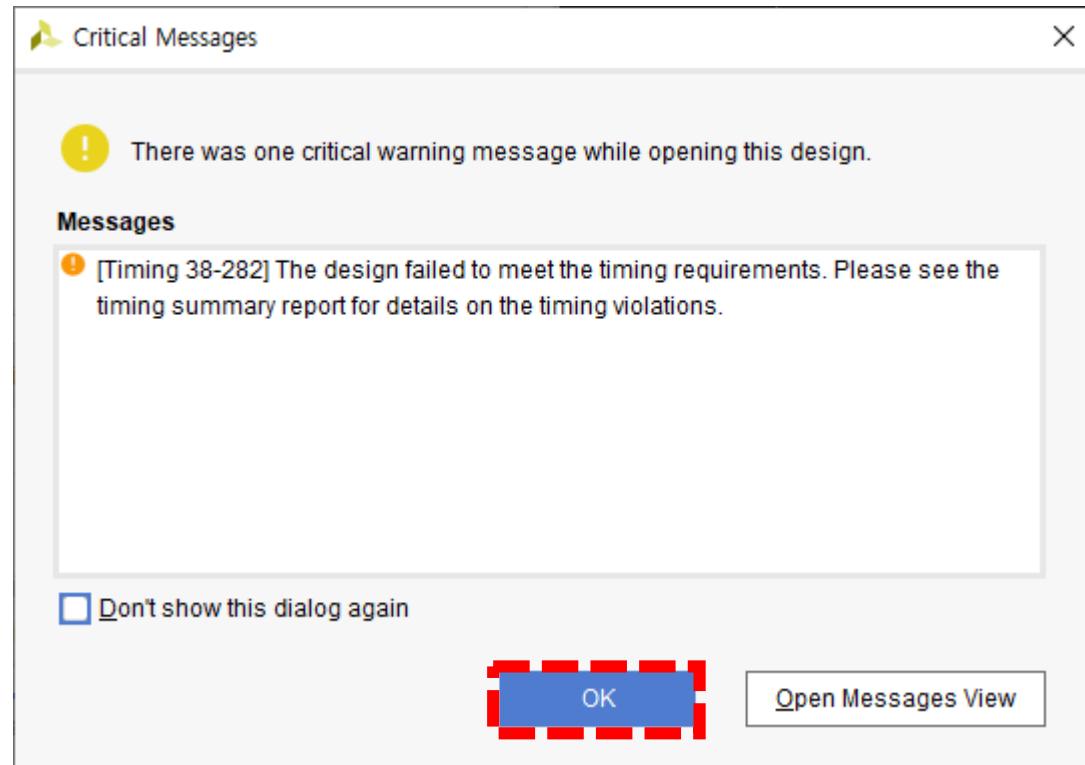
Step 6 Analysis Timing 1



❖ Report Timing Summary

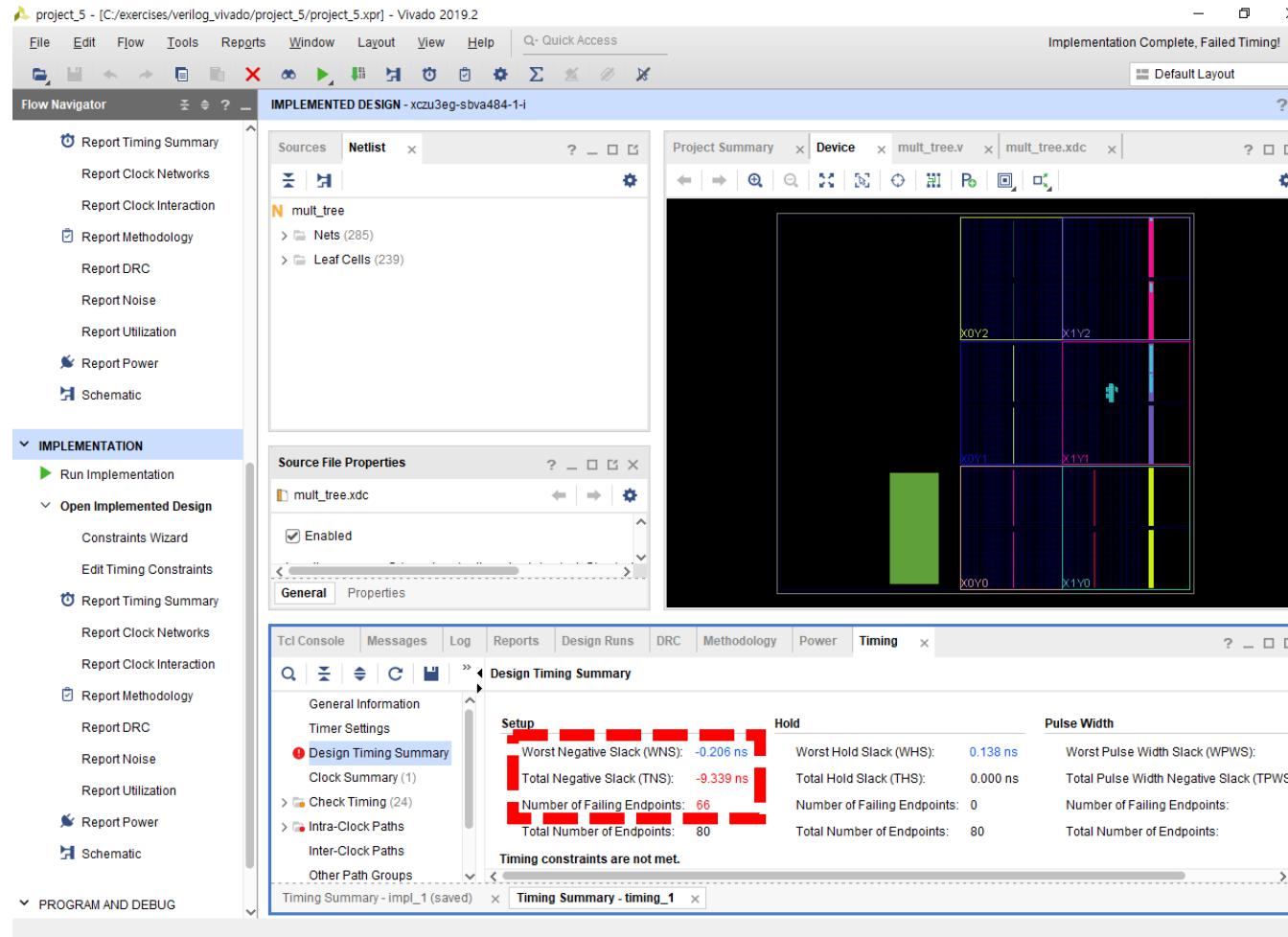
Window가 나오면 OK 버튼을 클릭한다.

Step 6 Analysis Timing 2



- ❖ 그림과 같은 Critical Messages Window가 나오면 Timing Violation이 발생했다는 의미이다.
- ❖ OK 버튼을 클릭한다.

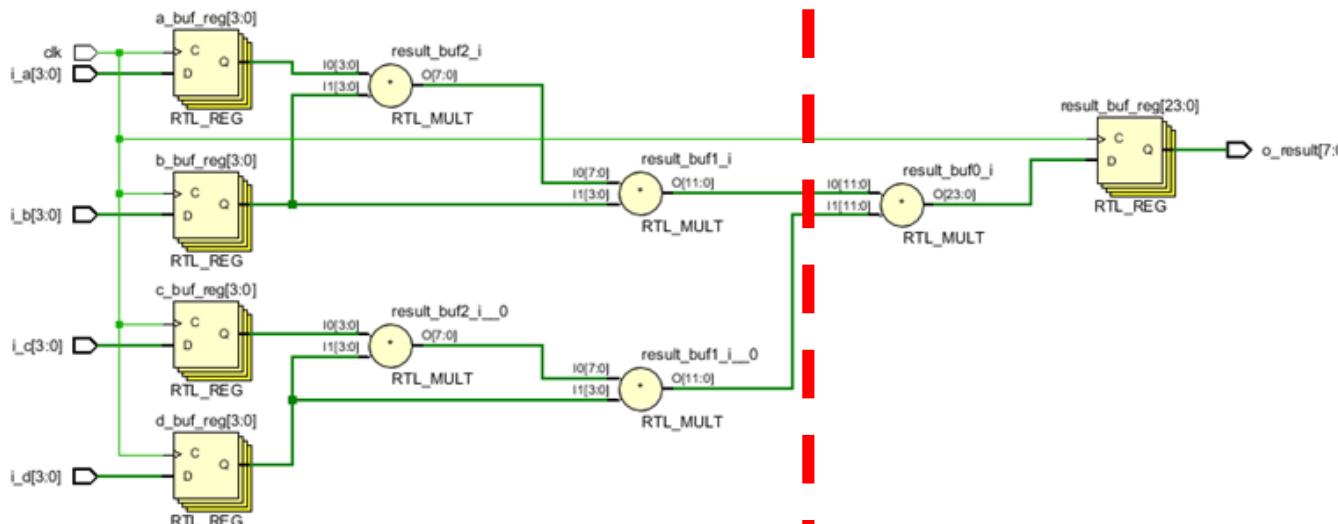
Step 6 Analysis Timing 3



❖ Vivado 하단 Windows안에 Timing탭의 Design Timing Summary 메뉴를 클릭하면 Setup 항목에서 Total Negative Slack이 -9.339 ns이고 Number of Failing Endpoints가 66라고 뜬 것을 확인할 수 있다.

❖ 66개의 Path에서 Timing Violation이 발생했고 신호가 늦은 시간들의 총합이 9.339 ns라는 의미이다.

Step 7 Modify Design Source 1



- ❖ Flow Navigator ⇒ RTL Analysis ⇒ Schematic을 클릭하여 Schematic을 보면 *a_buf*, *b_buf*, *c_buf*, *d_buf* 레지스터 출력은 각각 3개의 곱셈기를 통과하여 *result_buf* 레지스터에 저장되는 것을 확인할 수 있다.
- ❖ 점선 부분에 레지스터를 삽입하면 연산 결과에 영향을 미치지 않으면서 레지스터 사이의 신호 전달 시간을 줄일 수 있다.
- ❖ 중간에 삽입된 레지스터 때문에 입력이 들어가고 결과가 나오는데 걸리는 시간이 1 클럭 늘어나게 된다.

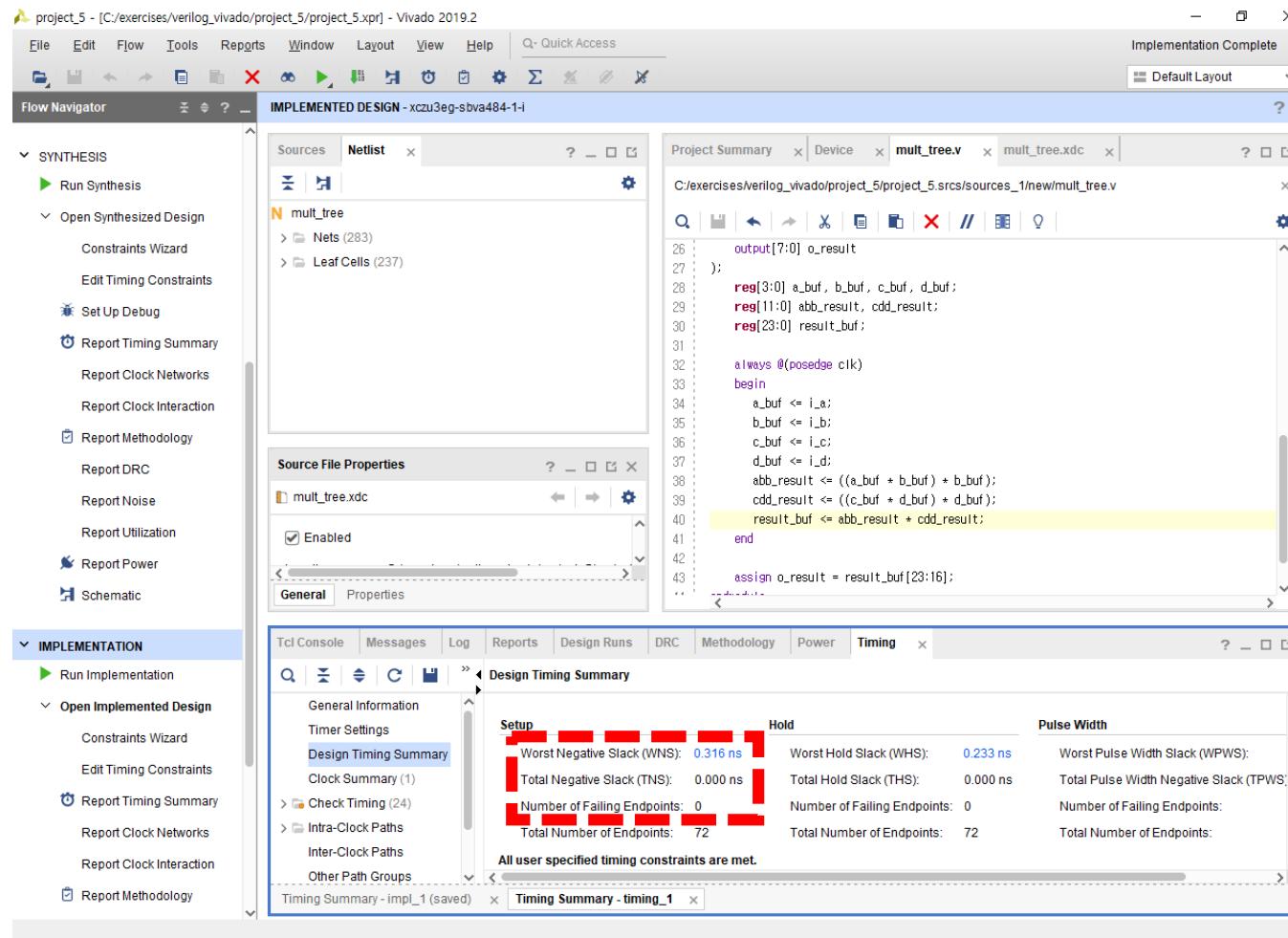
Step 7 Modify Design Source 2

- ❖ Sources Window의 mult_tree.v파일을 더블 클릭하여 Editor Window에 연다.
- ❖ mult_tree 모듈을 우측의 코드와 같이 수정한다.

```
module mult_tree(
    input clk,
    input[3:0] i_a, i_b, i_c, i_d,
    output[7:0] o_result
);
    reg[3:0] a_buf, b_buf, c_buf, d_buf;
    reg[11:0] abb_result, cdd_result;
    reg[23:0] result_buf;

    always @(posedge clk)
    begin
        a_buf <= i_a;
        b_buf <= i_b;
        c_buf <= i_c;
        d_buf <= i_d;
        abb_result <= ((a_buf * b_buf) * b_buf);
        cdd_result <= ((c_buf * d_buf) * d_buf);
        result_buf <= abb_result * cdd_result;
    end
    assign o_result = result_buf[23:16];
endmodule
```

Step 8 Confirm the Timing Summary



- ❖ Flow Navigator ⇒ Implementation
⇒ Run Implementation을 클릭하여, Synthesis와 Implementation을 실행한다.
- ❖ Flow Navigator ⇒ Implementation
⇒ Open Implemented Design ⇒ Report Timing Summary를 클릭한다.
- ❖ Vivado 하단 Windows에 Timing 탭의 Design Timing Summary 메뉴를 클릭하면 Setup Violation이 사라진 것을 확인할 수 있다.

