

# VHDL Design using Vivado

**Tool : Xilinx Vivado 2019.2**

**Kit : Ultra96 Training Kit**

# Chapter 1 Introduction

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

# Introduction to VHDL

- ❖ HDL(Hardware Description Language)은 하드웨어(디지털 논리 회로)를 묘사하는 언어라는 뜻이다.
- ❖ 현재 사용하고 있는 HDL은 VHDL과 Verilog HDL이 있으며 이 중에서 VHDL에 대해서 알아본다.
- ❖ VHDL의 V는 VHSIC (Very High Speed Integrated Circuit)의 약자이고 HDL은 Hardware Description Language의 약자이다. (※ VHDL은 약자 안에 약자가 들어 있는 구조의 이름이다.)
- ❖ VHDL은 미국 국방성에서 주문형 집적회로(ASIC)의 문서화에 사용하기 위해 개발된 언어이다. (※ 복잡한 매뉴얼로 회로의 동작 내용을 설명하는 대신, 회로의 동작 내용을 문서화하여 설명하기 위해 개발되었다.)
- ❖ 이런 문서화된 언어를 회로 디자인 과정에서 시뮬레이션에 사용하게 되었고 VHDL 파일을 읽어 들여서 논리 합성을 한 다음 실제 회로 형태를 출력하는 기능을 덧붙이도록 발전되었다.
- ❖ 오늘날에는 디지털 회로의 설계, 검증, 구현 등의 모든 용도로 사용하고 있다.
- ❖ VHDL은 에이다 프로그래밍 언어의 부분집합에 디지털 회로에 필수적인 시간 개념을 추가하는 방식으로 만들어졌으나 IEEE 표준화 작업을 거치면서 오늘날과 같은 형태와 문법을 가지게 되었다.
- ❖ VHDL은 가독성(Readability)이 우수하다는 평가를 받고 있다.
- ❖ 처음에 미국 국방부를 중심으로 많이 사용해서 VHDL은 주로 공기업에 많은 사용자가 분포하게 되었고 이에 대한 영향으로 우리나라에서도 공기업과 공기업의 협력업체들이 주로 사용하고 있다.

# Chapter 1 Introduction

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

# Introduction to Xilinx

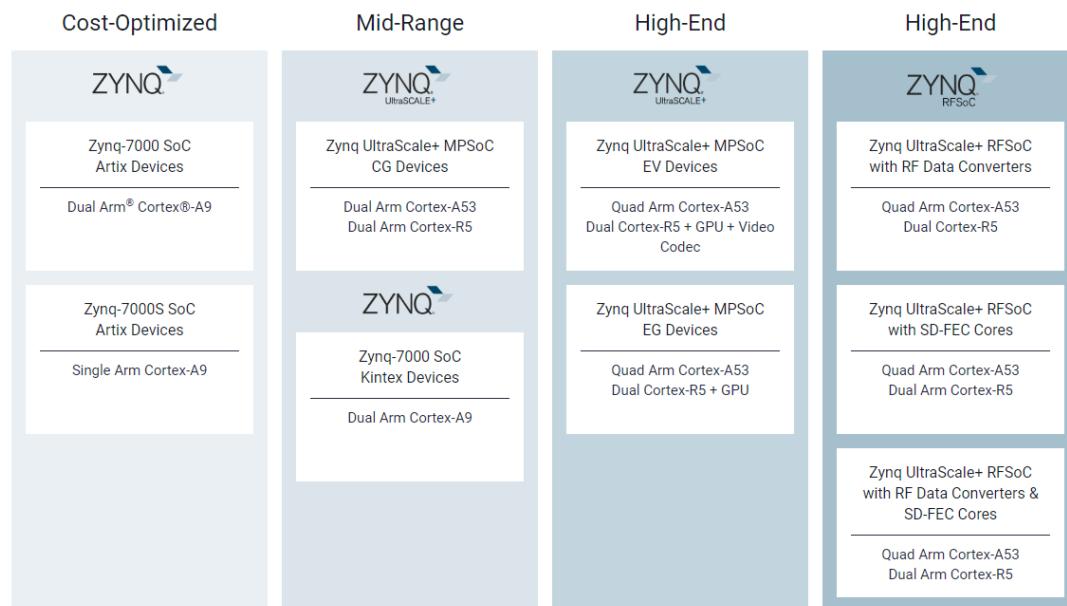
- ❖ 자일링스는 인텔과 함께 전세계 FPGA 시장을 양분하고 있으며 현재 이 두 회사가 FPGA 시장의 90% 이상을 점유하고 있다.
- ❖ 인텔은 2015년에 알테라를 인수하면서 FPGA 벤더 중 하나가 되었으며 Lattice Semiconductor 와 Microsemi가 특정 분야에서는 두각을 나타내어 일정 점유율을 차지하고 있다.
- ❖ 자일링스는 칩을 설계하는 팹리스(Fabless) 회사로 분류된다. 여기서 Fab은 제조 설비를 의미하는 fabrication의 줄임말이며 Fabless는 fabrication과 less를 합친 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



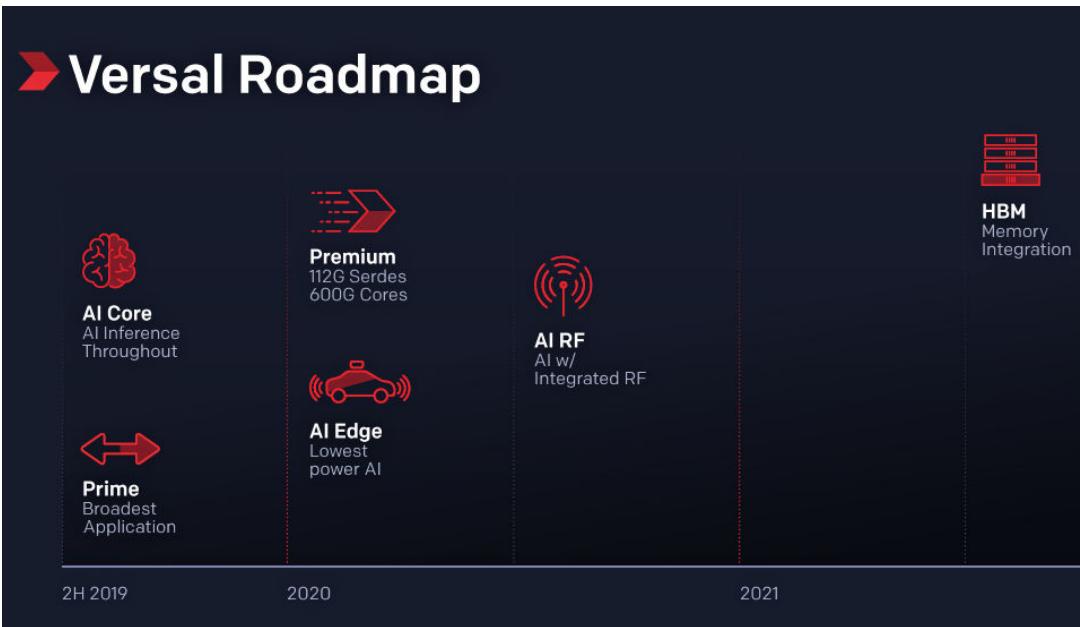
- ❖ 위 그림에서 위에 있는 45nm 와 같은 숫자는 회로선 폭의 크기이며 반도체를 만드는 공정의 정밀도를 의미한다.
- ❖ 자일링스의 최신 FPGA는 16nm 공정으로 생산하는 UltraScale+이다.
- ❖ FPGA 시리즈는 저가형 중간형 고가형 디바이스 패밀리로 나누어지는데 위 그림에서 아래에 위치한 디바이스가 저가형이고 위로 올라갈수록 고가형의 디바이스이다.

# 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 VHDL과 Verilog와 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 VHDL을 사용하여 RTL설계를 하는 부분에 대해서 다룰 예정이니 Vivado를 사용한다.

# Chapter 1 Introduction

- Introduction to VHDL
- 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 vhdl\_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 여러 가지 실습들을 통해 VHDL을 이용하여 디지털 시스템을 설계하고 검증하는 내용을 다룬다.

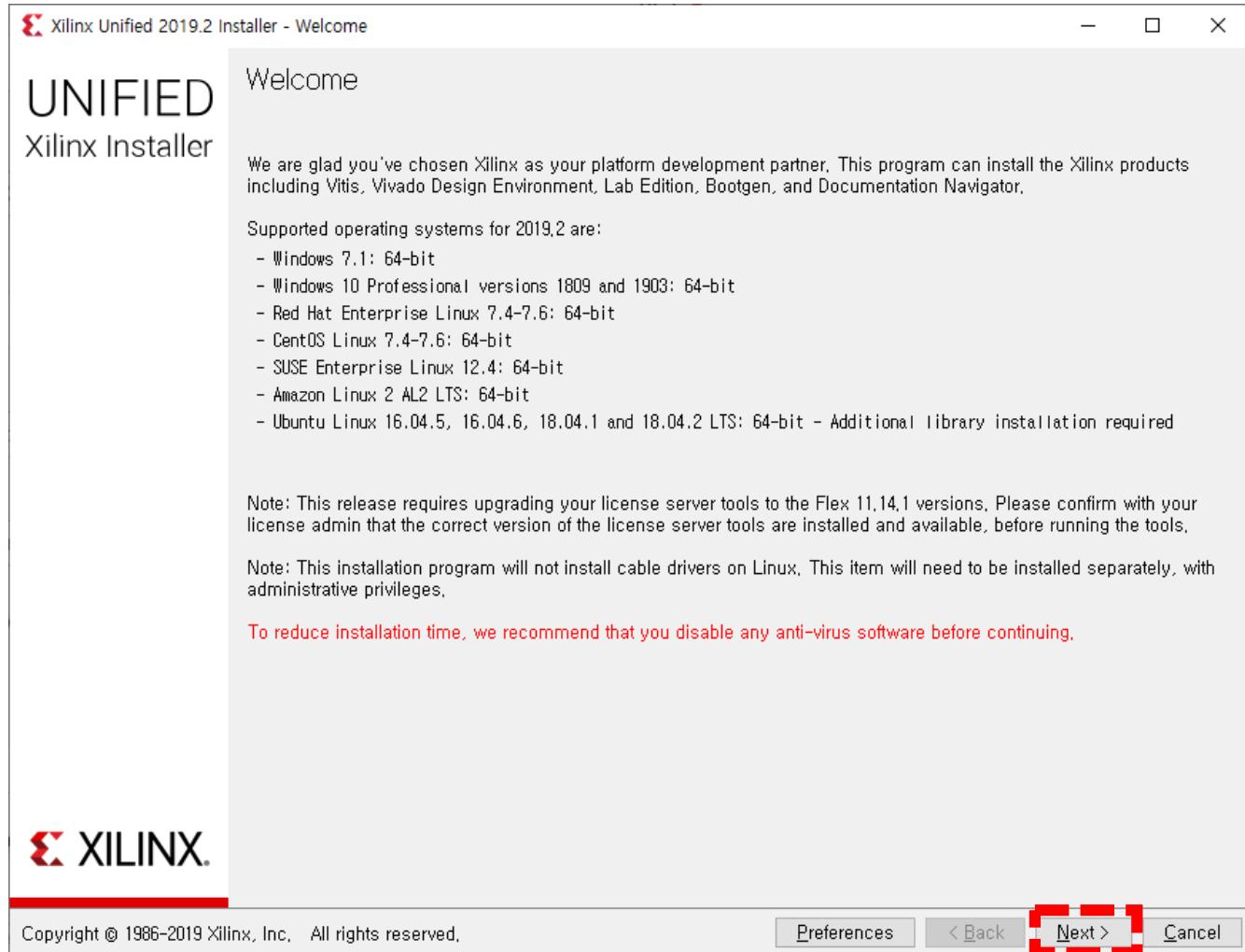
# Chapter 1 Introduction

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

# Step 1 Download Vivado

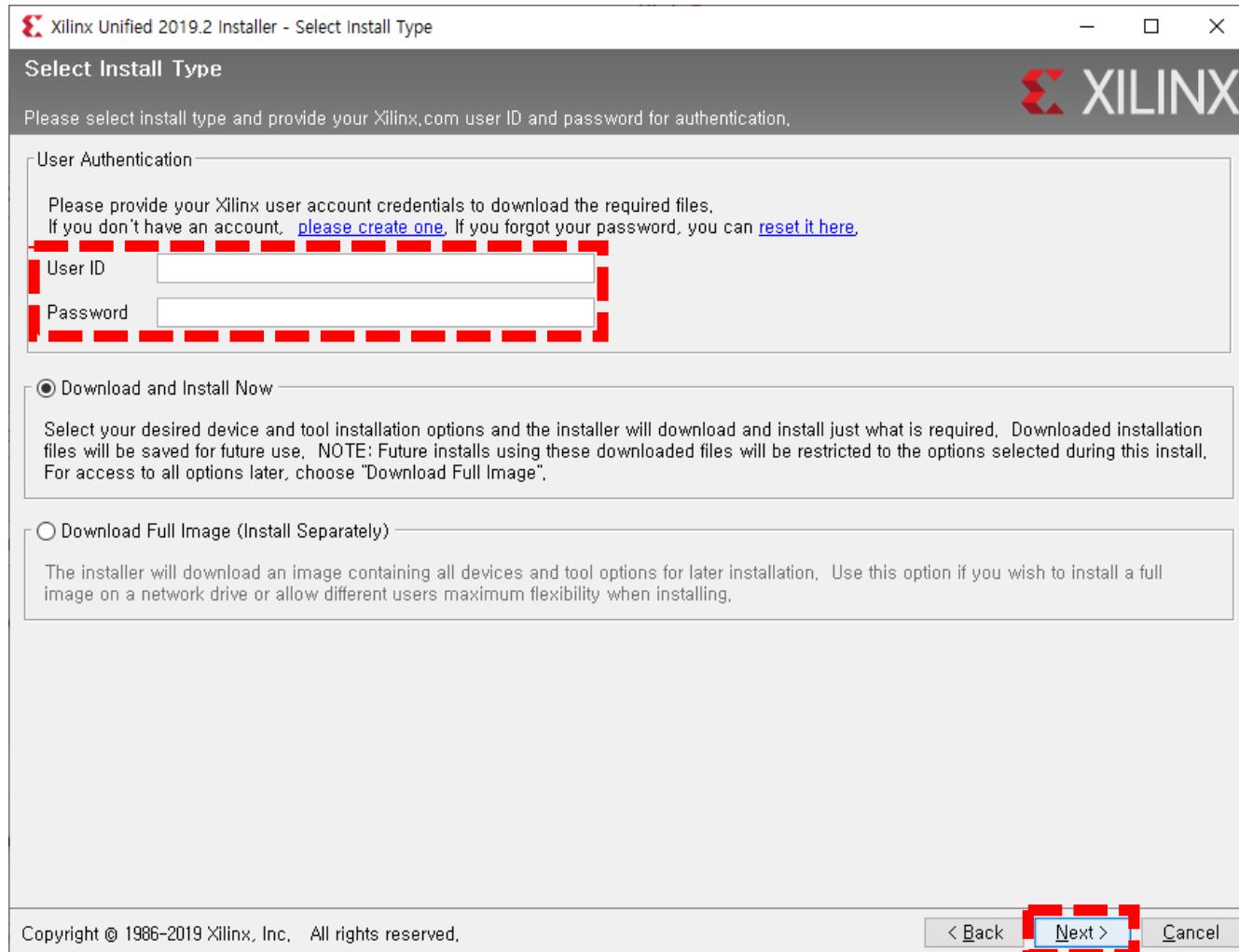
- ❖ 자일링스 홈페이지([www.xilinx.com](http://www.xilinx.com))에 접속한다.
- ❖ SUPPORT 메뉴 아래 Downloads & Licensing 메뉴를 클릭한다.
- ❖ Web Installer를 사용하는 방식과 Single File을 다운로드하여 설치하는 방식이 있는데 Web Installer방식은 작은 사이즈의 Web Installer를 다운로드 받아서 설치할 때 인터넷을 통해 필요한 파일을 다운로드 받아서 설치하는 방식이고 Single File 방식은 설치에 필요한 전체 파일을 다운로드 받아서 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