Chapter 8 Hardware Debugging

- Timing Constraints
- Static Timing Analysis
- **How to Debug Hardware Directly**
- Debugging using ILA
- Ultra96 Training Kit Exercises 5

How to Debug Hardware Directly

- ❖ 하드웨어를 디버깅하기 위해 Simulation과 Timing Analysis 방법에 대해서 배웠다.
- ❖ Simulation과 Timing Analysis 방법은 소프트웨어 툴을 사용하여 하드웨어를 디버깅하는 방법이어서 실제 회로에서는 이와 다르게 동작 할 수도 있다. (※ 이런 경우에는 동작하는 회로 상의 신호를 직접적으로 검증하는 방법을 사용한다.)
- ❖ 동작하는 회로 상의 신호를 직접적으로 검증하는 방법에는 ILA(Integrated Logic Analyzer)를 사용하는 방법과 검증하고자 하는 신호를 FPGA I/O로 연결한 후 오실로스코프와 같은 계측장비로 측정하는 방법이 있다.
- ❖ Digilent에서 제작한 Analog Discovery 2는 여러 가지 계측 기능들을 포함하고 있어서 값 비싼 계측장비들을 구매하지 않아도 저렴한 가격에 하드웨어를 디버깅할 수 있다.

Introduction to Analog Discovery 2

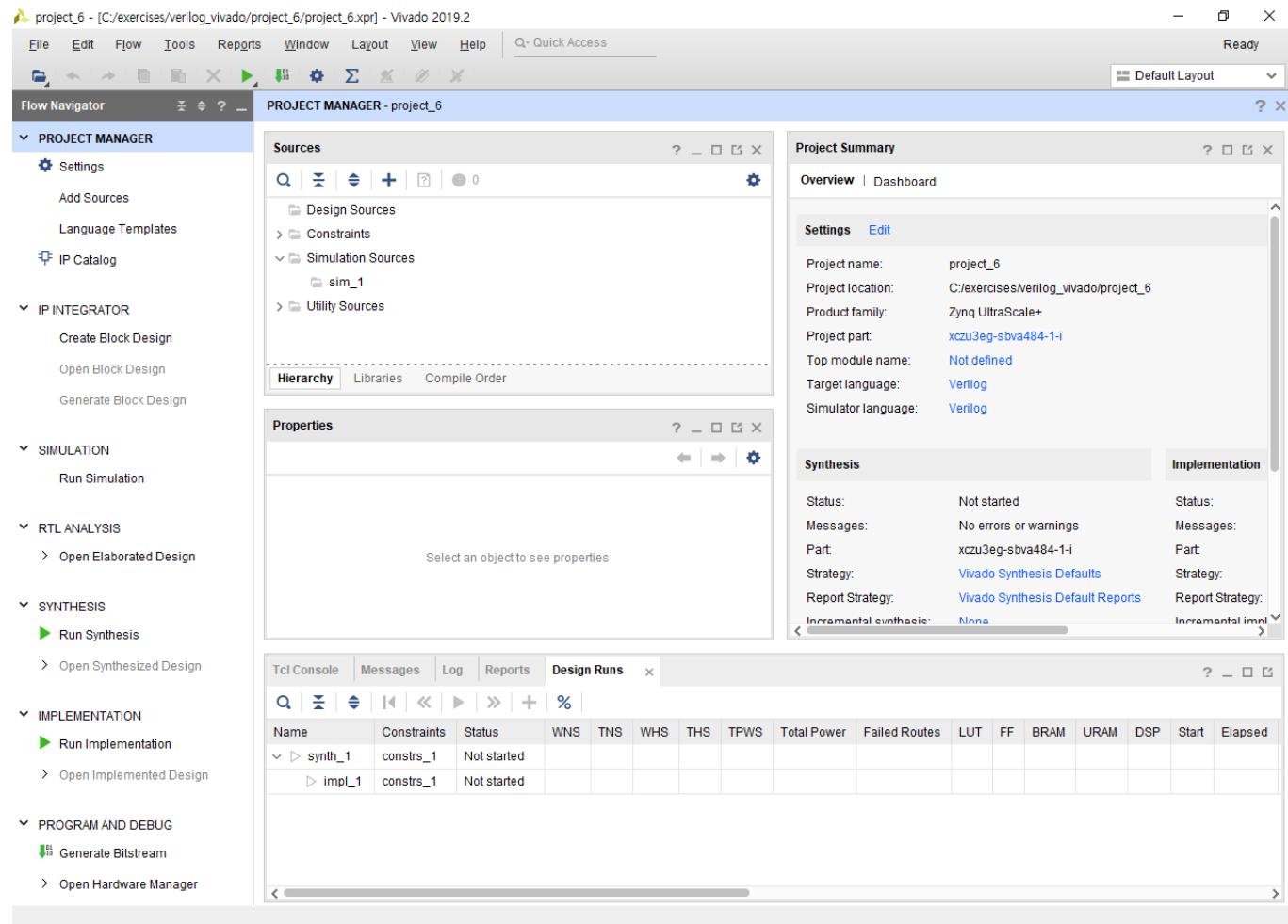


- ❖ Digilent의 휴대용 오실로스코프인 Analog Discovery 2 이다.
- ❖ Analog Discovery 2를 사용하기 위해서는 WaveForms 라는 PC용 무료 소프트웨어를 웹에서 다운로드 받아서 설치하면 사용 가능하다.
- ❖ Analog Discovery 2와 WaveForms는 USB 2.0 인터페이스를 통해 연동된다.
- ❖ https://www.inipro.net/goods/goods_view.php?goodsNo=1000617163에서 구매 가능하다.

Chapter 8 Hardware Debugging

- Timing Constraints
- Static Timing Analysis
- How to Debug Hardware Directly
- Debugging using ILA
- Ultra96 Training Kit Exercises 5

Step 1 Creating Vivado Project



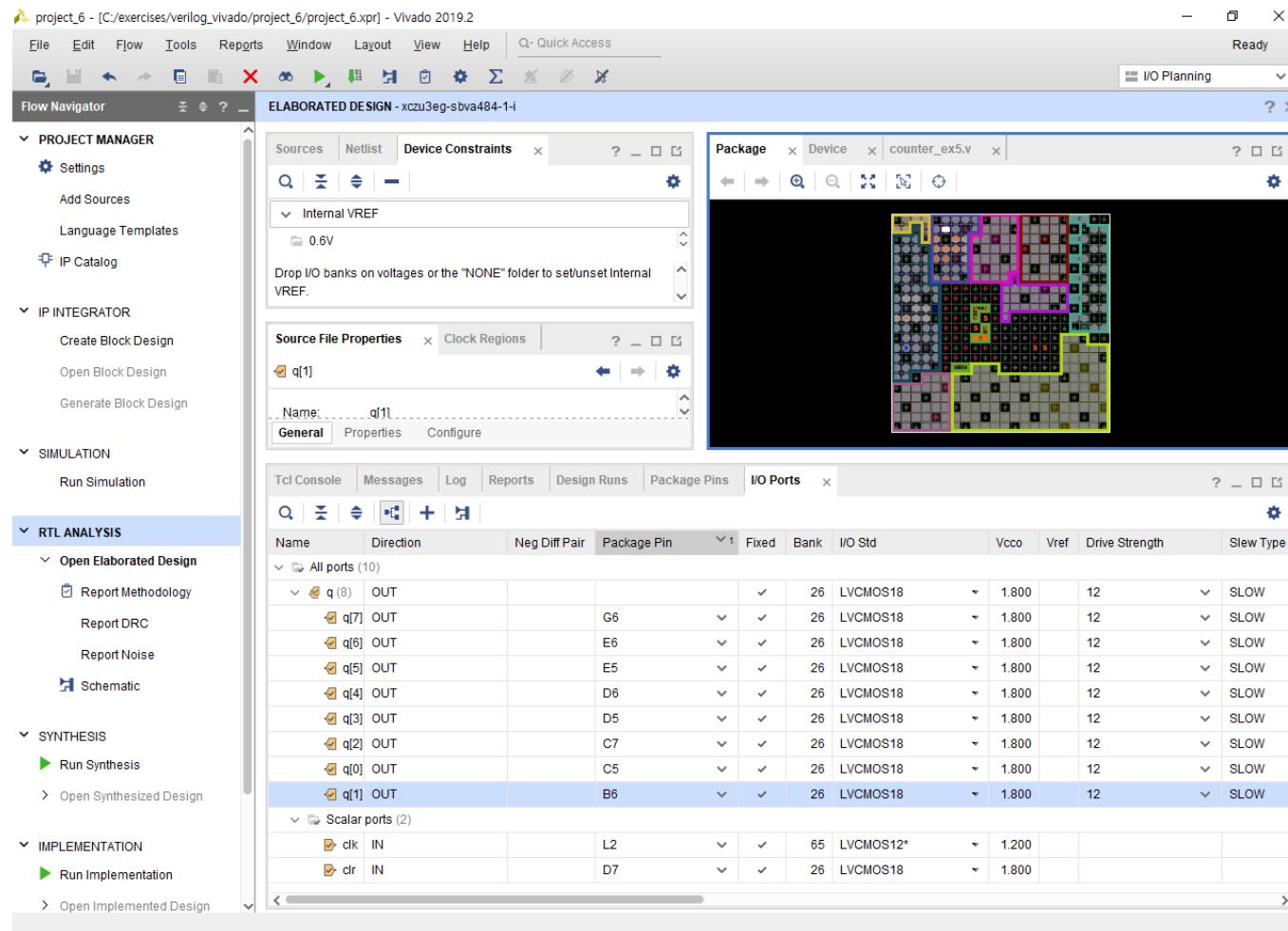
❖ Chapter 1의 Vivado 프로젝트 만들기를 참조하여 project_6 프로젝트를 만든다.

Step 2 Add Design Source in Vivado Project

- ❖ Chapter 2 Vivado Design Flow의 Step 2를 참고하여 counter_ex5.v 파일을 프로젝트에 추가한다.
- ❖ Sources Window의 counter_ex5.v 파일을 더블 클릭하여 Editor Window에 연 후 오른쪽에 있는 코드를 입력한다.

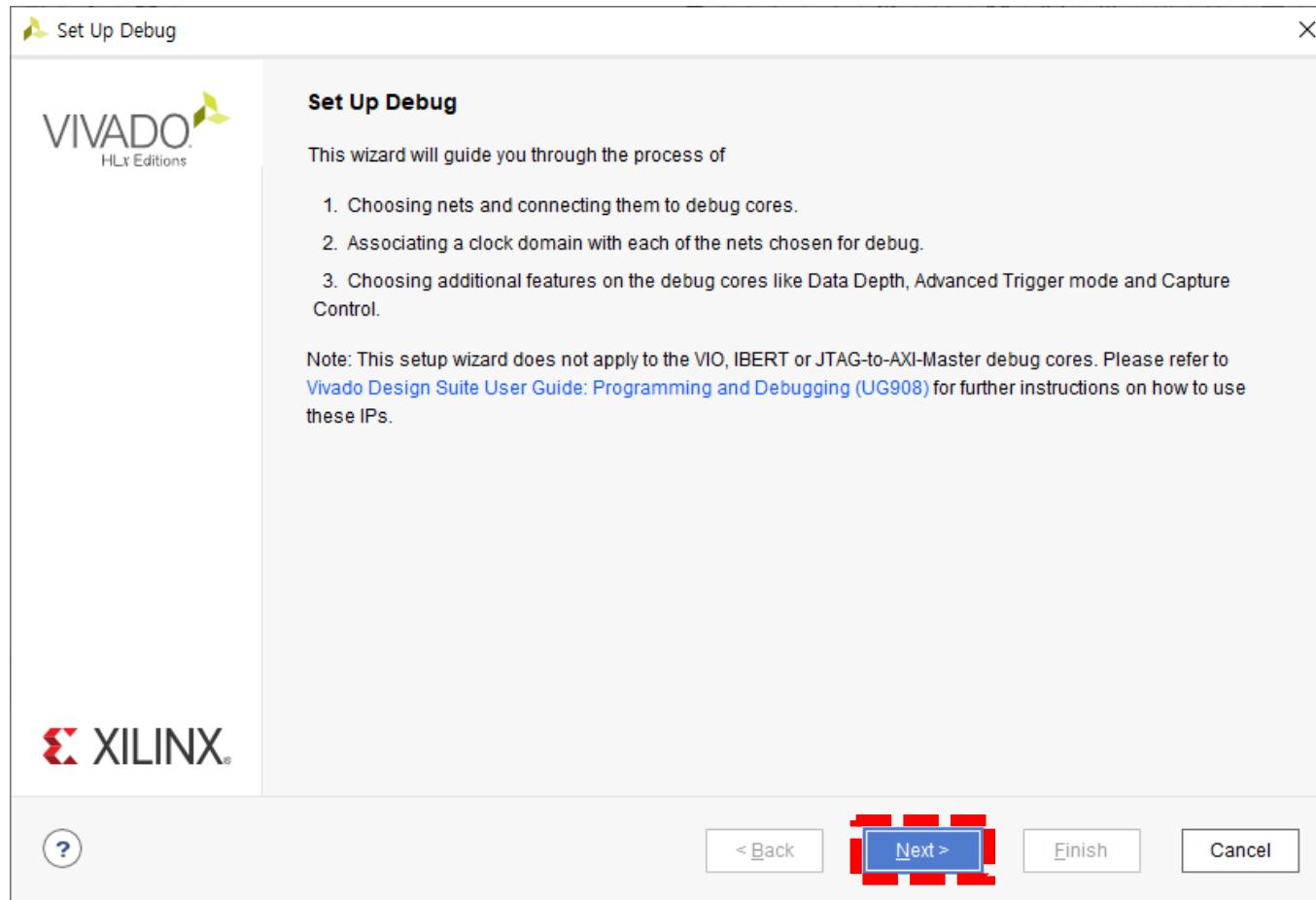
```
module counter_ex5(clk,clr,q);
    input clk,clr;
    output [7:0] q;
    reg [31:0] cnt;
    /****** DEBUG_MARK attribute *****/
    (* mark_debug = "true" *) wire clr;
    (* mark_debug = "true" *) wire [7:0] q;
    /****** */
    always @(posedge clr or posedge clk)
    begin
        if(clr)
            cnt <= 32'h00000000;
        else
            cnt <= cnt + 1;
    end
    assign q = cnt[31:24];
endmodule
```