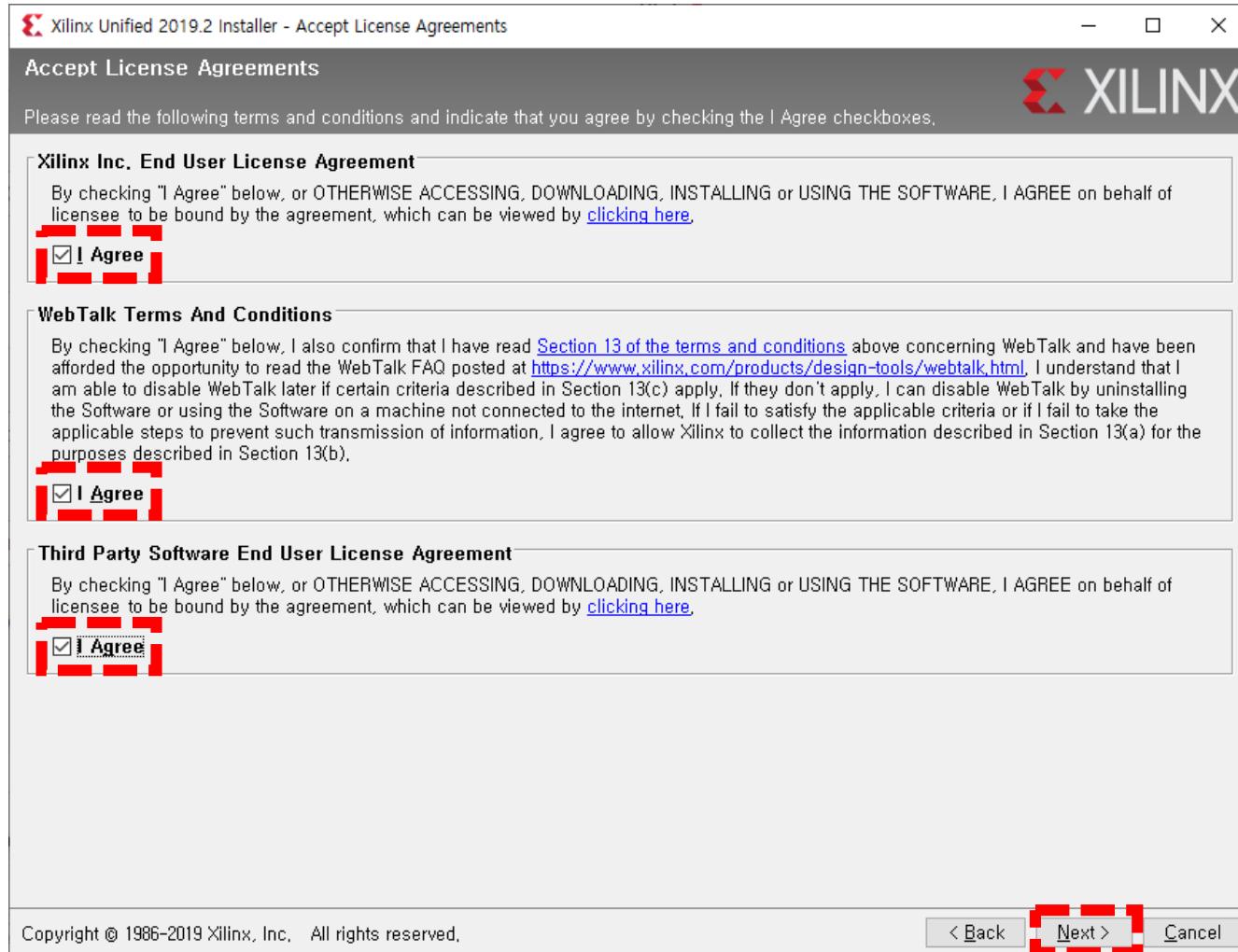
- ❖ 다운로드가 완료되면 다운로드한 설치 파일을 더블 클릭하여 실행한다.
- ❖ 설치 초기화면이 나오면 Next 버튼을 클릭한다.

# Step 2 Vivado Installation 2



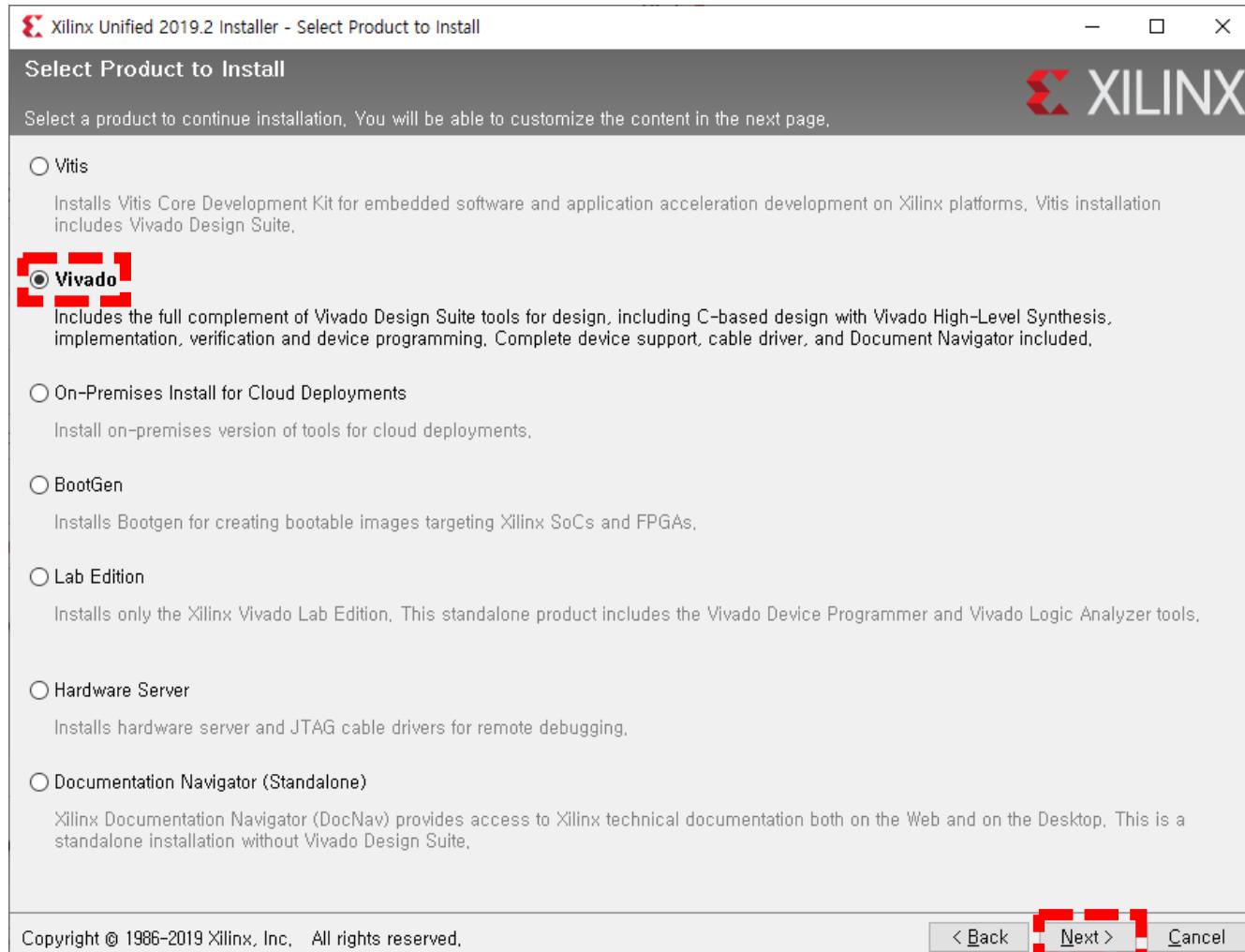
❖ 자일링스 유저임을 확인하는  
창이 나오면 자일링스 웹페이지에  
가입한 ID와 Password를  
입력한 후 Next 버튼을 클릭한  
다.

# Step 2 Vivado Installation 3



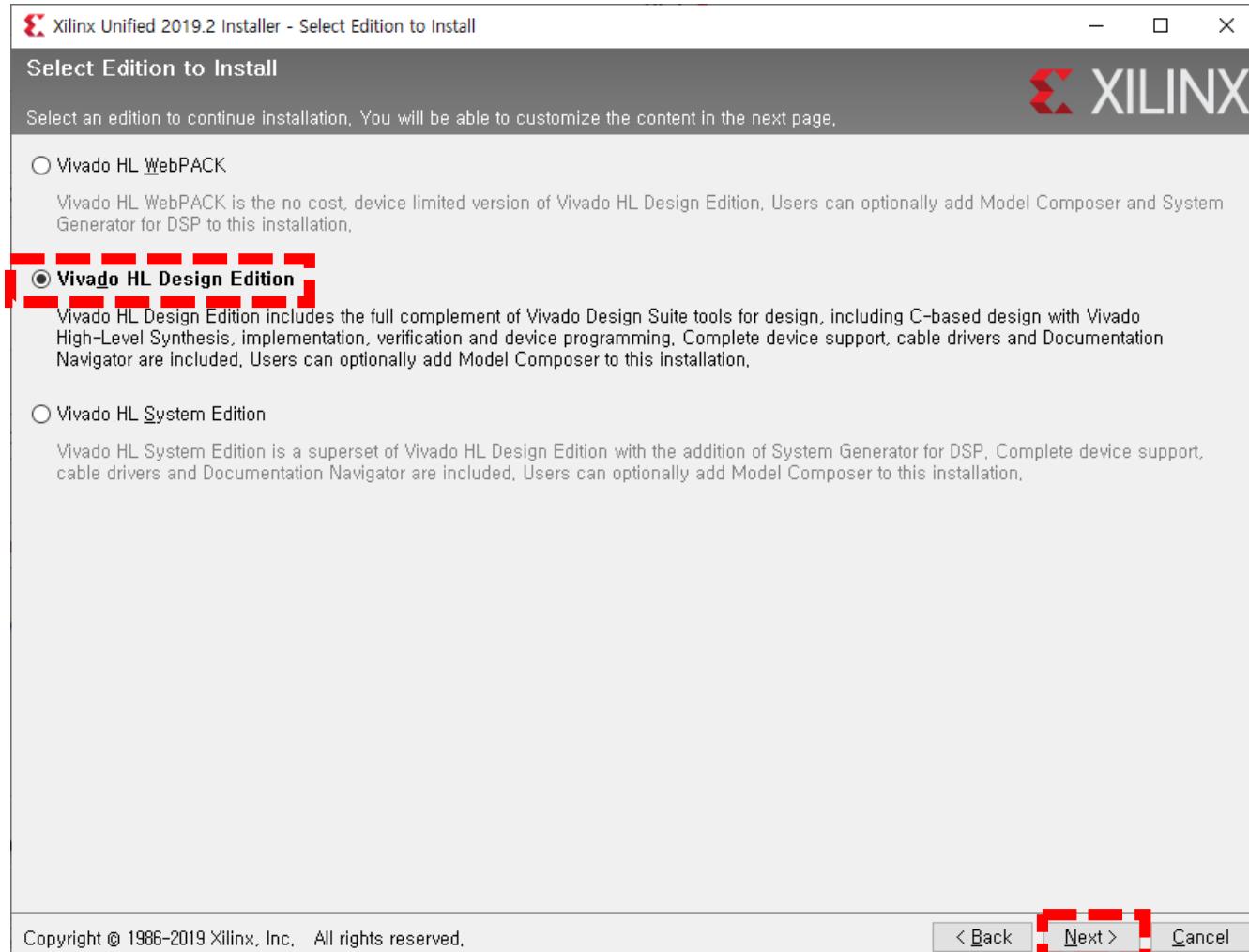
❖ I Agree 체크박스를 모두 선택  
후 Next 버튼을 클릭한다.

# Step 2 Vivado Installation 4



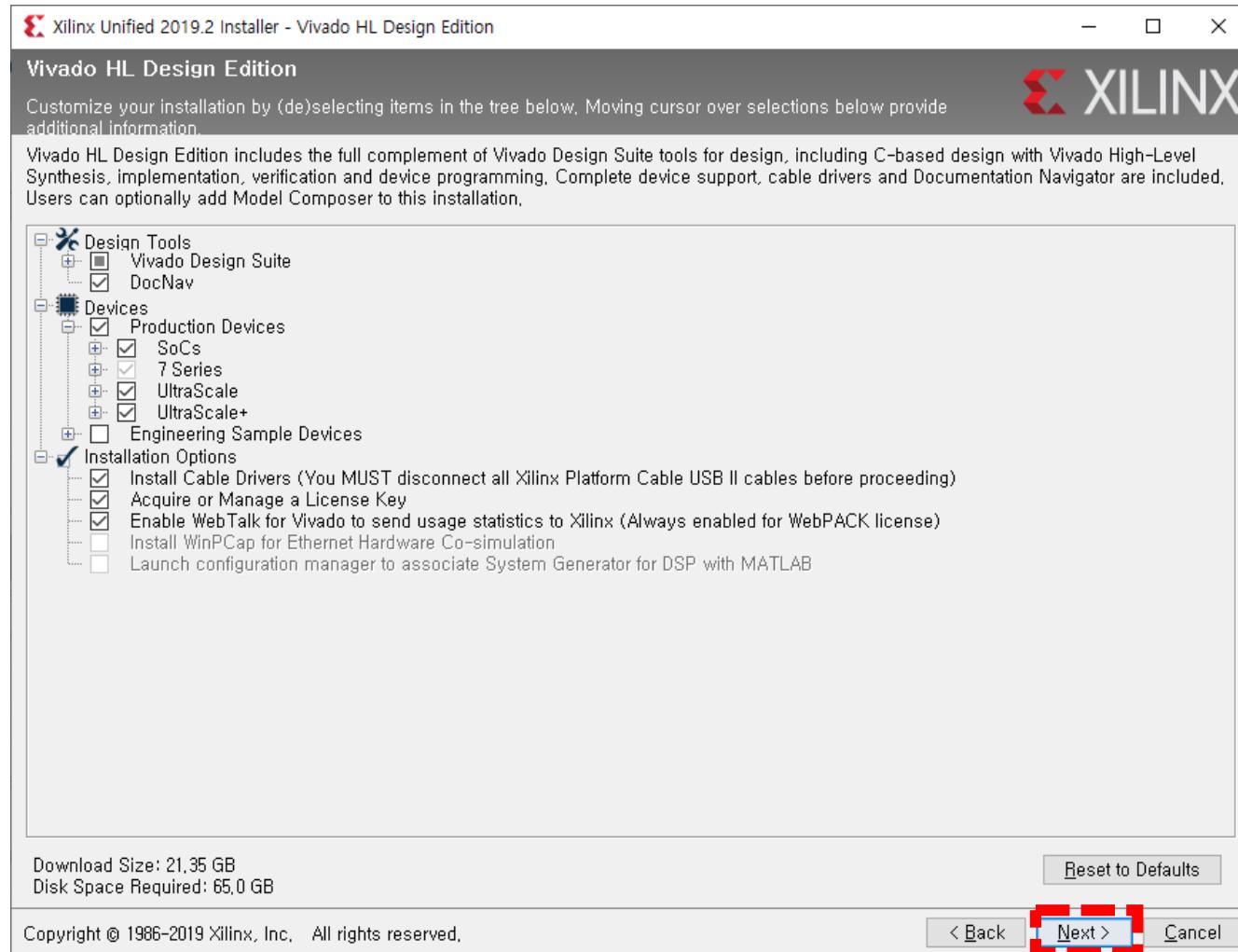
❖ 어떤 툴을 설치 할건지 선택하는 화면이 나오면 Vivado를 선택한 후 Next버튼을 클릭한다.

# Step 2 Vivado Installation 5



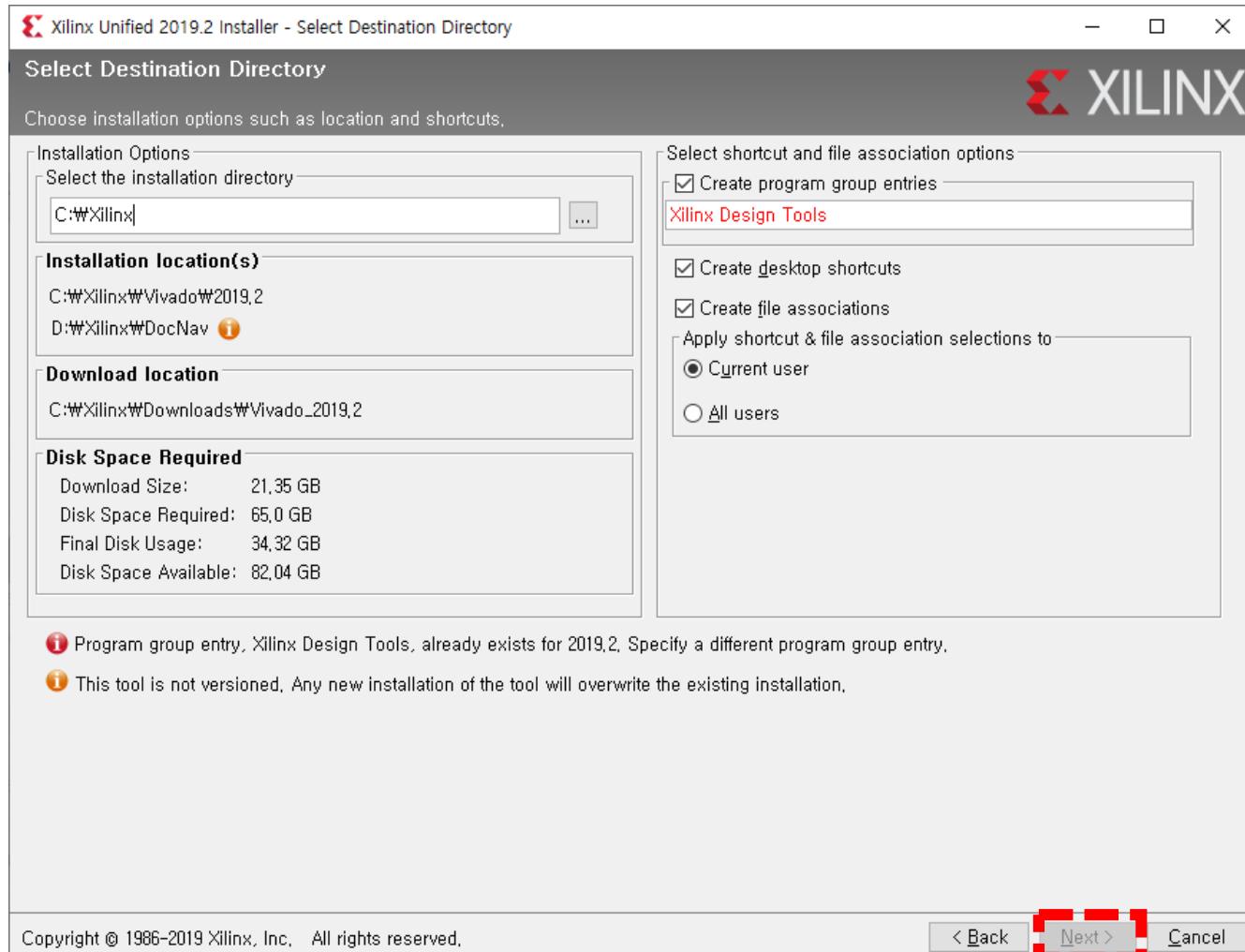
- ❖ 어떤 에디션을 설치 할건지 선택하는 화면이 나오면 Vivado HL Design Edition을 선택한 후 Next 버튼을 클릭한다.
- ❖ Vivado HL Webpack은 무료 버전이고 Vivado HL Design Edition과 System Edition은 유료 버전으로 자일링스로부터 라이선스를 구입하면 Xilinx Product Licensing Site(<https://xilinx.com/getlicense>)를 통해 라이선스를 발급받아서 사용할 수 있다.
- ❖ 본 교육에서는 Ultra96 보드에 포함된 바우처를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