Step 3 Add Constraints



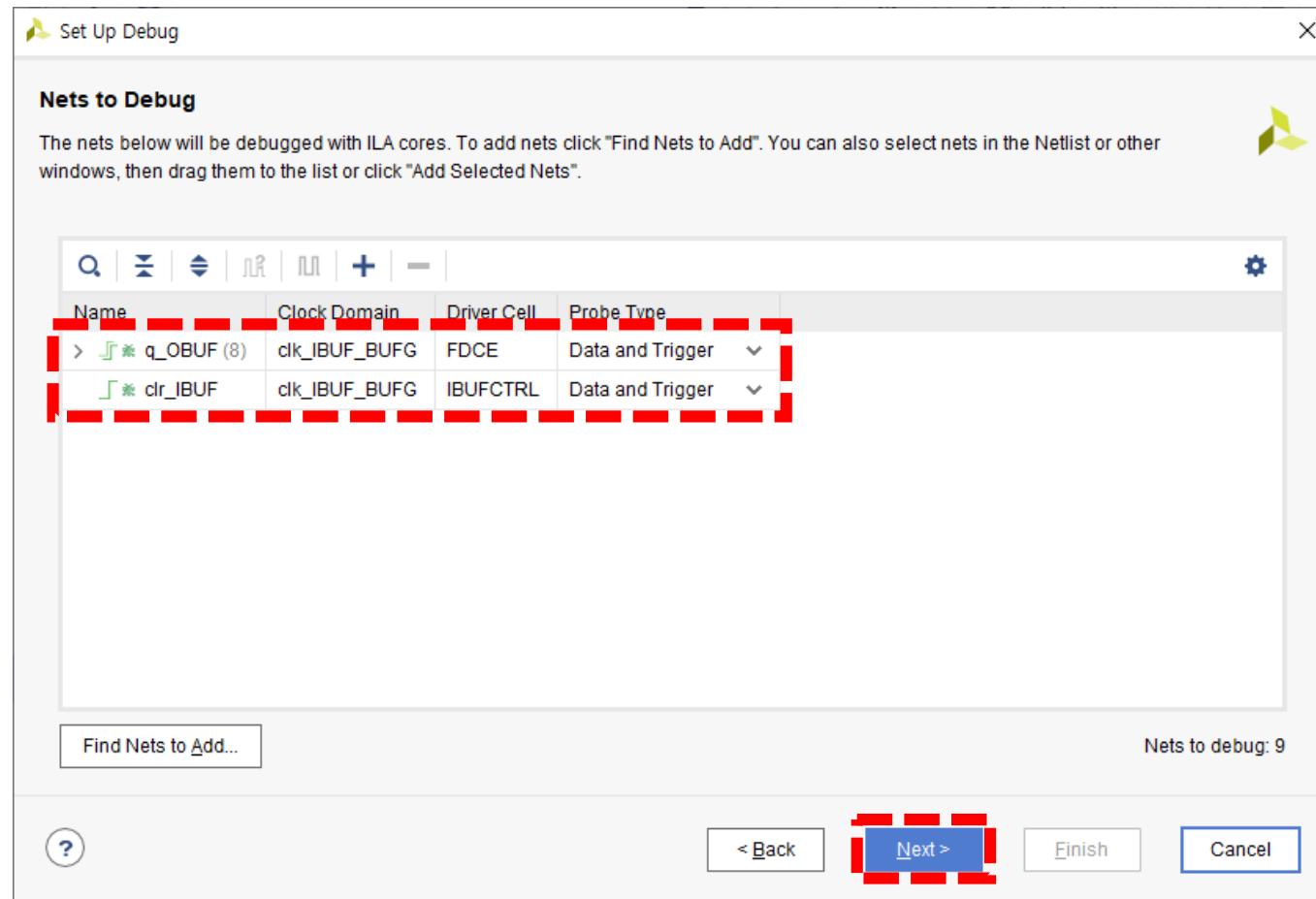
- ❖ Chapter 2 Vivado Design Flow의 Step 5를 참고하여 counter_ex5.xdc 파일을 프로젝트에 추가한다.
- ❖ Chapter 2 Vivado Design Flow의 Step 5를 참고하여 clk 포트는 L2핀에 연결하고 clr 포트는 PMOD_A커넥터 중 하나에 연결하고 q는 PMOD_B커넥터에 연결한다.
- ❖ clk의 I/O Std는 LVCMS12로 Constraints하고 clr, led포트는 LVCMS18로 Constraints해준다.

Step 4 Add ILA on Netlist 1



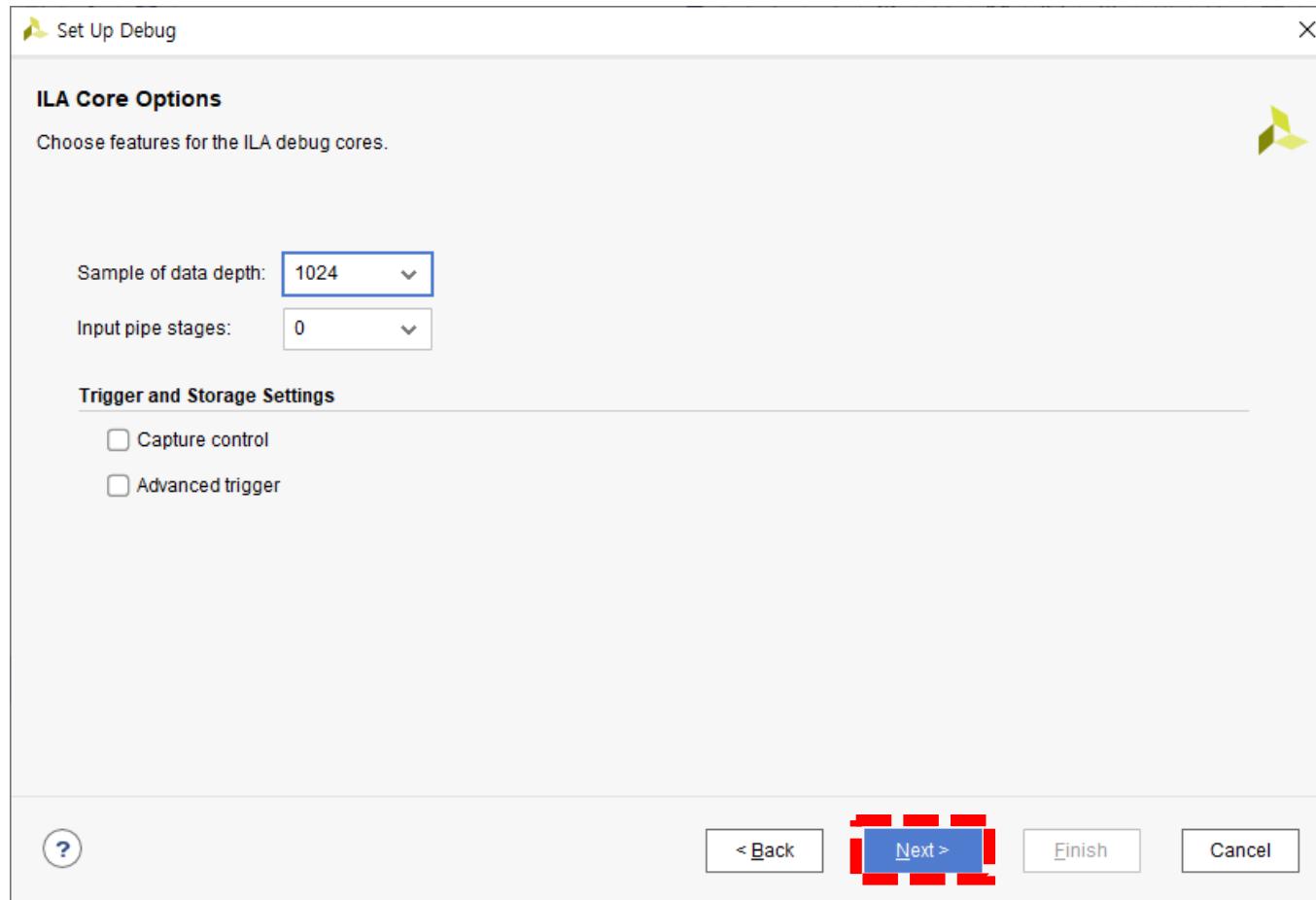
- ❖ Flow Navigator ⇒ Synthesis
⇒ Run Synthesis를 클릭하여 디자인을 합성한다.
- ❖ Flow Navigator ⇒ Synthesis
⇒ Open Synthesized Design
⇒ Set Up Debug를 클릭한다.
- ❖ Set Up Debug Window가 나오면 Next 버튼을 클릭한다.

Step 4 Add ILA on Netlist 2



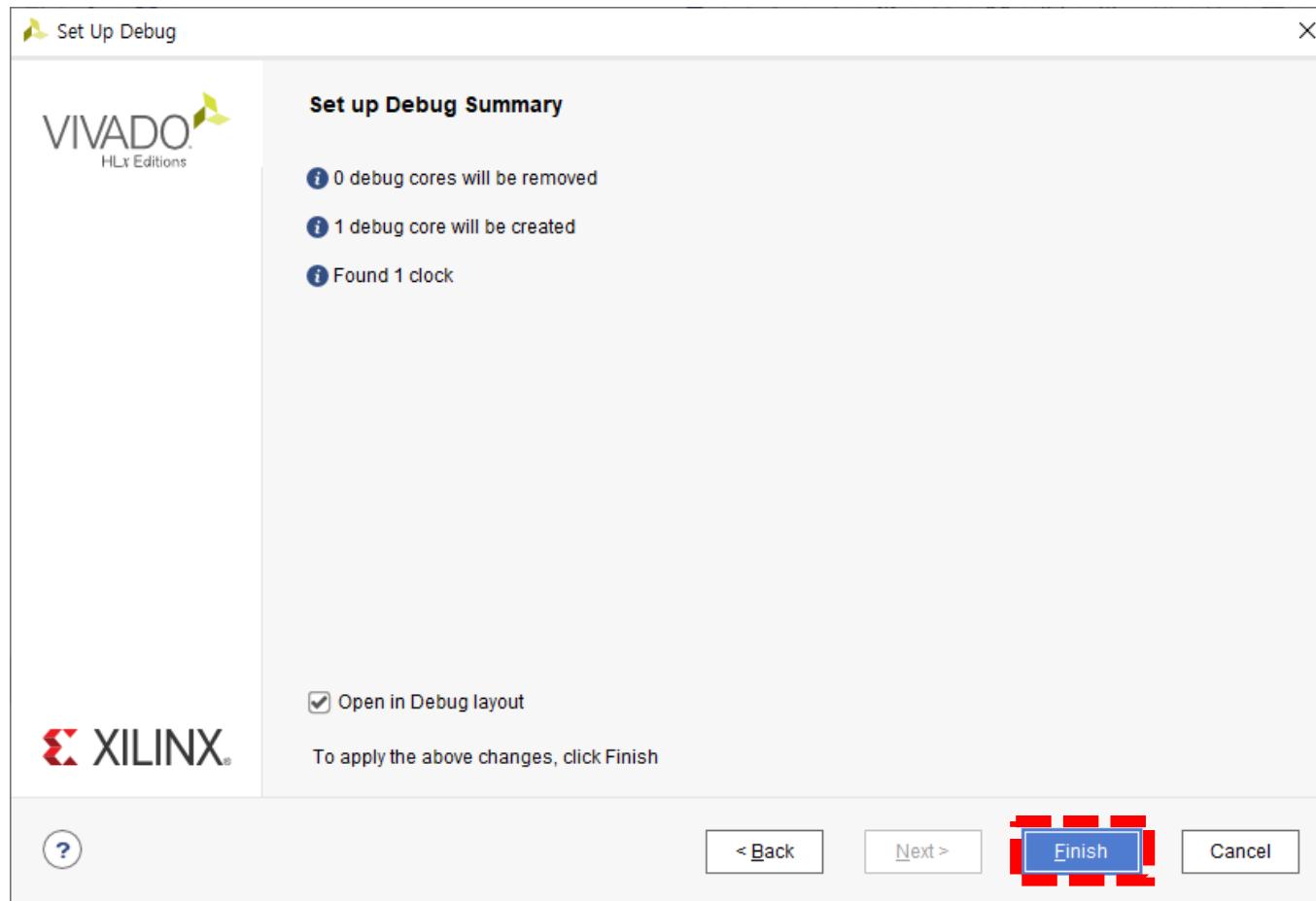
❖ 디버거로 계측 할 신호를 추가하는 창이 나오면 clr, q 신호를 추가한 후 Next 버튼을 클릭한다.

Step 4 Add ILA on Netlist 3



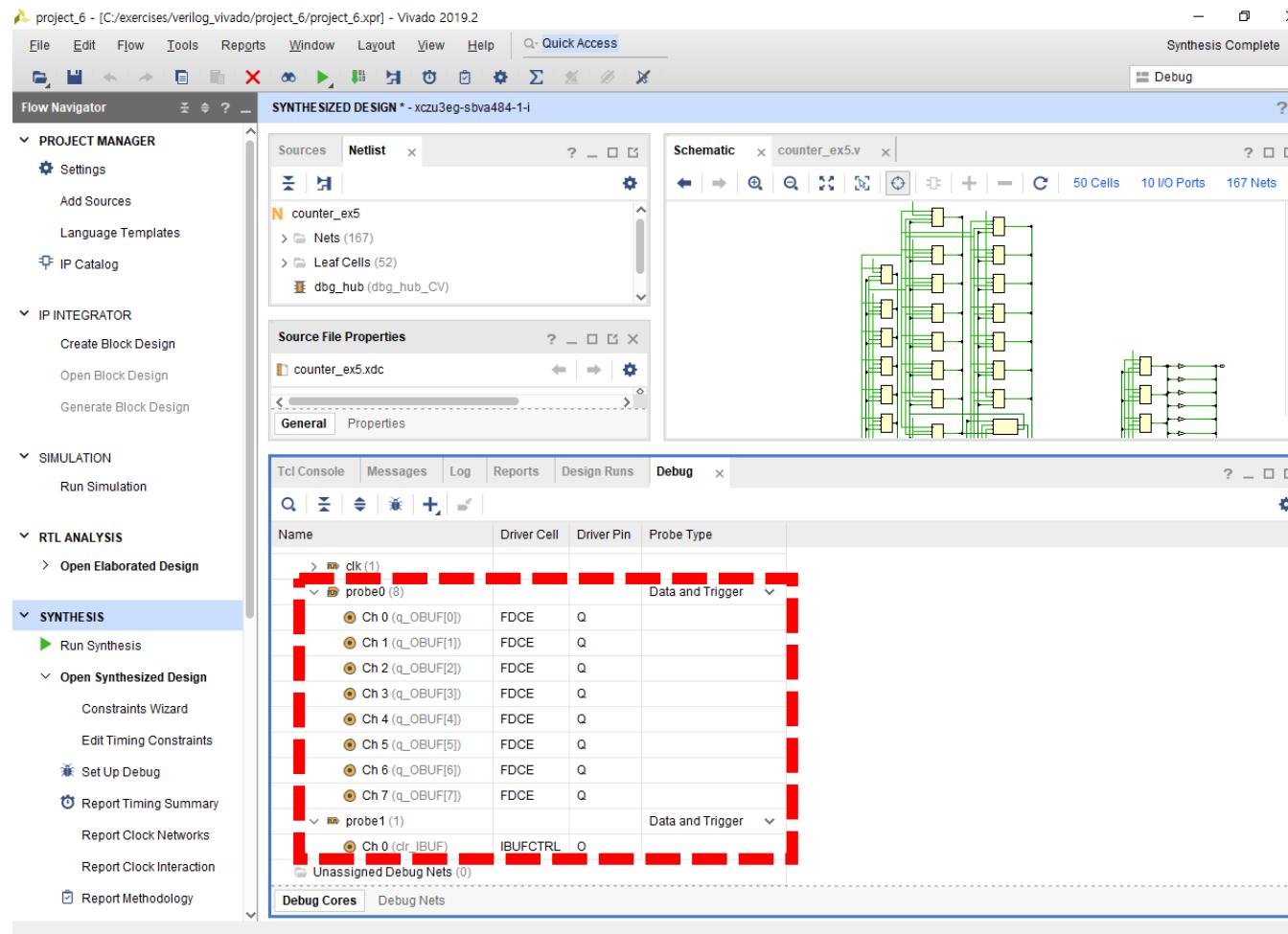
❖ Sample of data depth, Input pipe stages 등을 설정하는 창이 나오면 디폴트 상태 그대로 두고 Next 버튼을 클릭한다.