# Step 2 Vivado Installation 6



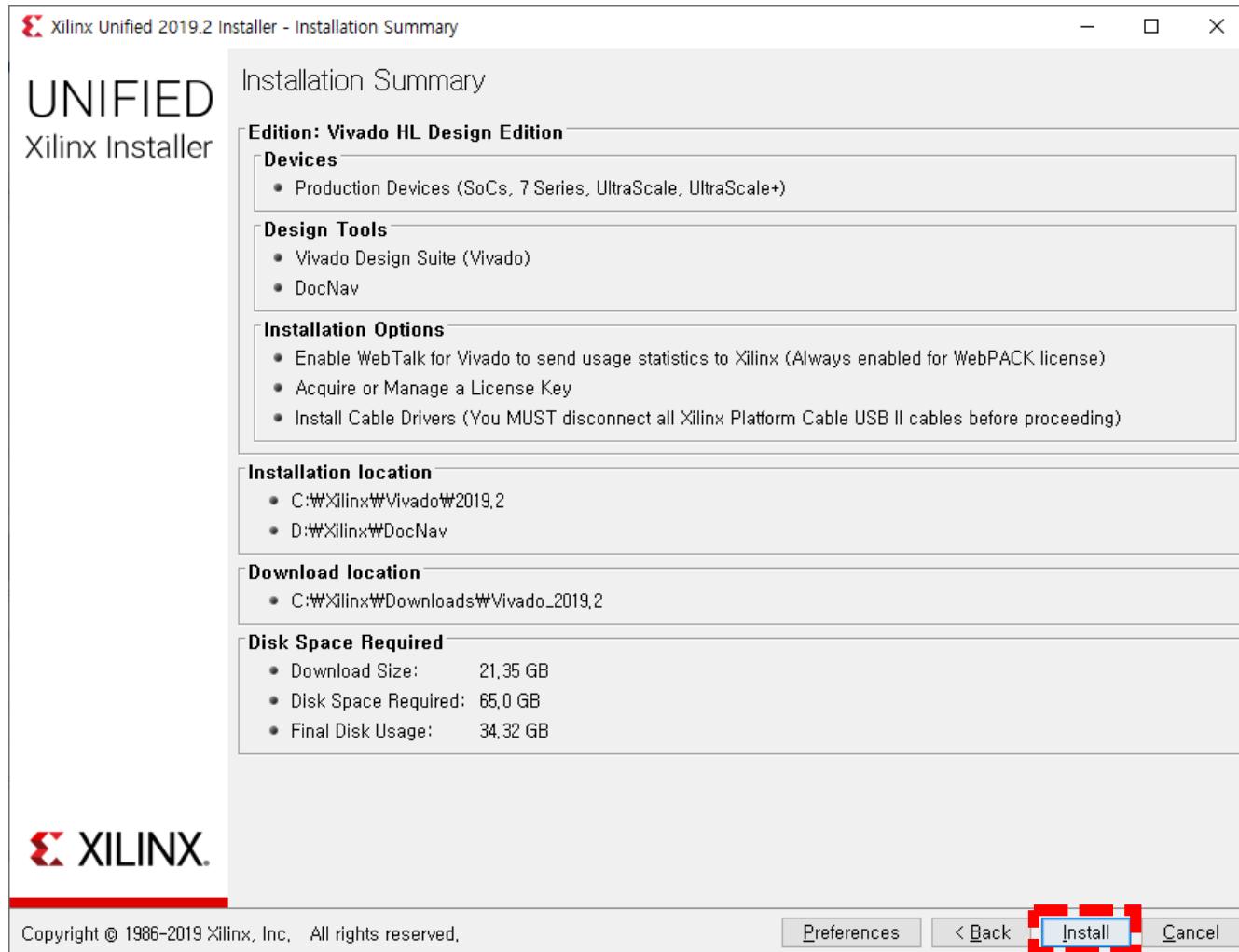
❖ 설치할 툴을 선택하는 화면이 나오면 디폴트 상태 그대로 두고 Next 버튼을 클릭한다.

# Step 2 Vivado Installation 7



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

# Step 2 Vivado Installation 8

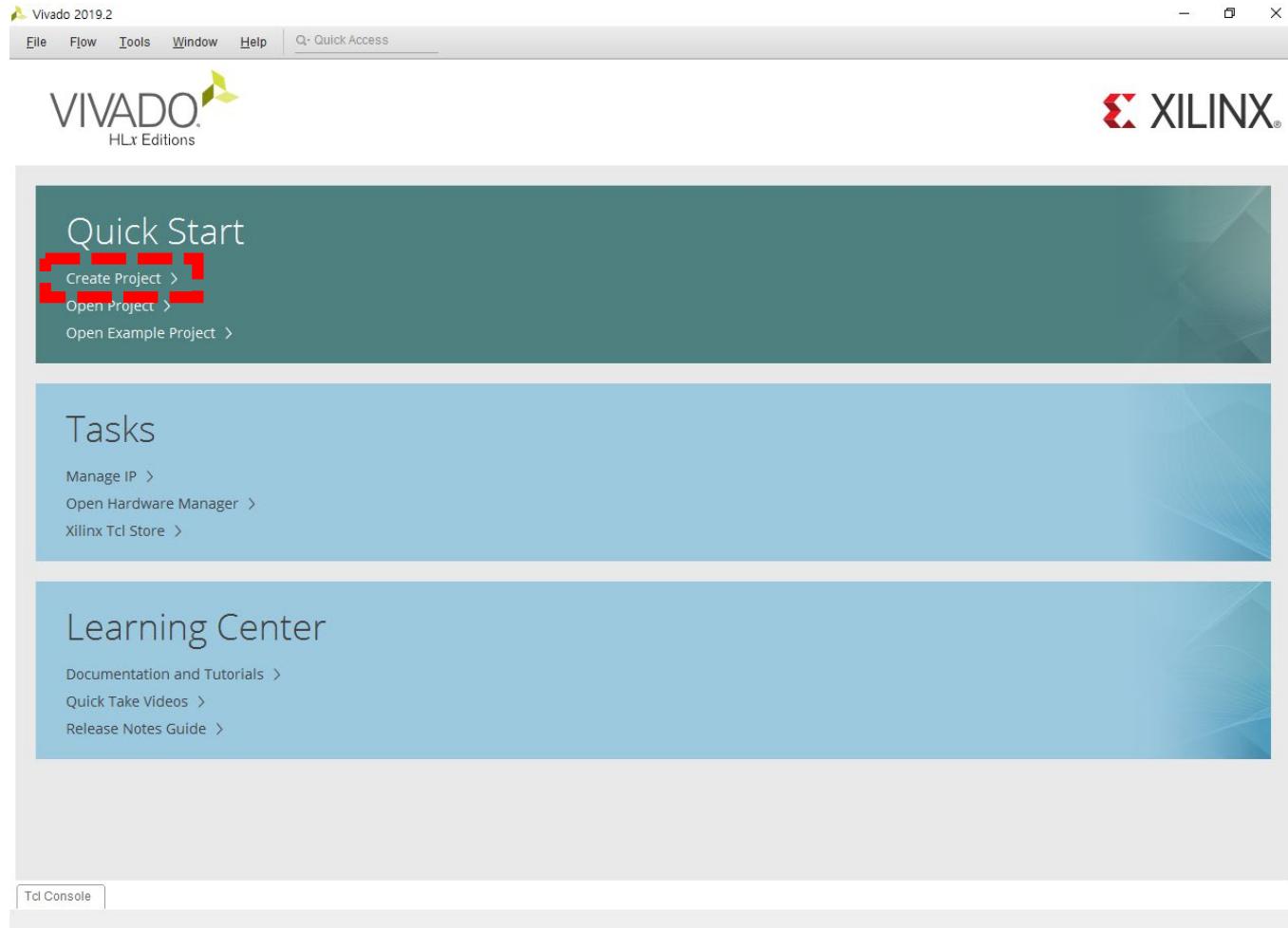


❖ 설치할 내용을 종합한 화면이다. 내용을 살펴보고 특이점이 없으면 Install 버튼을 클릭하여 설치를 시작한다. 설치시간은 컴퓨터 성능과 인터넷 속도에 따라 다르지만 일반적으로 30분에서 1시간정도 소요된다.