Step 4 Add ILA on Netlist 4



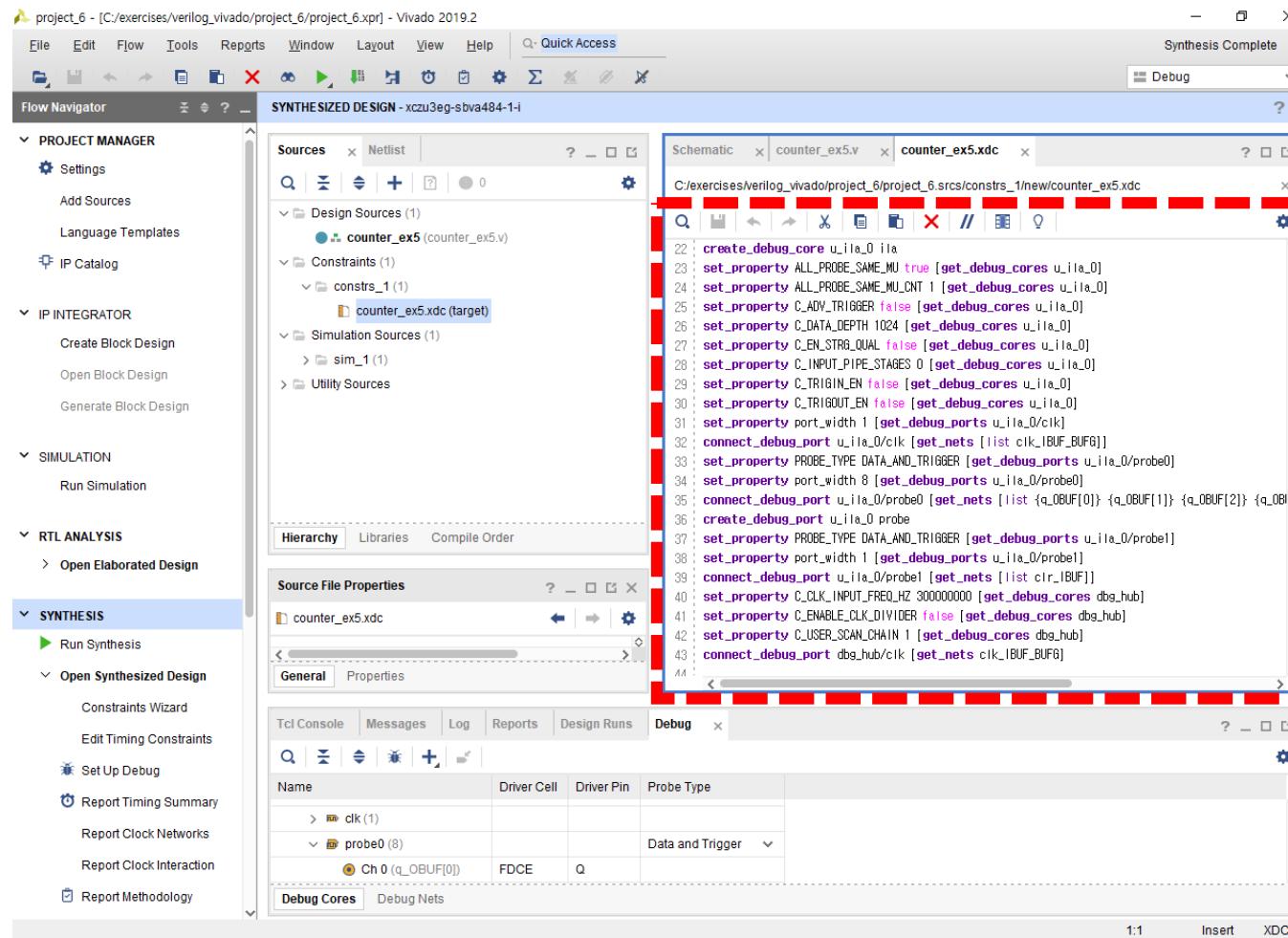
❖ Summary Window가 나오면
설정 내용을 확인한 후 Finish
버튼을 클릭한다.

Step 4 Add ILA on Netlist 5



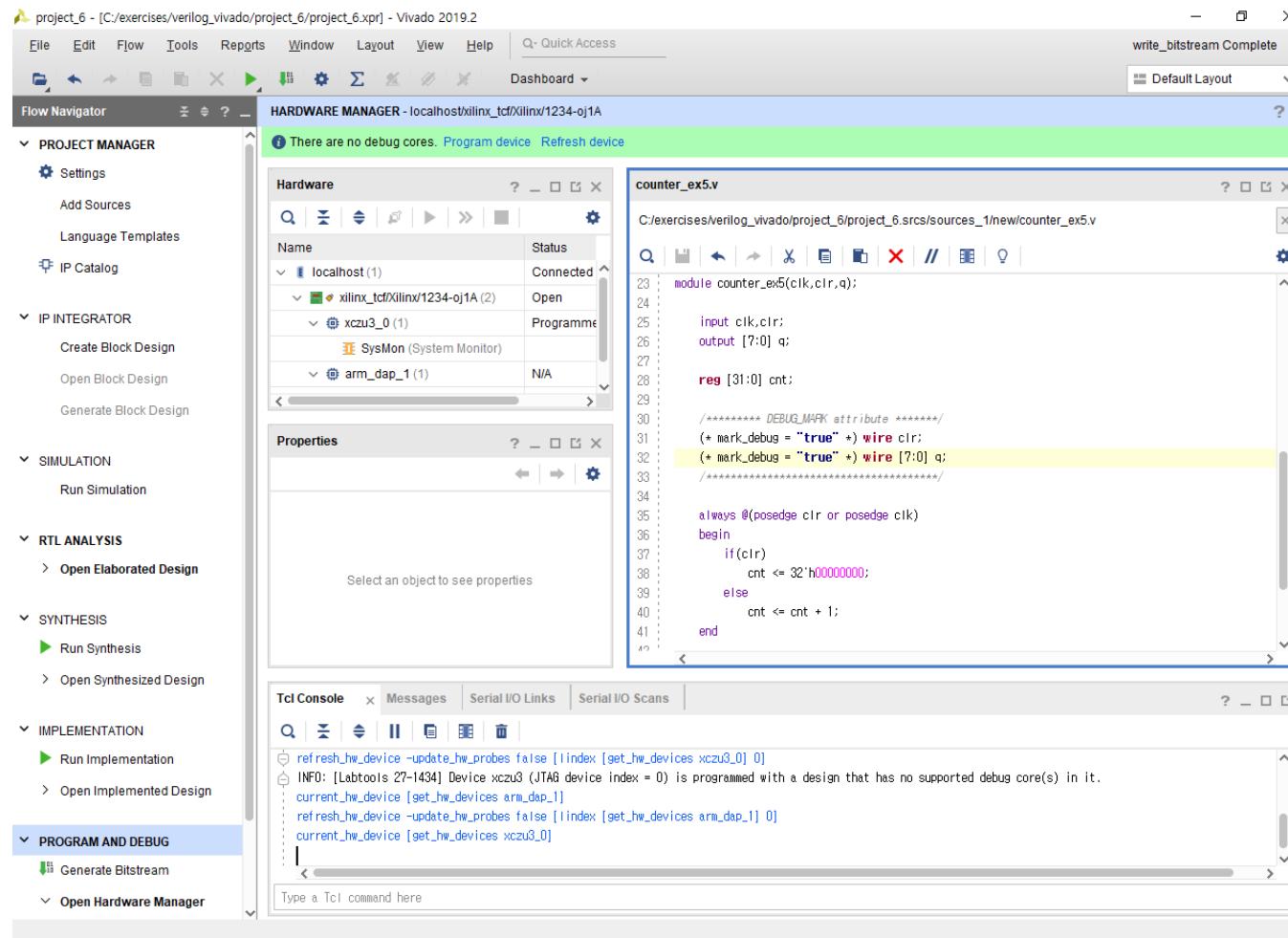
❖ Vivado 하단 Window의 Debug 탭을 보면 probe0와 probe1에 q와 clr이 Data and Trigger Probe Type으로 설정되어 있는 것을 확인할 수 있다.

Step 4 Add ILA on Netlist 6



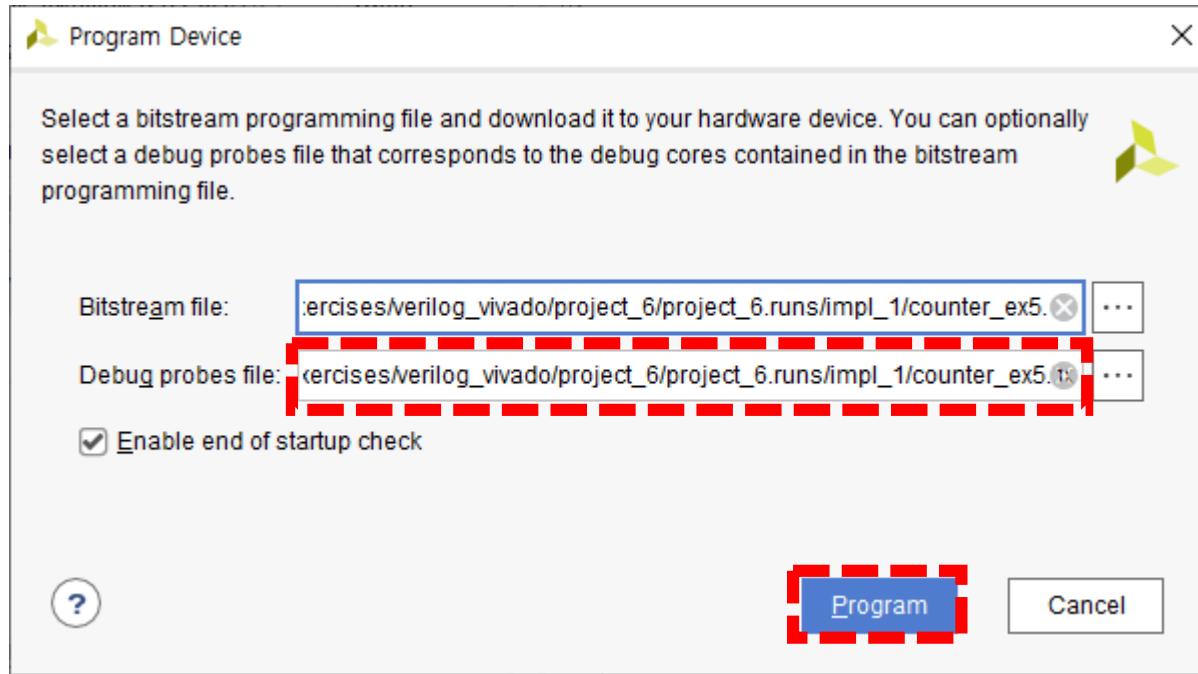
- ❖ Ctrl+S 또는 File ⇒ Constraints ⇒ Save 메뉴를 클릭하여 Constraints를 저장한다.
- ❖ Sources Window의 counter_ex5.xdc파일을 더블 클릭하여 내용을 보면 ILA를 추가한 Constraints가 추가되어 있는 것을 확인할 수 있다.

Step 5 Generating Bitstream and Programming Device 1



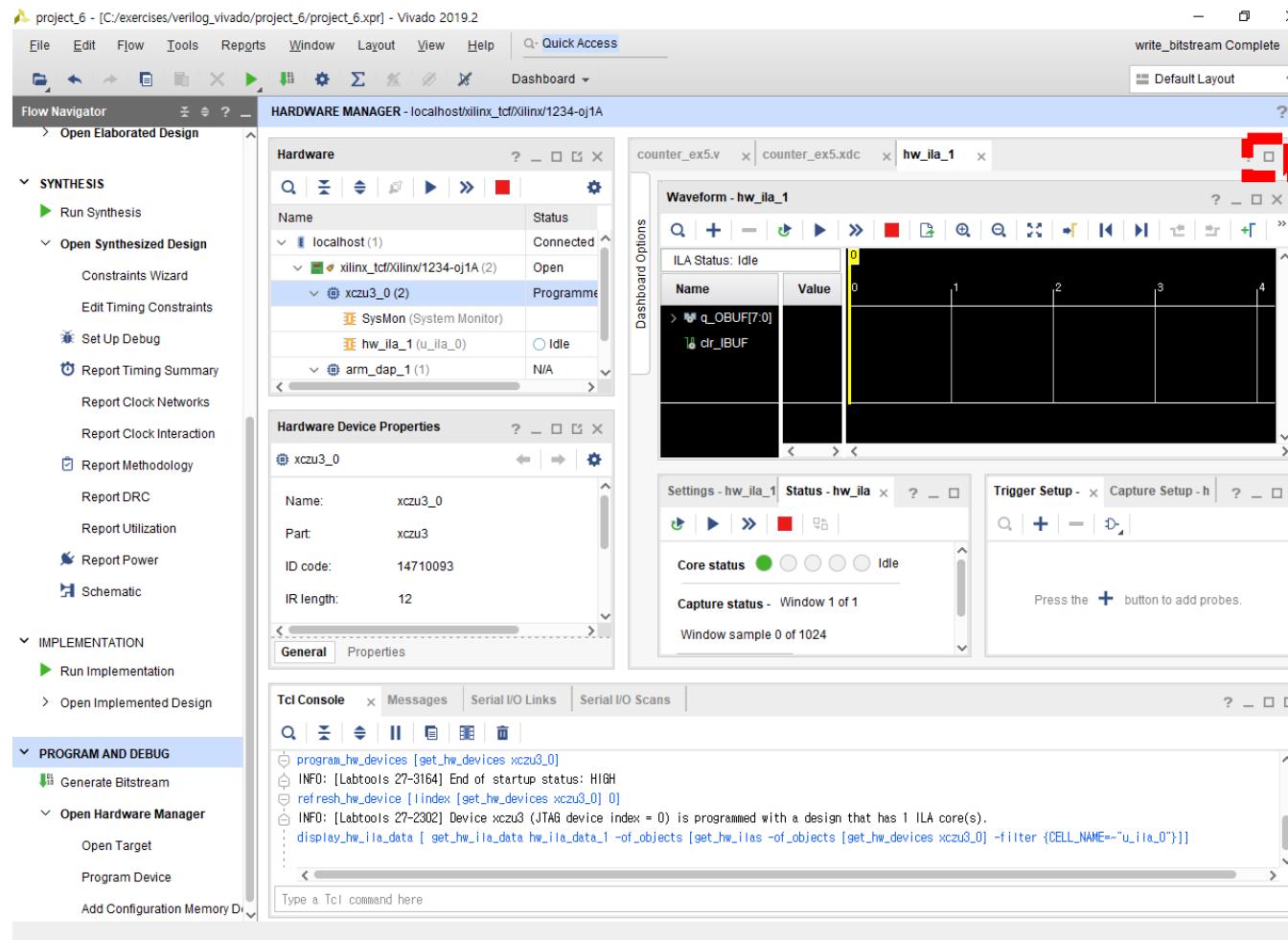
❖ Chapter 2 Vivado Design
Flow의 Step 6를 참고하여
Bitstream을 생성한다.

Step 5 Generating Bitstream and Programming Device 2



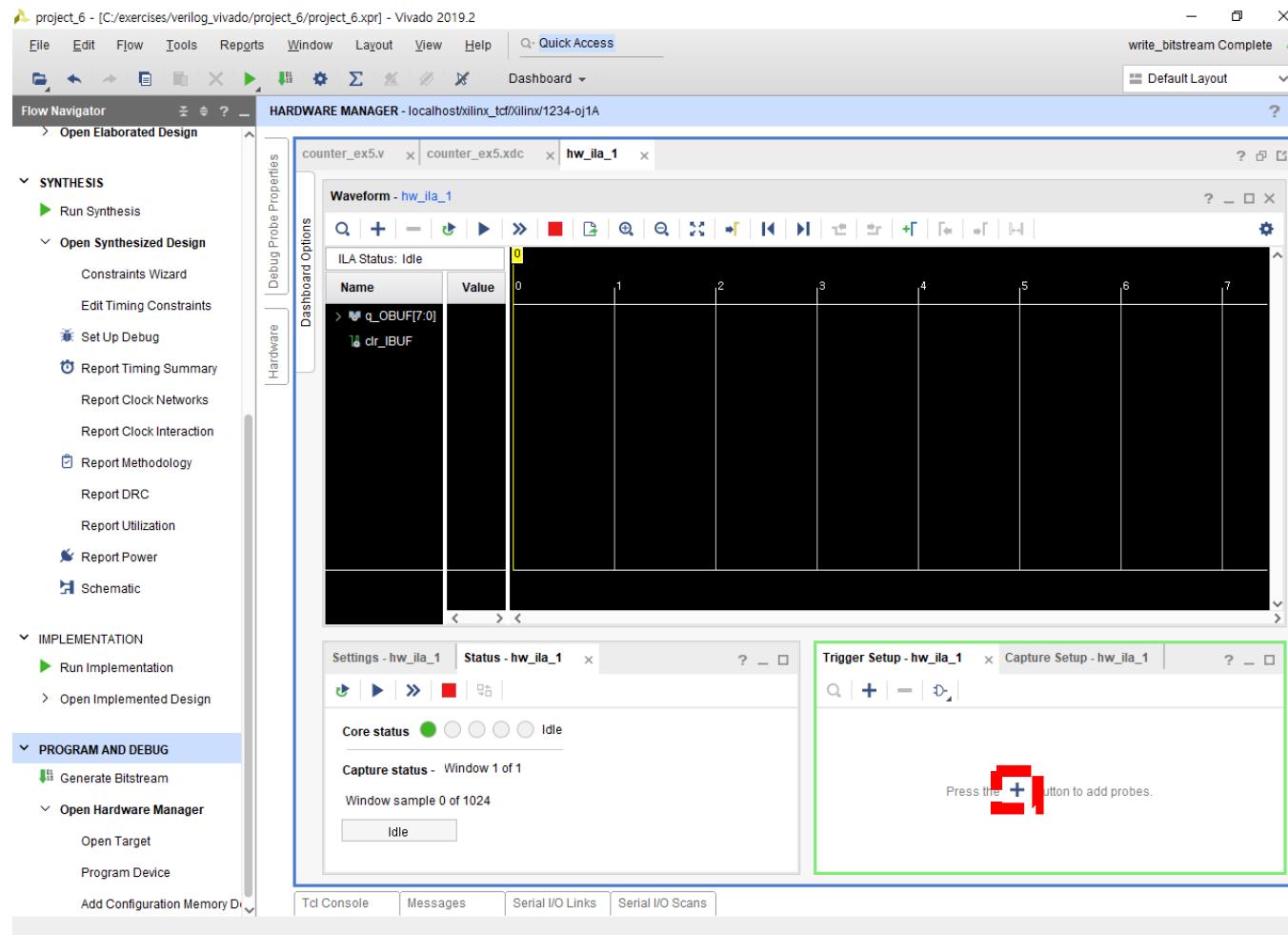
- ❖ Program device 버튼을 클릭하여 Program Device Window가 나오면 Bitstream file과 함께 디폴트로 Debug probes file이 입력되어 있는 것을 확인할 수 있다.
- ❖ Program 버튼을 클릭하여 Bitstream과 Debug probes를 프로그래밍한다.

Step 7 Debug Signal using ILA 1



❖ 프로그래밍이 완료되어
hw_ila_1 Window가 나오면 를 클릭하여 화면을 확장한다.

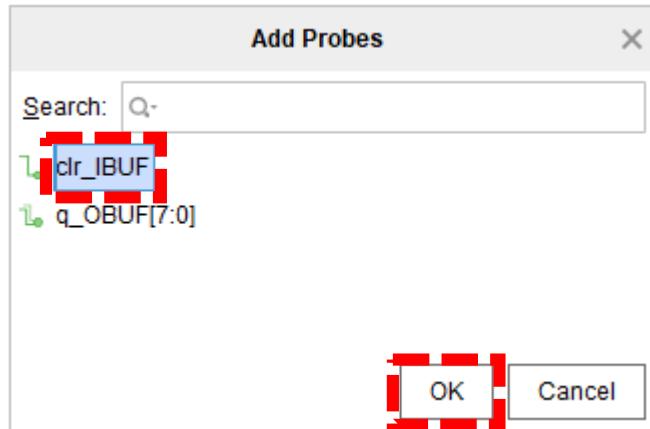
Step 7 Debug Signal using ILA 2



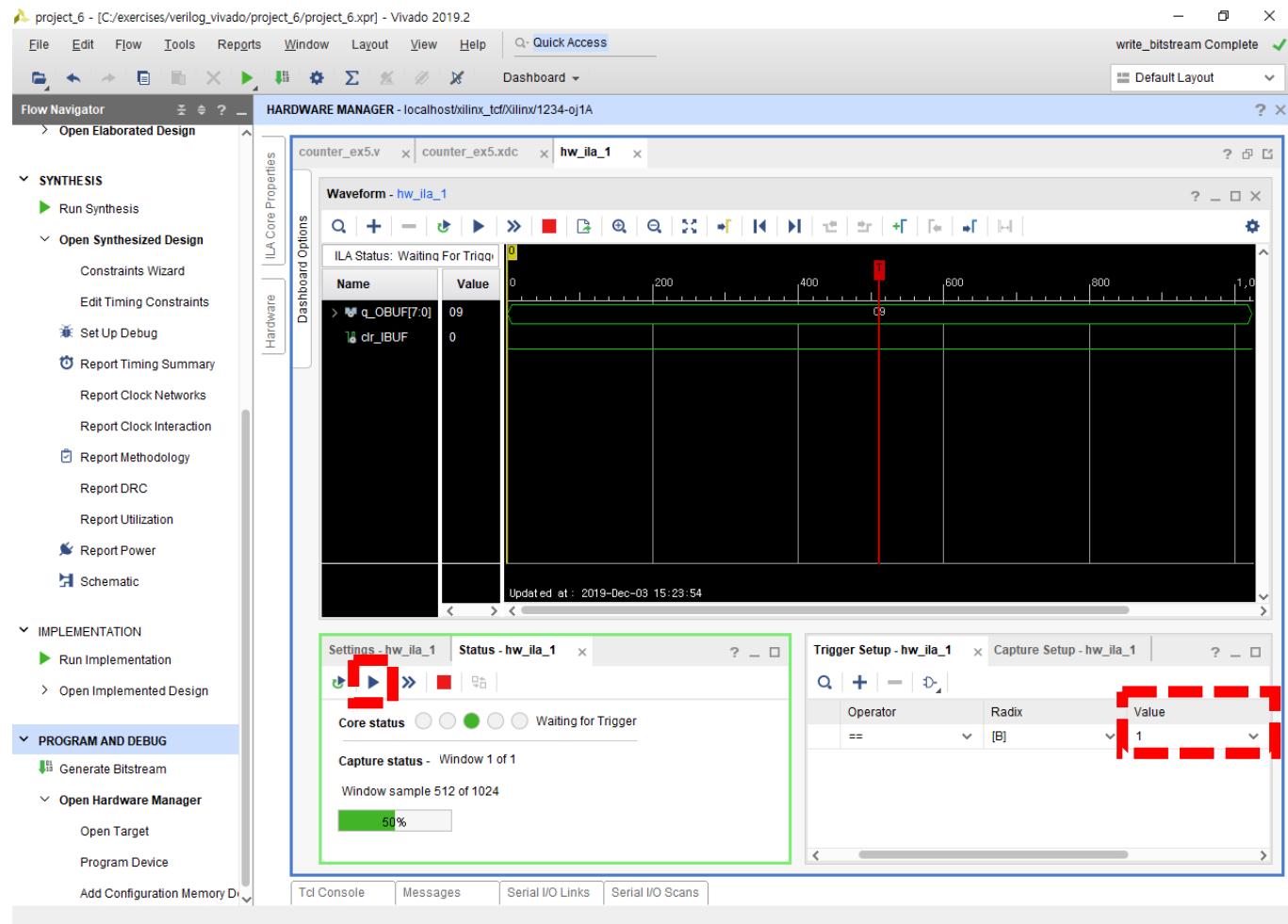
◆ 우측 하단 Window의 Trigger
Setup – hw_ilia_1 탭 안에 +
버튼을 클릭한다.

Step 7 Debug Signal using ILA 3

❖ Add Probes Window가 나오면 clr_IBUF 선택한 후 OK버튼을 클릭한다.

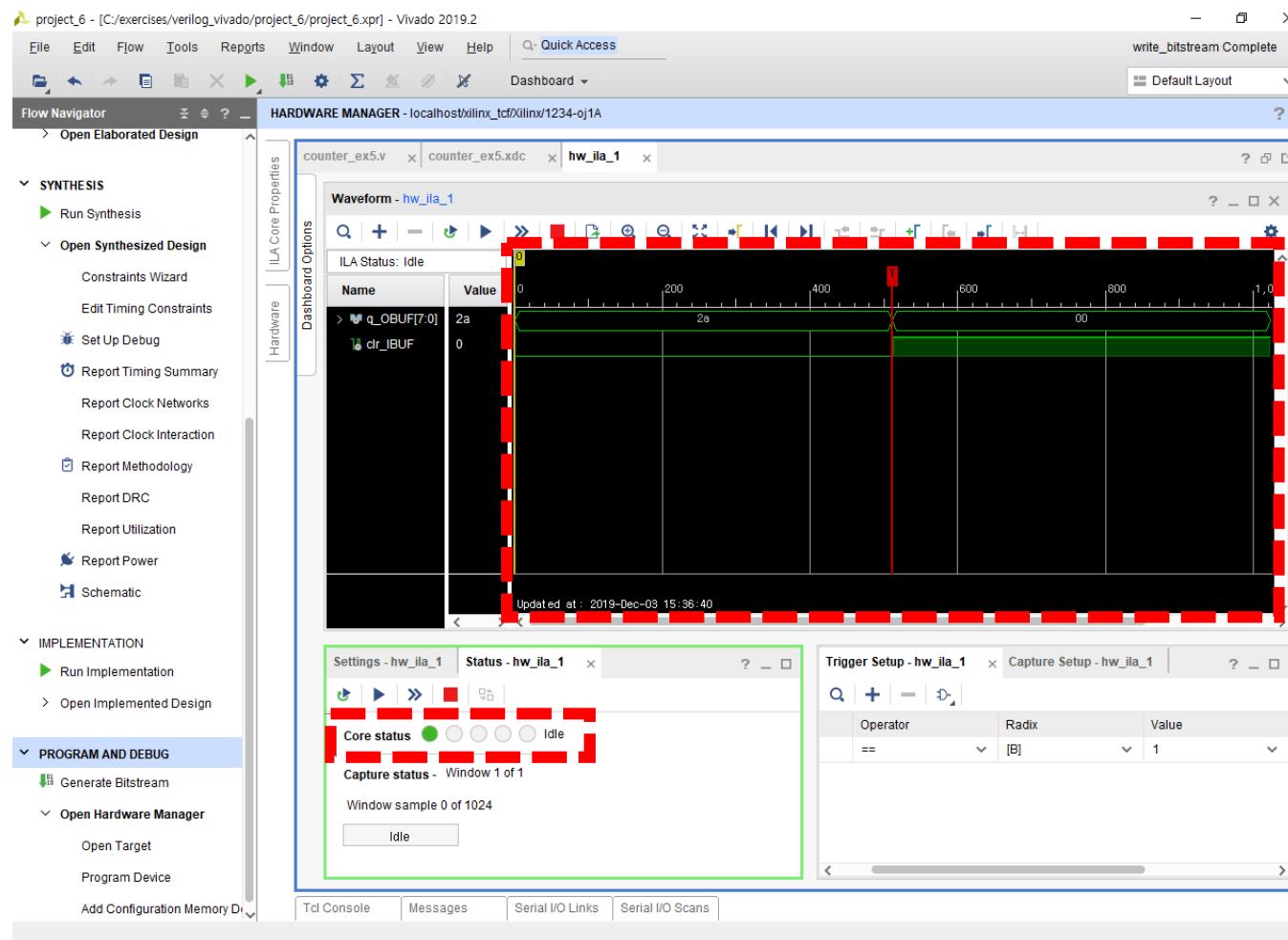


Step 7 Debug Signal using ILA 4



- ❖ 우측 하단 Window의 Trigger Setup
 - `hw_ila_1` 탭 안에 `clr_IBUF`가 추가된 것을 확인할 수 있다.
- ❖ `clr_IBUF`의 Value를 1로 설정한다.
(※ `clr_IBUF`의 값이 1'b1과 같아지는 순간 신호를 캡쳐 한다는 의미이다.)
- ❖ 좌측 하단 Window의 Status –
 `hw_ila_1` 탭 안에 ▶ 버튼을 클릭하면 Core Status가 Waiting for Trigger 상태로 바뀌는 것을 확인할 수 있다. (※ `clr_IBUF`의 값이 1'b1이 될 때까지 기다리는 상태이다.)