# Chapter 1 Introduction

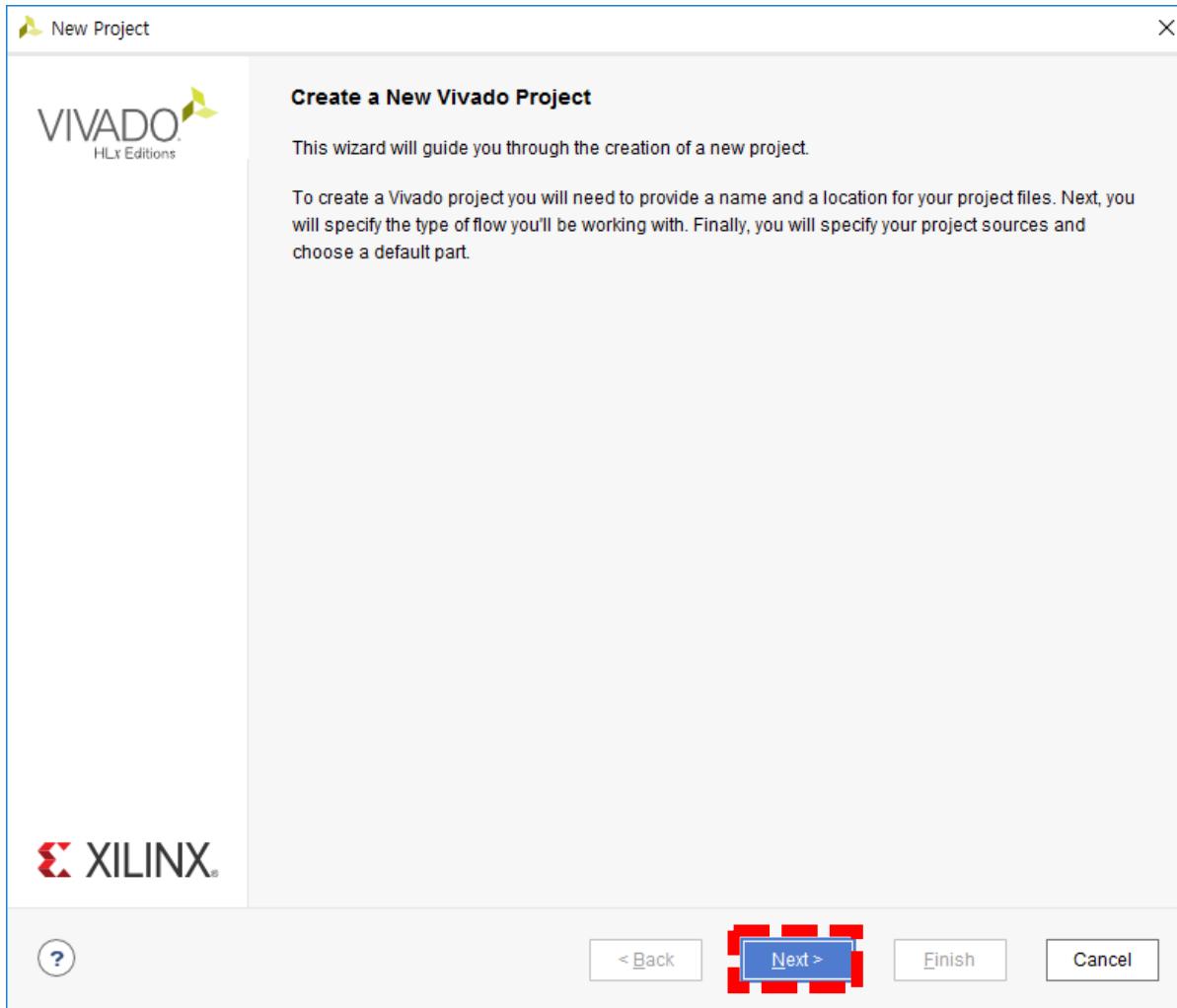
- Introduction to VHDL
- Introduction to Xilinx Devices and Tools
- Introduction to Inipro and Ultra96 Training Kit
- Vivado Installation
- Creating Vivado Project

# Step 1 Creating Vivado Project 1



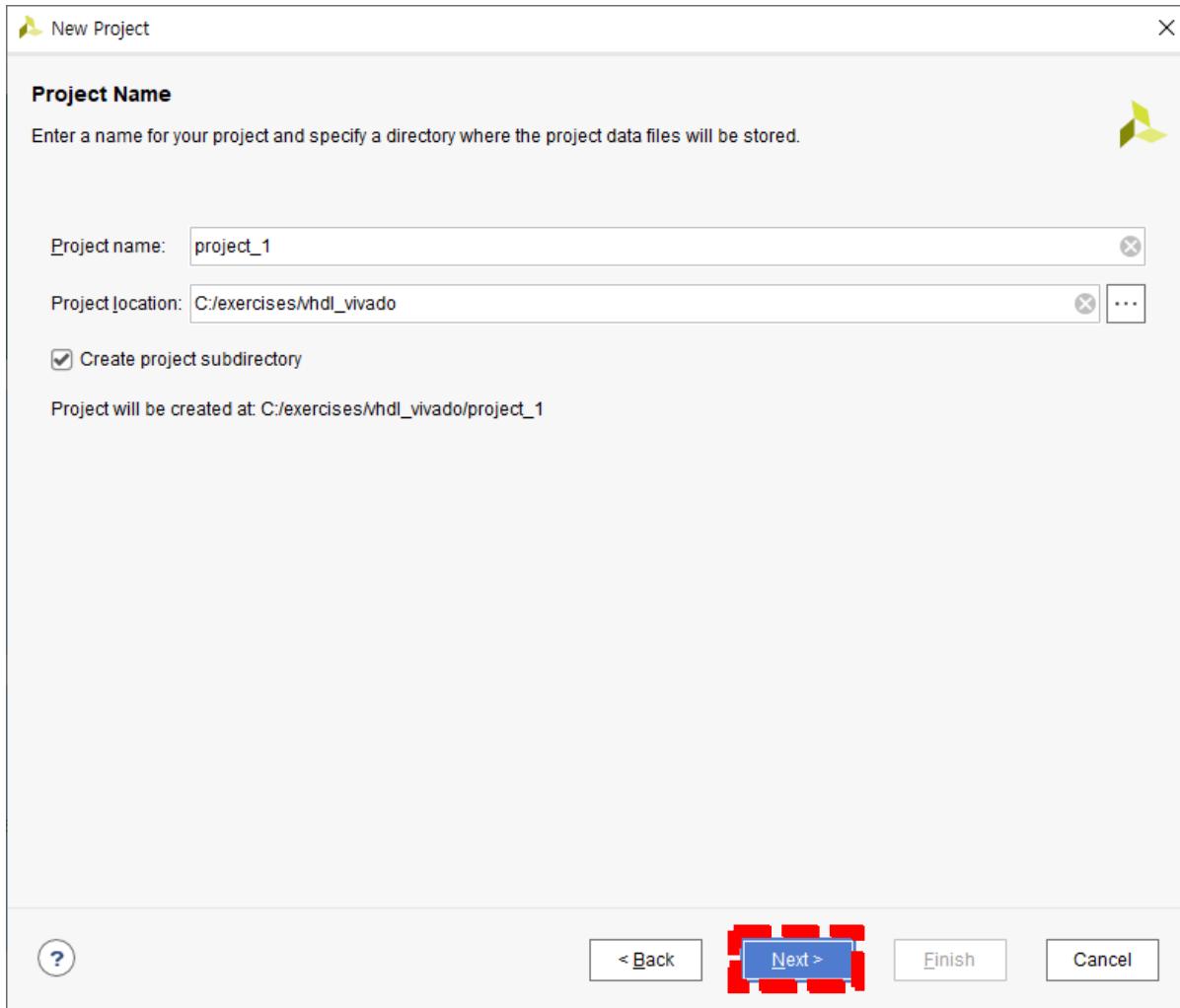
- ❖ 바탕화면 또는 시작화면에서 Vivado를 클릭하여 Vivado를 실행한다.
- ❖ Vivado 실행 초기화면에서 Quick Start 아래 Create Project를 클릭한다.