Step 7 Debug Signal using ILA 5



- ❖ PMOD_A 커넥터에 연결된 Pmod BTN의 BTN0버튼을 누르면 좌측 하단 Window의 Status –hw_il_1 탭 안에 Core Status가 Idle상태로 바뀌는 것을 확인할 수 있다.
- ❖ hw_il_1 탭의 Waveform을 보면 clr_IBUF가 1'b0에서 1'b1로 바뀌고 그 순간의 q_OBUF의 값이 8'h93에서 8'h00으로 Clear되는 것을 확인할 수 있다.
- ▶ 버튼을 클릭하면 Core Status가 Waiting for Trigger 상태로 다시 바뀌고 clr_IBUF의 값이 1'b1이 될 때까지 기다린다. (※ BTN0 버튼을 누를 때 LED의 값과 Waveform에서 보여주는 값이 일치하는지 확인해 보자.)

Chapter 8 Hardware Debugging

- Timing Constraints
- Static Timing Analysis
- How to Debug Hardware Directly
- Debugging using ILA
- Ultra96 Training Kit Exercises 5

Ultra96 Training Kit Exercises 5

- ❖ EX5-1 Chapter 7에서 만든 카운터 예 중 하나를 선택하여 MARK_DEBUG attribute를 추가하고 Integrated Logic Analyzer(ILA) 기능을 이용하여 카운터의 출력값을 계측하시오.
- ❖ EX5-2 Chapter 7에서 만든 카운터 예 중 하나를 선택하여 Analog Discovery 2를 통해 카운터의 출력값을 계측하시오.

Chapter 9 Memory

- **Creating Memory**
- ROM (Read Only Memory)
- Memory IP Cores
- Ultra96 Training Kit Exercises 6

Memory

- ❖ 디지털 회로에서 메모리는 코드 또는 데이터 등을 저장하는데 사용한다.
- ❖ 많은 디지털 회로들은 메모리에 있는 데이터를 가지고 와서 데이터를 처리 후에 메모리에 다시 저장하는 일을 수행한다.
- ❖ 많은 하드웨어 모듈들이 메모리에 접근하여 데이터가 처리되므로 메모리를 어떻게 설계하는지에 따라 전체 하드웨어의 성능을 좌우한다.
- ❖ 메모리를 Verilog로 어떻게 표현하여 사용하는지 알아본다.

Memory Example 1

- ❖ 메모리 예를 보면 reg [15:0] mem [0:7] 로 배열을 사용하여 메모리를 선언하였다. (※ width가 16-bit 크기이고 depth가 8개인 메모리를 의미한다.)
- ❖ always block은 선언된 메모리를 어떻게 읽고 쓰는지에 대한 내용을 표현하였다.
- ❖ posedge clk을 Sensitivity List에 넣어서 클럭 Rising Edge에 읽고 쓰여 진다.
- ❖ wr_en신호가 1(High)일 때 메모리의 address 포트로 입력되는 메모리 번지에 data 포트로 입력되는 값이 쓰여지고 메모리의 address 포트로 입력되는 주소의 메모리 값은 mem_out 포트로 출력된다.

```
module mem_ex1(clk,wr_en,data,address,mem_out);  
  
    input clk,wr_en;  
    input [15:0] data;  
    input [2:0] address;  
    output reg [15:0] mem_out;  
  
    reg [15:0] mem [0:7];  
  
    always @ (posedge clk)  
    begin  
        if(wr_en) begin  
            mem[address] <= data;  
        end  
        mem_out <= mem[address];  
    end  
  
endmodule
```

Memory Example 2

- ❖ 메모리 예를 보면 data 포트로 입력된 데이터는 wr_addr 포트로 입력되는 주소에 쓰여지고 rd_addr 포트로 입력되는 주소의 메모리 값은 mem_out 포트로 출력된다.
- ❖ address 포트만 둘로 나누어져 있지만 data 포트 또는 control포트들도 읽는 것과 쓰는 것으로 나누어서 두 개씩 만들 수 있다.
- ❖ 이와 같이 설계된 메모리는 데이터를 쓰는 기능과 읽는 기능이 완벽히 구분되어 있어서 동시에 수행할 수 있다.

```
module mem_ex2(clk,wr_en,data,rd_addr,wr_addr,mem_out);  
  
    input clk,wr_en;  
    input [15:0] data;  
    input [2:0] rd_addr,wr_addr;  
    output reg [15:0] mem_out;  
  
    reg [15:0] mem [0:7];  
  
    always @ (posedge clk)  
    begin  
        if(wr_en) begin  
            mem[wr_addr] <= data;  
        end  
        mem_out <= mem[rd_addr];  
    end  
  
endmodule
```

Memory Example 3

- ❖ 두 개의 clock을 사용하고 두 개의 data/address포트를 가지고 있는 Dual-clock Dual-port Memory이다.
- ❖ 다른 모듈로부터 입력되는 입력데이터와 다른 모듈로 출력되는 출력데이터의 속도가 다를 때 이와 같은 회로를 사용 할 수 있다.

```
module mem_ex3(wr_clk,rd_clk,wr_en,data,rd_addr,wr_addr,mem_out);  
    input wr_clk,rd_clk,wr_en;  
    input [15:0] data;  
    input [2:0] rd_addr,wr_addr;  
    output reg [15:0] mem_out;  
    reg [15:0] mem [0:7];  
  
    always @ (posedge wr_clk)  
    begin  
        if(wr_en) begin  
            mem[wr_addr] <= data;  
        end  
    end  
  
    always @ (posedge rd_clk)  
    begin  
        mem_out <= mem[rd_addr];  
    end  
endmodule
```

Memory Example 4

❖ 메모리를 생성하였을 때 메모리의 초기 값을 설정하고 싶다면 Chapter 5에서 언급한 것과 같이 initial block을 사용하여 초기 값을 설정할 수 있다.

```
module mem_ex4(wr_clk,rd_clk,wr_en,  
    data,rd_addr,wr_addr,mem_out);  
  
    input wr_clk,rd_clk,wr_en;  
    input [15:0] data;  
    input [2:0] rd_addr,wr_addr;  
    output reg [15:0] mem_out;  
  
    reg [15:0] mem [0:7];  
  
    initial begin  
        mem[0] = 16'habcd;  
        mem[1] = 16'h79ca;  
        mem[2] = 16'h1358;  
        mem[3] = 16'h976a;  
        mem[4] = 16'h84ad;  
        mem[5] = 16'hd3f5;  
        mem[6] = 16'hf4a2;  
        mem[7] = 16'hc0d1;  
    end  
  
    always @ (posedge wr_clk)  
    begin  
        if(wr_en) begin  
            mem[wr_addr] <= data;  
        end  
    end  
  
    always @ (posedge rd_clk)  
    begin  
        mem_out <= mem[rd_addr];  
    end  
endmodule
```

Memory Example 5

❖ 코드 예제에서는 reg [15:0] mem [0:1][0:3]으로 2차원 배열을 사용하여 메모리를 선언하였다.

❖ 기존의 1차원 배열로 선언한 것과 비교해보면 생성되는 메모리는 동일하지만 메모리를 Addressing하는 주소가 2개로 달라짐을 알 수 있다.

```
module mem_ex5(wr_clk,rd_clk,wr_en,  
data,mem_out,rd_addr0,rd_addr1,  
wr_addr0,wr_addr1);  
  
input wr_clk,rd_clk,wr_en;  
input [15:0] data;  
input [0:0] rd_addr0,wr_addr0;  
input [1:0] rd_addr1,wr_addr1;  
output reg [15:0] mem_out;  
reg [15:0] mem [0:1][0:3];  
  
initial begin  
    mem[0][0] = 16'habcd;  
    mem[0][1] = 16'h79ca;  
    mem[0][2] = 16'h1358;  
    mem[0][3] = 16'h976a;  
  
    mem[1][0] = 16'h84ad;  
    mem[1][1] = 16'hd3f5;  
    mem[1][2] = 16'hf4a2;  
    mem[1][3] = 16'hc0d1;  
end  
  
always @ (posedge wr_clk)  
    if(wr_en)  
        mem[wr_addr0][wr_addr1] <= data;  
  
always @ (posedge rd_clk)  
    mem_out <= mem[rd_addr0][rd_addr1];  
  
endmodule
```

Chapter 9 Memory

- Creating Memory
- ROM (Read Only Memory)
- Memory IP Cores
- Ultra96 Training Kit Exercises 6

ROM

- ❖ ROM은 Read Only Memory의 약자로 읽는 것만 가능하고 쓰는 것은 불가능한 메모리를 말한다.
- ❖ ROM이 앞에서 설명한 메모리와 다른 점은 쓰는 기능이 없어서 저장되어 있는 값이 바뀌지 않는 것이다.
- ❖ 앞에서 생성한 메모리 코드 예제 중에서 wr_en을 0(LOW)으로 고정하여 입력하거나 always block안에 메모리에 값을 쓰도록 표현된 내용을 삭제하면 ROM으로 사용할 수 있다.

ROM Example 1

❖ mem_ex4 모듈에서 쓰는 기능을 뺀 모듈로 ROM을 표현한 것이다.

```
module rom_ex1(clk,addr,rom_out);
    input clk;
    input [2:0] addr;
    output reg [15:0] rom_out;
    reg [15:0] rom [0:7];
    initial begin
        rom[0] = 16'habcd;
        rom[1] = 16'h79ca;
        rom[2] = 16'h1358;
        rom[3] = 16'h976a;
        rom[4] = 16'h84ad;
        rom[5] = 16'hd3f5;
        rom[6] = 16'hf4a2;
        rom[7] = 16'hc0d1;
    end
    always @ (posedge clk)
        rom_out <= rom[addr];
endmodule
```

ROM Example 1

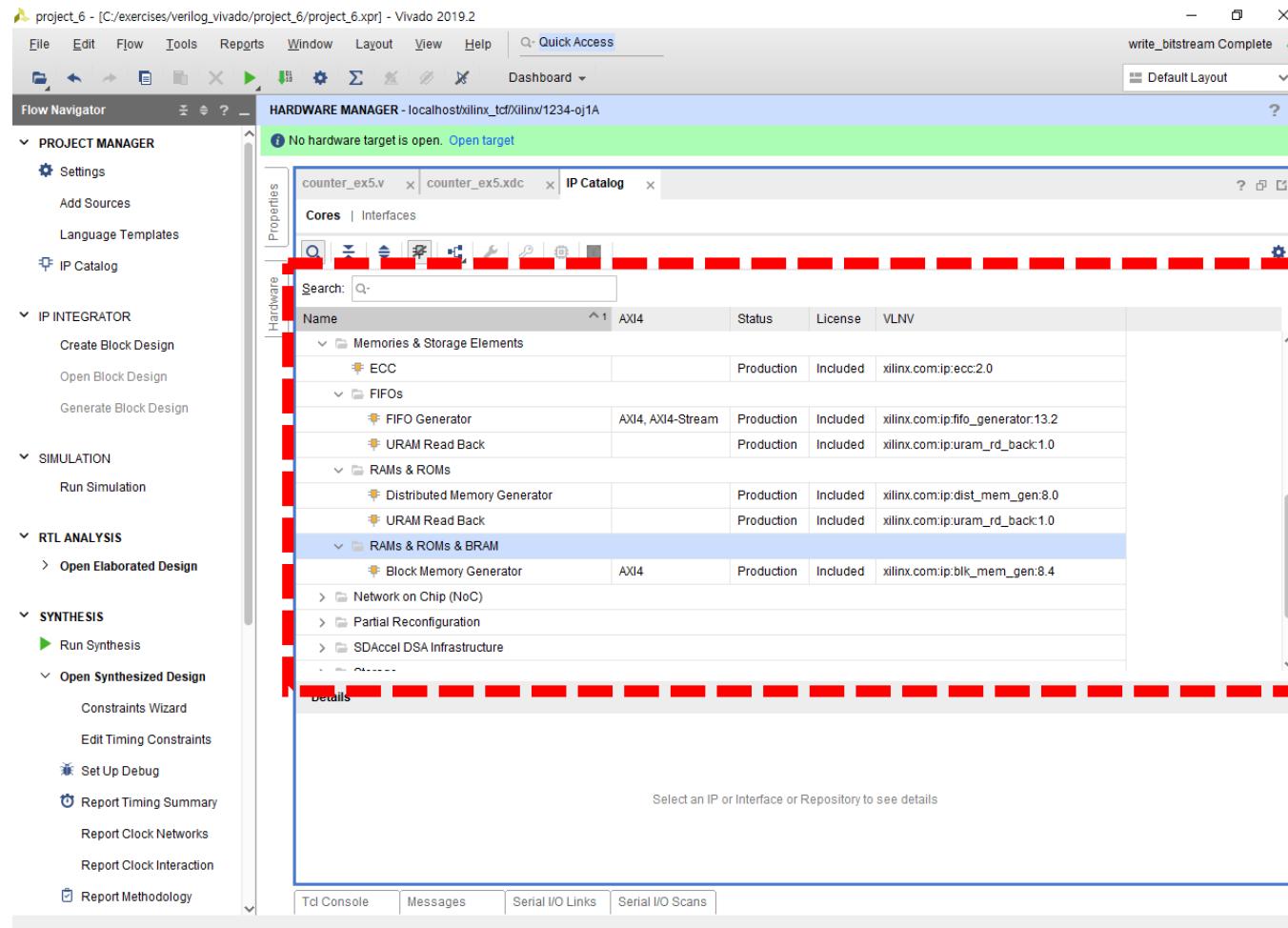
- ❖ case Statement를 사용하여 ROM을 표현한 모듈이다.
- ❖ rom_ex1 모듈과 rom_ex2 모듈은 서로 표현한 내용은 다르지만 합성한 결과물은 동일한 회로가 된다.
- ❖ Vivado에서 Flow Navigator ⇒ Synthesis ⇒ Schematic을 클릭하여 합성된 회로를 비교해보면 확인할 수 있다.

```
module rom_ex2(clk,addr,rom_out);
    input clk;
    input [2:0] addr;
    output reg [15:0] rom_out;
    always @ (posedge clk)
    begin
        case (addr)
            3'b000 : rom_out = 16'habcd;
            3'b001 : rom_out = 16'h79ca;
            3'b010 : rom_out = 16'h1358;
            3'b011 : rom_out = 16'h976a;
            3'b100 : rom_out = 16'h84ad;
            3'b101 : rom_out = 16'hd3f5;
            3'b110 : rom_out = 16'hf4a2;
            3'b111 : rom_out = 16'hc0d1;
        endcase
    end
endmodule
```

Chapter 9 Memory

- Creating Memory
- ROM (Read Only Memory)
- Memory IP Cores
- Ultra96 Training Kit Exercises 6

Memory IP Cores 1



- ❖ Vivado에서 제공하는 IP Core를 생성하여 Memory로 사용할 수 있다.
- ❖ Flow Navigator ⇒ Project Manager ⇒ IP Catalog를 클릭한 후 카탈로그 목록 중에 Memories & Storage Elements 아래 내용을 보면 Memory를 생성할 수 있는 IP Core들이 있다.

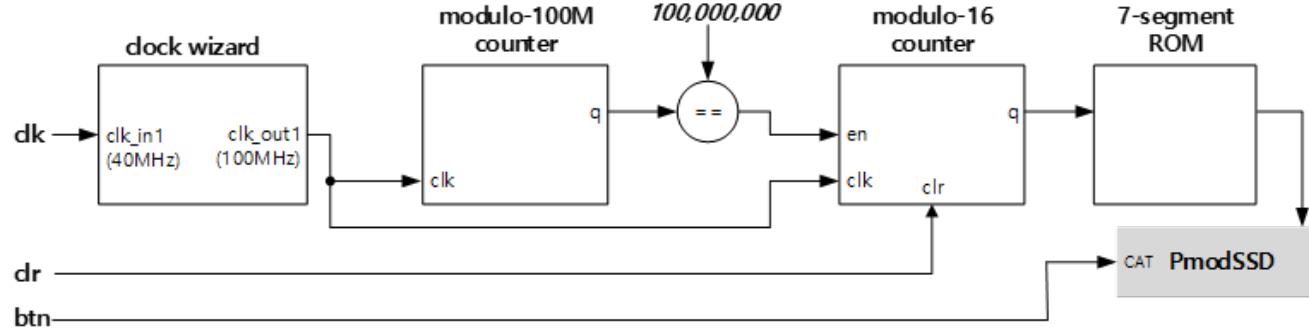
Memory IP Cores 2

- ❖ Distributed Memory Generator는 LUT를 사용하여 Memory를 생성하는 IP Core이고 Block Memory Generator는 BlockRAM을 사용하여 Memory를 생성하는 IP Core이다.
- ❖ FIFO Generator를 사용하면 FIFO(First Input First Output) IP Core를 생성하여 사용할 수 있다.
- ❖ FIFO는 일반 메모리처럼 특정 주소에 값을 쓰거나 읽는 형태의 메모리가 아니고 데이터를 쓰기 위한 신호를 보내면 입력되는 데이터를 계속 쓰고 읽기 위한 신호를 보내면 FIFO안의 데이터 중 먼저 들어온 데이터부터 순서대로 출력하는 메모리이다.
- ❖ 먼저 입력된 것이 먼저 출력된다고 하여 FIFO(First Input First Output)라고 부른다.
- ❖ FIFO는 데이터를 쓰는 회로와 읽는 회로가 서로 동기가 맞지 않아서 중간에 버퍼의 역할이 필요할 때 사용한다.

Chapter 9 Memory

- Creating Memory
- ROM (Read Only Memory)
- Memory IP Cores
- Ultra96 Training Kit Exercises 6

Ultra96 Training Kit Exercises 6



- ❖ EX6-1 앞에서 만든 메모리 예 중 하나를 선택하여 그 모듈과 테스트벤치를 작성하고 시뮬레이션을 통해 검증을 하시오.
- ❖ EX6-2 앞에서 만든 ROM 예 중 하나를 선택하여 그 모듈과 테스트벤치를 작성하고 시뮬레이션을 통해 검증을 하시오.
- ❖ EX6-3 7-Segment에 0~F까지 표시할 수 있는 8-bit 데이터 16개를 ROM에 저장하면 ROM의 입력인 Address는 4-bit가 되고 출력인 Data는 8-bit가 된다. 이 Data 출력을 7-Segment에 연결하면 0부터 F까지 출력할 수 있다. 이전에 배운 Clocking Wizard, Modulo Counter 등을 활용하고, 아래 블록도를 참고하여, 7-Segment에 0부터 F까지 1초에 한 번씩 표시되도록 구현하시오.

Chapter 10 FSM

- **Introduction to FSM**
- **FSM Implementation**
- **Ultra96 Training Kit Exercises 7**

FSM

- ❖ FSM은 Finite State Machine의 약자로 한정되어 있는 상태를 가진 머신이라는 의미를 가지고 있다.
- ❖ FSM은 하드웨어가 어떻게 동작해야 하는지를 한정된 상태들의 변화에 의해서 표현한 것이다.
- ❖ FSM은 이런 상태들의 변화를 설계하여 하드웨어의 두뇌 역할을 하는 컨트롤러를 설계할 때 사용한다. (※ FSM은 우리 주변의 간단한 전자기기에서부터 복잡한 프로세서까지 컨트롤러 역할을 하는 모듈을 설계할 때 사용한다.)
- ❖ 컨트롤러를 FSM으로 어떻게 만드는지 알아본다.

How to Design FSM

- ❖ FSM을 설계하려면 먼저 FSM이 어떻게 동작해야 하는지 파악해야 한다.
- ❖ 스테이트 다이어그램 (State Diagram)으로 표현하여 FSM이 어떻게 동작하는지를 구체적으로 명시한다.
- ❖ 스테이트 다이어그램을 토대로 Verilog 로 표현하여 FSM 설계를 완성할 수 있다.

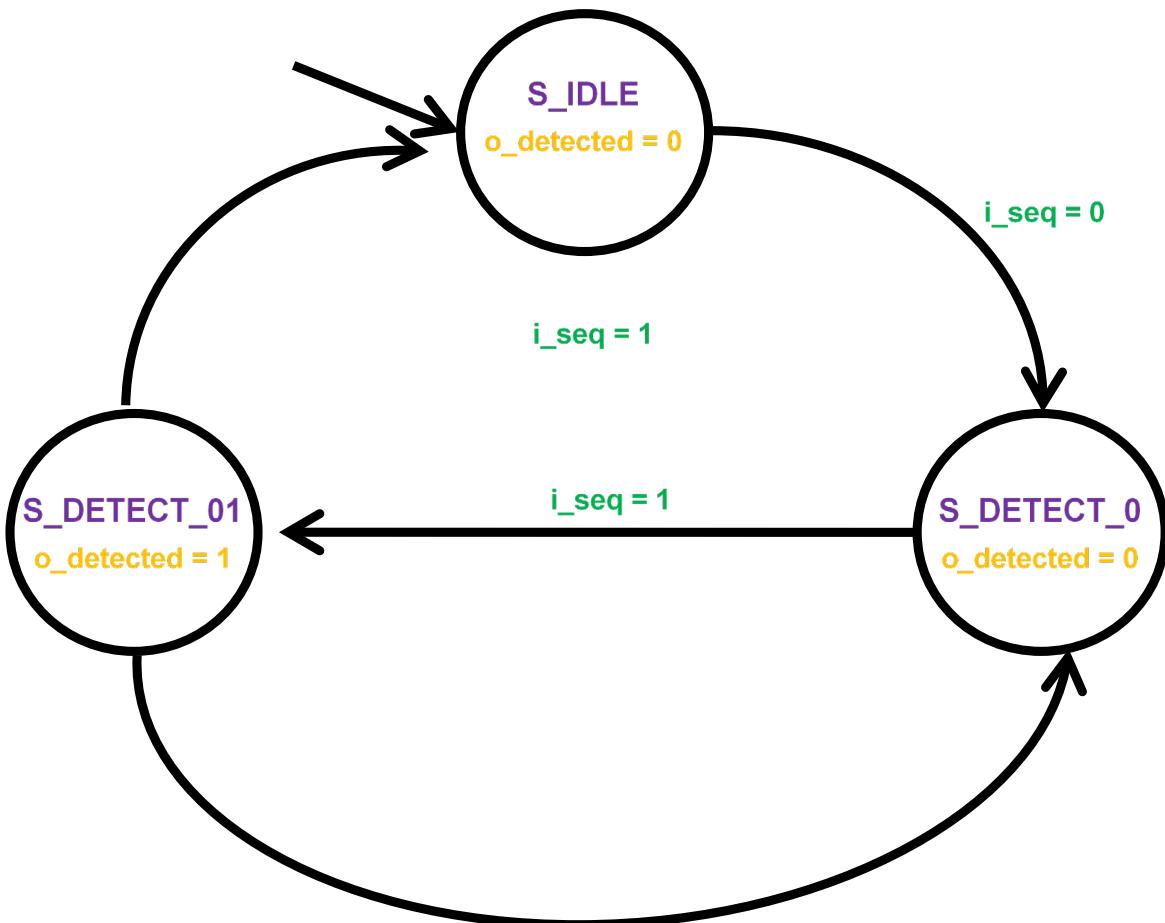
FSM Example

- ❖ 앞에서 사용해왔던 Pmod BTN을 예를 들어 보자.
- ❖ 푸시 버튼으로부터 1이 입력될 때 특정 동작을 하려면 버튼을 계속 누르고 있어야 한다. 또한, 버튼 입력 값만으로는 버튼을 누른 시점을 정확히 판단하지도 못한다.
- ❖ 만약 버튼을 누른 순간을 검출할 수 있는 회로가 있다면 그 뒤에 추가적인 모듈을 연결하여 버튼을 누를 때마다 0과 1이 토글되도록 할 수도 있고, 3가지 이상의 옵션을 순차적으로 고르게 할 수도 있다.
- ❖ 버튼을 누르는 순간의 입력 값은 여러 클럭에 걸쳐 "...00000011111..."의 형태로 시퀀스가 변화하기 때문에 결국 "01" 시퀀스 검출 회로를 설계하면 된다.
- ❖ 이 방식도 버튼을 누르는 순간에 on/off가 수없이 반복되는 채터링 문제가 있으면 사용할 수 없지만, Pmod BTN 모듈의 경우 채터링 방지 회로가 내장되어 있으므로 이 부분은 고려하지 않는다.

Create a State Diagram for Example

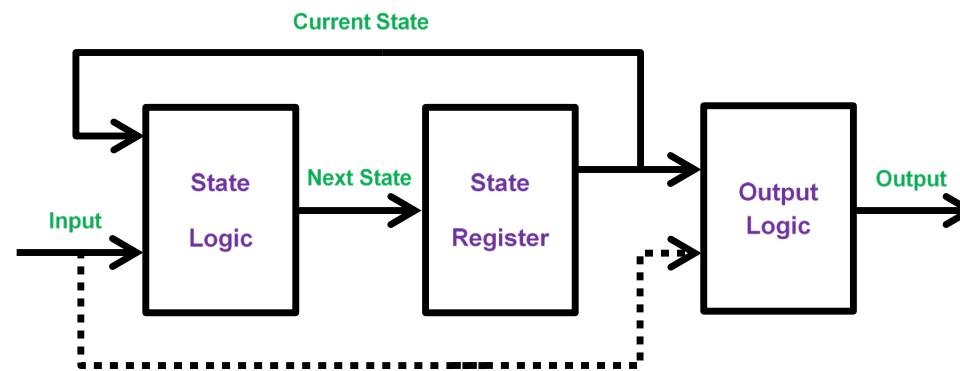
- ❖ 스테이트 디어그램을 작성하려면 제일 먼저 스테이트를 몇 개로 할 것인지 고려해봐야 한다.
- ❖ 회로가 리셋되거나 시퀀스를 처음부터 다시 기다리는 S_IDLE상태, 시퀀스 "0"이 검출된 S_DETECT_0상태, 시퀀스 "01"이 검출된 S_DETECT_01상태로 표현할 수 있다.
- ❖ 이것을 스테이트 디어그램으로 표현하면 왼쪽 그림과 같다.

Create a State Diagram for FSM Example



- ❖ State Diagram을 작성하려면 제일 먼저 State를 몇 개로 할 것인지 고려해봐야 한다.
- ❖ 회로가 리셋되거나 시퀀스를 처음부터 다시 기다리는 S_IDLE상태, 시퀀스 "0"이 검출된 S_DETECT_0상태, 시퀀스 "01"이 검출된 S_DETECT_01상태로 표현할 수 있다.
- ❖ State Diagram에서 동그라미는 스테이트를 표현한 것이고 화살표는 스테이트가 바뀌는 것(State Transition)을 표현한 것이다.
- ❖ State가 바뀌는 것을 표현한 화살표 위의 신호들은 Input Port로 들어오는 Input Signal들이고 State 이름 아래 신호들은 Output Port로 나가는 Output Signal들이다.

Create a Block Diagram for FSM Example



- ❖ State Diagram을 Block Diagram으로 표현한 그림이다.
- ❖ FSM Example은 3 개의 Hardware Block으로 구현할 수 있다.
- ❖ State Logic은 Current State와 Input 신호를 입력 받아서 Next State가 무엇이 될지 결정하는 로직이다.
- ❖ State Logic Block에서 출력된 Next State는 State Register에 들어오는 클럭 신호가 Rising Edge가 되면 Current State로 출력된다.
- ❖ Output Logic은 Current State와 Input Signal을 입력 받아서 Output이 무엇이 될지 결정하는 로직이다.
- ❖ 참고로 Output Logic에 Current State 신호만 입력되는 FSM을 Moore Machine이라 부르고 Current State와 Input 신호가 입력되는 FSM 을 Mealy Machine이라고 부른다.

Chapter 10 FSM

- Introduction to FSM
- **FSM Implementation**
- Ultra96 Training Kit Exercises 7

FSM Implementation 1

- ❖ 01시퀀스 검출기의 포트인 clk, rst, i_seq Input Port와 o_detected Output Port를 표현해 준다.
- ❖ parameter로 State들을 0,1,2로 Encoding한다.
(※ parameter로 선언하면 readability가 좋아져서 코드 해석을 좀 더 쉽게 할 수 있도록 해준다.)
- ❖ State Transition을 구현하기 위해 current_state 와 next_state를 reg로 선언한다.
- ❖ 3개의 Block으로 표현된 Block Diagram과 같이 3개의 Block을 각각 3개의 always Block으로 표현해준다.

```
module detect_01_fsm(  
    rst,  
    clk,  
    i_seq,  
    o_detected  
);  
    input rst;  
    input clk;  
    input i_seq;  
    output reg o_detected;  
  
    parameter S_IDLE=0, S_DETECT_0=1, S_DETECT_01=2;  
    reg [1:0] current_state, next_state;
```

FSM Implementation 2

- ❖ State Logic Block을 하나의 always Block으로 표현한 코드이다.
- ❖ Sensitivity List에 State Logic 블록의 입력인 current_state와 입력신호들 대신 *를 입력하였다.
- ❖ next_state <= current_state; Statement는 Default Condition이라고 부른다. (※ Default Condition이란 순차적으로 해석되는 always 안에서 뒤에서 내용이 바뀌지 않으면 디폴트 컨디션이 적용된다는 의미로 반복되는 코드를 줄여 주기 위해 사용한다.)
- ❖ case Statement를 사용하여 current_state가 어디인지를 판별하고 if Statement를 사용하여 입력되는 신호를 판별하여 next_state를 결정하는 내용을 표현한다.

```
always @(*)
begin
    next_state <= current_state;
    case (current_state)
        S_IDLE :
            if (i_seq == 1'b0)
                next_state <= S_DETECT_0;
        S_DETECT_0 :
            if (i_seq == 1'b1)
                next_state <= S_DETECT_01;
        S_DETECT_01 :
            if (i_seq == 1'b0)
                next_state <= S_DETECT_0;
            else
                next_state <= S_IDLE;
    default :
        next_state <= S_IDLE;
    endcase
end
```

FSM Implementation 3

- ❖ State Register Block을 하나의 always Block으로 표현한 코드이다.
- ❖ Register를 만들기 위해 posedge clk을 Sensitivity List에 넣는다.
- ❖ rst 신호가 '1'(High)이면 current_state는 S_IDLE상태가 되도록 설계한다.
- ❖ State Logic Block에서 출력된 next_state는 State Register Block에 입력되어 clk신호가 Rising Edge 되는 순간 current_state로 출력된다.
- ❖ 출력된 current_state는 Output Logic Block으로 입력되고 State Logic Block으로도 입력된다.

```
always @(posedge clk)
begin
    if (rst)
        current_state <= S_IDLE;
    else
        current_state <= next_state;
end
```

FSM Implementation 4

- ❖ Output Block을 하나의 always Block으로 표현 한 코드이다.
- ❖ Sensitivity List에 Output Logic Block의 입력인 current_state 대신 * 를 입력하였다.
- ❖ case Statement는 current_state를 판별하여 Output Port인 o_detected로 출력한다.