# Step 1 Creating Vivado Project 2



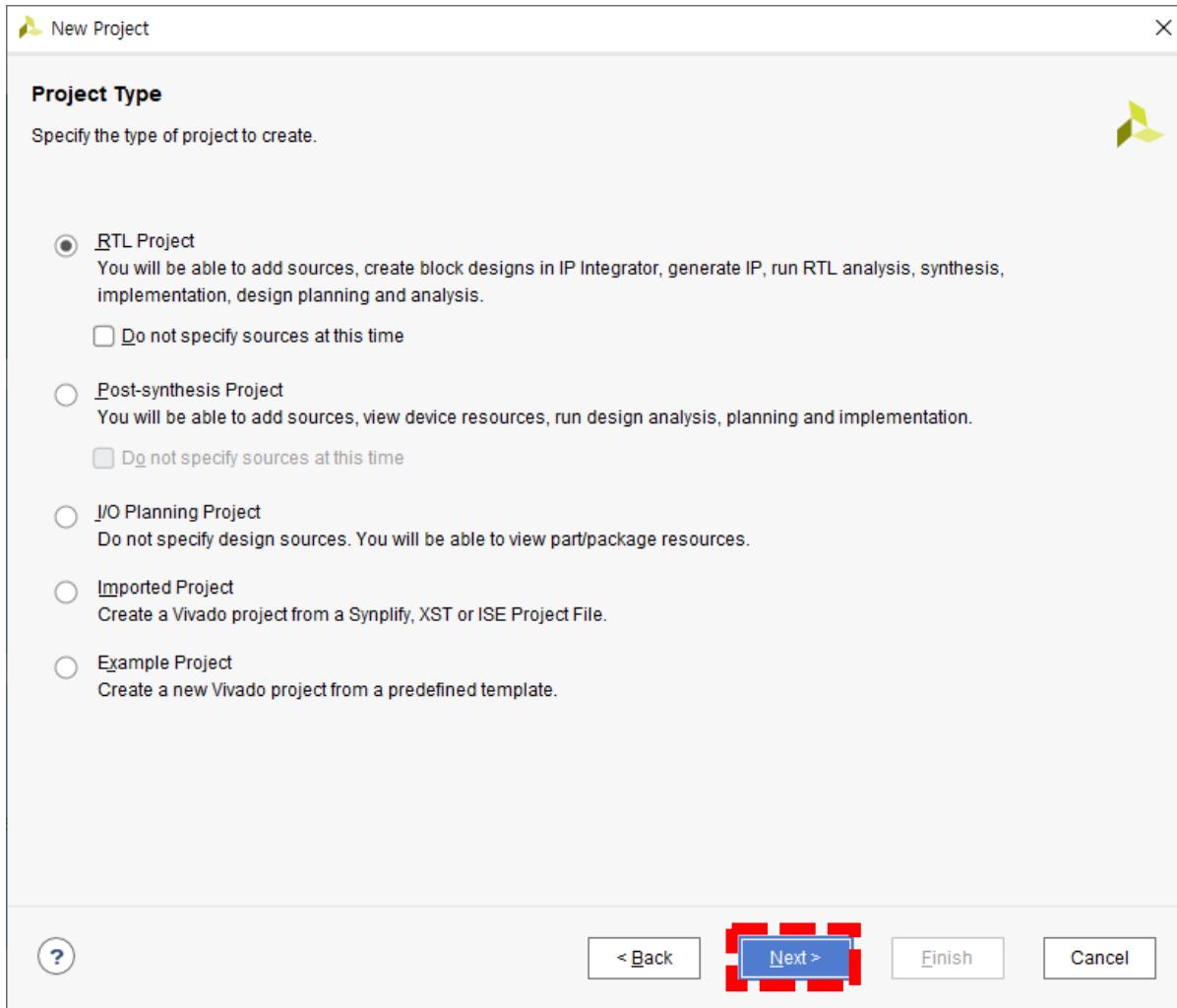
❖ 프로젝트 생성을 위한 초기화  
면이 나오면 Next 버튼을 클릭  
한다.

# Step 1 Creating Vivado Project 3



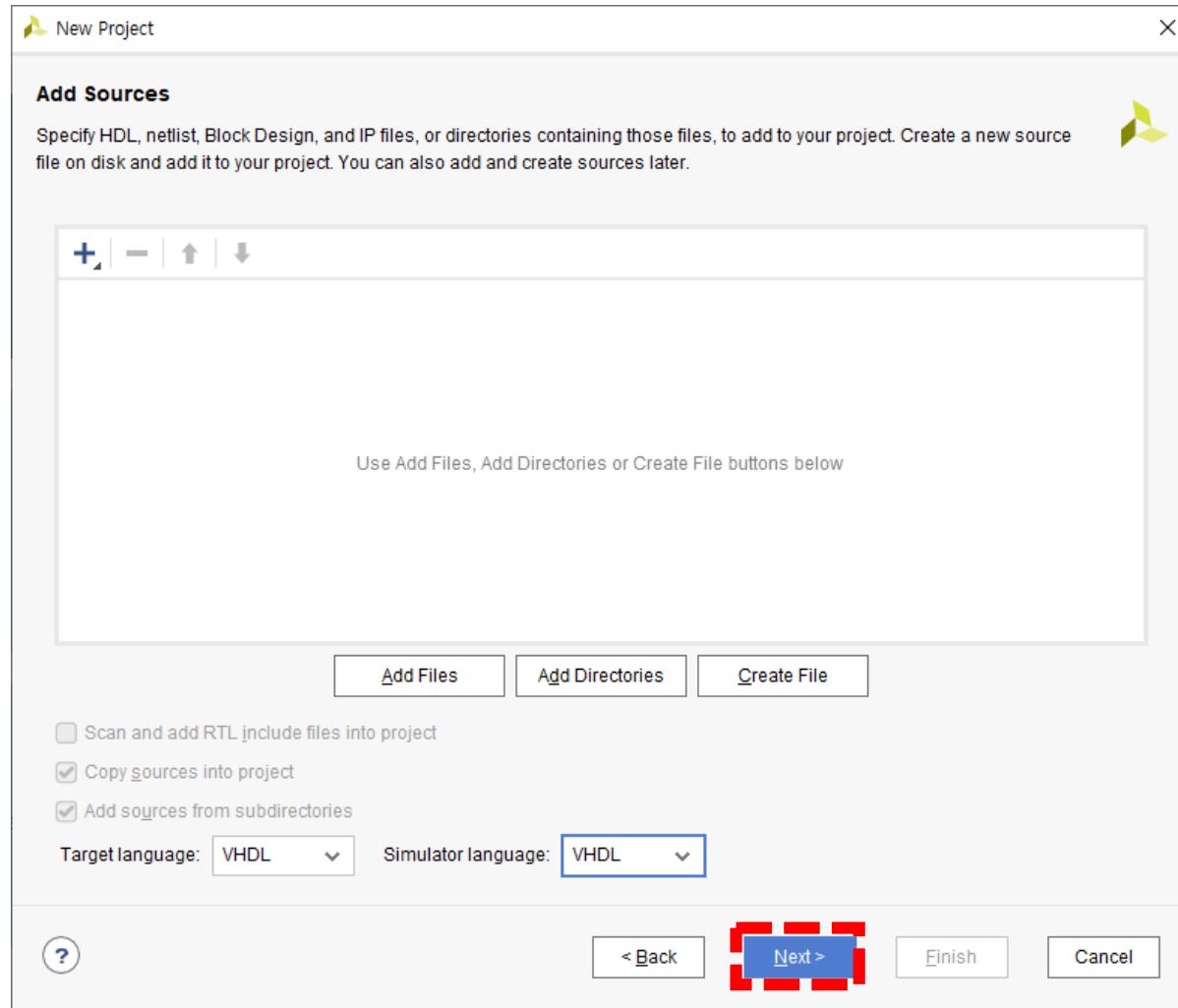
- ❖ 프로젝트 이름과 경로를 선택하는 화면이 나오면 본인이 사용할 프로젝트 경로를 선택하고, 프로젝트 이름은 디폴트 그대로 project\_1을 사용한다.
- ❖ 폴더가 생성되어 있지 않으면 폴더를 생성해야 하므로 Create project subdirectory 체크박스를 체크하고, Next 버튼을 클릭한다.