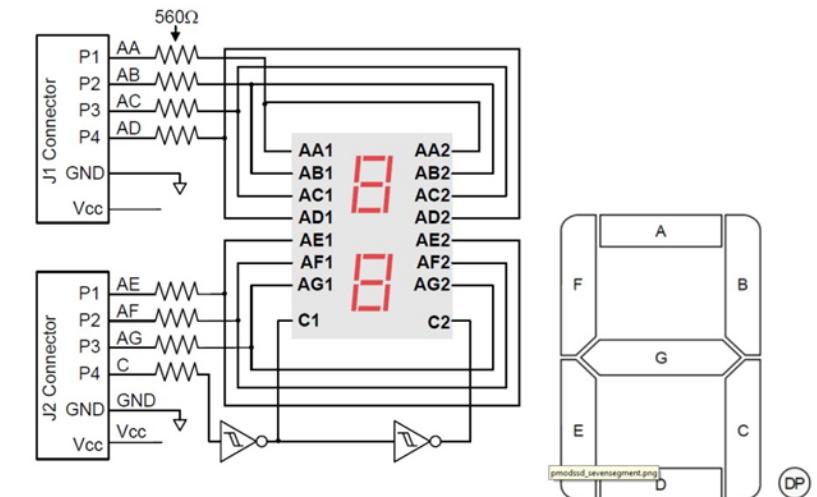
```
always @(*)  
begin  
    case(current_state)  
        S_IDLE:  
            o_detected <= 1'b0;  
        S_DETECT_0:  
            o_detected <= 1'b0;  
        S_DETECT_01:  
            o_detected <= 1'b1;  
        default:  
            o_detected <= 1'b0;  
    endcase  
end
```

Chapter 10 FSM

- Introduction to FSM
- FSM Implementation
- Ultra96 Training Kit Exercises 7

Ultra96 Training Kit Exercises 7

- ❖ EX7-1 앞에서 FSM의 예로 사용한 01시퀀스 검출기의 모듈과 테스트벤치를 작성하고 시뮬레이션을 통해 검증을 하시오.
- ❖ EX7-2 Pmod96보드의 PMOD_A, PMOD_B Connector에 Pmod Splitter Cable을 이용하여 Pmod SSD 2개를 연결하고, PMOD_C Connector에 PmodBTN을 연결한 후, EX6-3을 확장하여 4개의 7-Segments에 분 초를 표시하는 시계를 구현하시오 (※ Pmod SSD를 보면 C pin에 의해서 두 개의 7-Segment 중 하나만 켜지도록 한다. 결국 두 개의 7-Segment를 동시에 켤 수는 없고 C pin에 1ms~10ms 속도의 Clock을 연결하여 일정속도로 좌측과 우측 7-Segment가 켜졌다 꺼지기를 반복하게 한 다음 Clock Signal이 '0'(Low)일 때는 오른쪽 7-Segment에 표시될 값을 출력하고 '1'(High)일 때는 왼쪽 7-Segment에 표시될 값을 출력하면 착시현상에 의해 둘 다 켜져 있는 것으로 보이게 된다.)



Chapter 11 Verilog HDL Advanced Syntax

- Parameter
- Generate
- Subprogram

Parameter 1

- ❖ Parameter는 Chapter 3에서 소개한 내용처럼 기본적으로 상수를 선언하기 위한 용도로 사용하고 모듈이나 IP의 재사용을 위해 사용된다.
- ❖ 오른쪽 코드 예제를 보면 counter 모듈 안에 parameter로 width가 선언되어 있다.
- ❖ 이 counter 모듈을 다른 모듈에서 Instantiation할 때 parameter로 선언된 값들을 재설정하여 Instantiation 할 수 있다. (※ width와 같은 parameter를 재설정하여 Instantiation을 하면 data size를 다르게 하여 Instantiation 할 수 있다.)

```
module counter(clk,clr,q);  
  
parameter width=16;  
input clk,clr;  
output reg [width-1:0] q;  
  
always @ (posedge clk)  
begin  
    if(clr)  
        q <= 0;  
    else  
        q <= q + 1;  
end  
  
endmodule
```

Parameter 2

- ❖ counter_top 모듈에서 counter 모듈을 2번 Instantiation하였다.
- ❖ Instantiation에서 #(16), #(.width(32))으로 설정한 부분이 있는데 괄호 안에 설정한 값이 Parameter의 값으로 전달된다.
- ❖ #(16)은 Parameter가 선언된 순서대로 값을 설정하는 방법이고 #(.width(32))은 Parameter로 선언한 이름과 일치하는 값을 설정하는 방법이다.
- ❖ 첫 번째로 Instantiation은 width가 16으로 설정된 Counter 모듈을 Instantiation하여 16-bit 카운터가 되고 두 번째로 Instantiation은 width가 32로 설정된 Counter 모듈을 Instantiation하여 32-bit 카운터가 된다.
- ❖ Chapter 3에서 언급한 것처럼 Parameter는 모듈이나 IP의 재사용을 위해 사용된다.

```
module counter_top(clk,clr,q1,q2);  
  
    input clk,clr;  
    output [15:0] q1;  
    output [31:0] q2;  
  
    counter #(16) cnt1(clk,clr,q1);  
    counter #(.width(32)) cnt2(clk,clr,q2);  
  
endmodule
```

Chapter 11 Verilog HDL Advanced Syntax

- Parameter
- Generate
- Subprogram

Generate

- ❖ Generate Statement는 Procedural Block 밖에 표현한 Concurrent Statement들을 하드웨어로 생성할지를 결정하는 Statement이다.
- ❖ Generate Statement에는 Generate If 와 Generate For Statement가 있다.
- ❖ Generate If는 If 뒤에 따라오는 식에 따라 생성할지 안 할지를 결정할 수 있는 Statement이고 Generate for는 뒤에 따라오는 식의 범위만큼 반복하여 여러 개를 생성할 수 있는 Statement이다.
- ❖ 오른쪽 코드 예제는 generate if와 generate for를 사용하여 shift register를 표현한 예제이다.

```
module shift_reg(clk,din,dout);  
  
    input clk,din;  
    output dout;  
    wire [14:0] inter;  
  
    genvar i;  
    generate  
        for(i=0;i<16;i=i+1) begin : genfor  
            if(i==0)           flipflop u0 (clk,din,inter[i]);  
            if(i>0 && i<15)  flipflop ux (clk,inter[i-1],inter[i]);  
            if(i==15)          flipflop u15 (clk,inter[i-1],dout);  
        end  
    endgenerate  
  
endmodule
```

Chapter 11 Verilog HDL Advanced Syntax

- Parameter
- Generate
- Subprogram

Function

- ❖ function keyword를 넣은 후 return_type 또는 return되는 값의 bit size를 넣고 function name을 코딩한다.
- ❖ function keyword와 begin keyword 사이는 Data Type Declaration을 하는 부분으로 input keyword를 사용해 Function에 입력되는 인수들과 Function 내부에서 사용할 Data Type들을 선언해 준다.
- ❖ begin으로 시작하고 Function에서 구현할 기능을 순차 구문들을 넣어서 표현한 후 function name에 특정 신호 값을 할당하여 그 값을 return한다.

```
function [15:0] multiply_func;  
    input [7:0] a,b;  
    reg [15:0] c;  
begin  
    c = a * b;  
    multiply_func = c;  
end  
endfunction
```

Task

- ❖ task keyword를 넣은 후 task이름을 쓰고 선언부에는 input, output, inout keyword를 사용해 task에 입출력되는 인수들과 Task 내부에서 사용할 Data Type들을 선언하는 부분이다.
- ❖ begin으로 시작하고 Task에서 구현할 기능을 순차 구문들을 넣어서 표현한다.
- ❖ a와 b로 들어오는 입력신호 2개를 곱하기 하여 출력신호 c에 인가하여 출력하는 Task이다.
- ❖ Function은 인수로 input만 선언 할 수 있지만 Task는 output도 선언할 수 있어서 output Port로 계산된 결과를 전달할 수 있다.

```
task multiply_task;  
    input [7:0] a,b;  
    output [15:0] c;  
begin  
    c = a * b;  
end  
endtask
```

Verilog Header

- ❖ Function과 Task는 Verilog 코드의 선언부에 넣어서 아래 구현부에서 Function과 Task를 불러서 사용할 수 있다.
- ❖ C에서 사용하는 것처럼 Function과 Task를 Header파일에 작성하고 이 Header파일을 Include하여 사용할 수 있다.
- ❖ 오른쪽 코드예제를 subprogram.vh 파일에 저장한다.

```
function [15:0] multiply_func;  
    input [7:0] a,b;  
    reg [15:0] c;  
begin  
    c = a * b;  
    multiply_func = c;  
end  
endfunction  
  
task multiply_task;  
    input [7:0] a,b;  
    output [15:0] c;  
begin  
    c = a * b;  
end  
endtask
```

Include

- ❖ 오른쪽 코드 예제와 같이 `include` Statement를 사용하여 Verilog Header파일을 불러와서 사용할 수 있다.

```
module subprogram_ex(a,b,c,result_func,result_task);
    input [7:0] a,b,c;
    output [15:0] result_func;
    output reg [15:0] result_task;

    `include "subprogram.vh"

    assign result_func = multiply_func(a,b);

    always @ (b,c)
    begin
        multiply_task(b,c,result_task);
    end
endmodule
```

Chapter 12 Projects

- **Stopwatch Project**
- **VGA Project**

Stopwatch Project

- ❖ EX7-2에서 Ultra96 보드에 Pmod96을 연결하고 2개의 Pmod SSD와 1개의 Pmod BTN을 연결하여 구현한 4개의 7-Segment에 분 초를 표시하는 시계에 세 가지 컨트롤 할 수 있는 신호를 추가하면 Stopwatch를 완성할 수 있다.
- ❖ Up/Down Port를 버튼 하나에 연결하여 한번 눌려지면 시계가 증가하고 다시 한번 눌려지면 시계의 카운트가 감소하도록 한다.
- ❖ Reset Port를 버튼 하나에 연결하여 버튼이 눌려지면 0분0초가 되도록 한다.
- ❖ Start/Stop Port를 버튼 하나에 연결하여 한번 눌려지면 시계의 카운트를 멈추고 다시 한번 눌려지면 시계의 카운트를 시작하도록 한다.
- ❖ 버튼을 눌렀다 뗄 때 상태가 한번만 변하도록 하기 위해 Chapter 10에서 구현한 FSM 회로를 활용할 수 있다.

Chapter 12 Projects

- Stopwatch Project
- VGA Project

VGA Project 1

- ❖ VGA는 영상신호 인터페이스를 위한 기존 아날로그 방식 중의 하나이다.
- ❖ VESA가 VGA 영상의 국제 표준을 만들어서 모니터에 원하는 화면이 나오도록 하려면 이 VESA에서 만든 표준에 따라서 영상신호를 출력해야 한다.
- ❖ 인터넷을 통해 VESA Monitor Timing Standard를 검색하면 이 표준 문서를 쉽게 구할 수 있다.
- ❖ 이 문서에는 다양한 해상도에 대한 표준을 가지고 있는데 이 중에서 640 X 480 60Hz를 사용하면 Pixel Clock을 대략 25MHz정도의 Clock을 사용하면 정상 동작한다.
- ❖ Pixel Clock은 화면에 한 개의 점이 표시되는 속도를 의미하는 것으로 Pixel Clock이 25MHz라는 것은 40ns에 한 번씩 화면에 점 하나를 표시한다는 의미이다.
- ❖ 화면은 좌에서 우로 수평해상도(640)만큼 한 라인이 표시되고 이 라인이 수직해상도(480) 만큼 표시는 것을 한 프레임이라고 부르고 한 프레임을 1초에 60번 표시하는 것이 60Hz이다.
- ❖ 실제 해상도는 640 X 480이지만 이 해상도는 실제 화면에 Pixel이 표시되는 부분에 대한 해상도이고 실제 타이밍은 이 해상도의 앞 뒤로 Porch와 Sync Timing이 포함되어 Porch와 Sync를 포함한 전체 해상도는 800 X 525가 된다.
- ❖ Pixel Clock은 $1s / 60 / 525 / 800 = 39.68\text{ns}$ 로 계산된다. (※ 표준 문서에는 39.7ns / 25.175MHz로 기재되어 있다.)

VGA Project 2

- ❖ Clocking Wizard를 사용하여 Pixel Clock(pclk)인 25.175MHz 클럭 신호를 만들고 오른쪽 코드 예제와 같이 코딩을 한다.
- ❖ Horizontal Sync Timing과 Vertical Sync Timing 맞추어 HS와 VS신호를 출력하고 있다.
- ❖ Porch를 계산하여 hcnt와 vcnt가 원하는 카운트 값이 되었을 때 R, G, B출력이 나가도록 추가해주면 RGB에 출력되는 색깔이 계산된 좌표에 찍히도록 할 수 있다.
- ❖ Pmod VGA를 PMOD_A와 PMOD_B에 연결한 후 모니터에 점 또는 라인이 표시되도록 여러 가지 시도를 해본 후 Stopwatch에서 구현한 시간이 모니터에 표시되도록 구현해보자.

```
always @ (posedge pclk)
begin
    hcnt <= hcnt + 1;
    if(hcnt == 800) begin
        hcnt <= 0;
        vcnt <= vcnt + 1;
        if(vcnt == 525) begin
            vcnt <= 0;
        end
    end

    if(hcnt>=0 && hcnt<96)
        vga_hs <= 1'b0;
    else
        vga_hs <= 1'b1;

    if(vcnt>=0 && vcnt<2)
        vga_vs <= 1'b0;
    else
        vga_vs <= 1'b1;
end
```