# Step 1 Creating Vivado Project 4



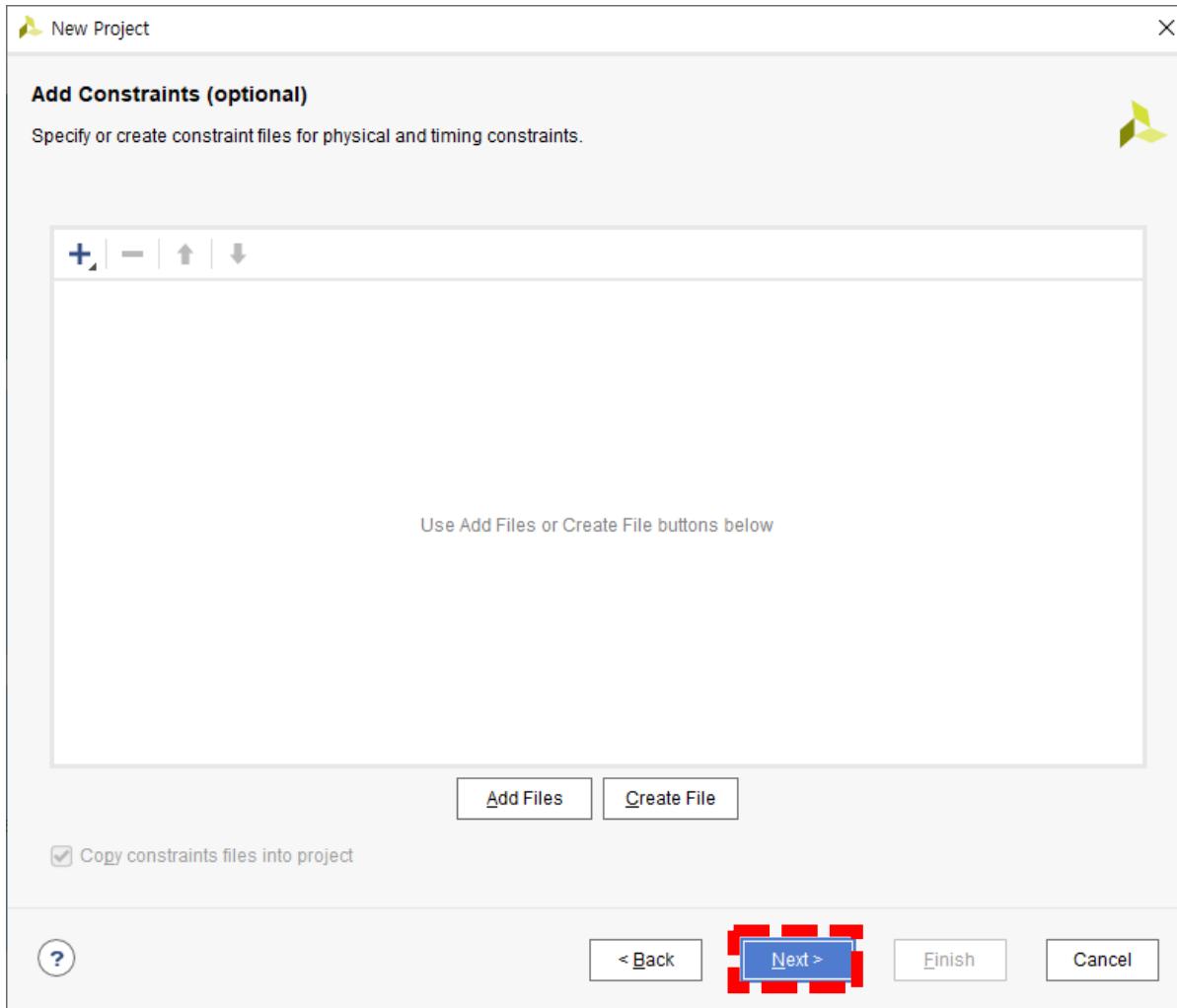
- ❖ 프로젝트 타입을 선택하는 화면이 나오면 RTL 설계에 대한 내용을 배울 예정이니 RTL Project를 선택한 후 Next 버튼을 클릭 한다.
- ❖ Do not specify sources at this time 옵션은 소스파일없이 프로젝트를 생성하는 옵션인데 프로젝트 생성단계에서 소스파일을 추가하는 메뉴를 살펴보기 위해서 여기서는 체크를 해제하고 진행하도록 한다.

# Step 1 Creating Vivado Project 5



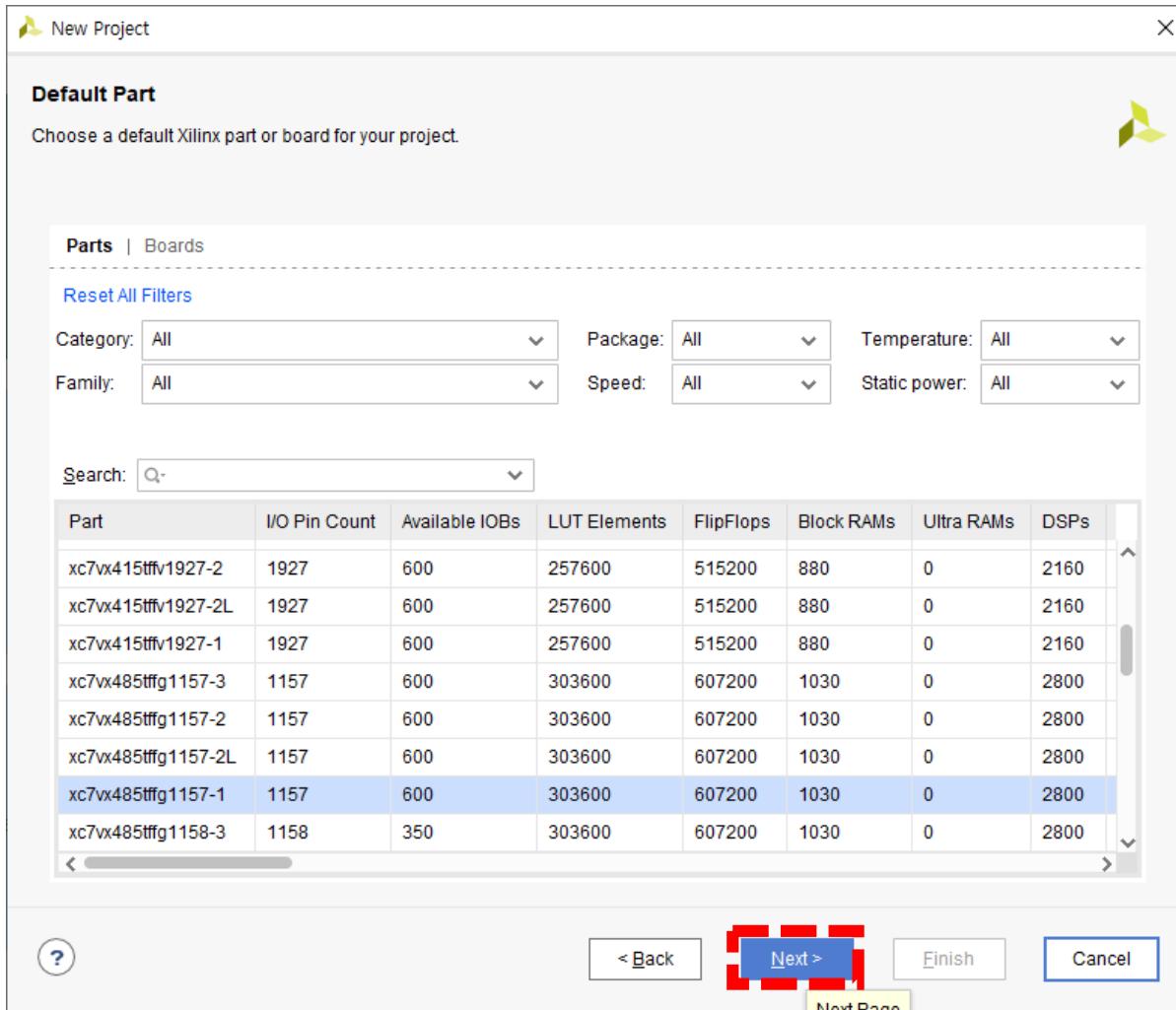
- ❖ 기존에 작성된 소스파일을 프로젝트에 추가하는 화면이 나오면 추가할 소스파일이 없으니 넘어간다.
- ❖ Target language와 Simulator language 옵션에서 그림과 같이 VHDL을 선택하고 Next 버튼을 클릭한다.

# Step 1 Creating Vivado Project 6



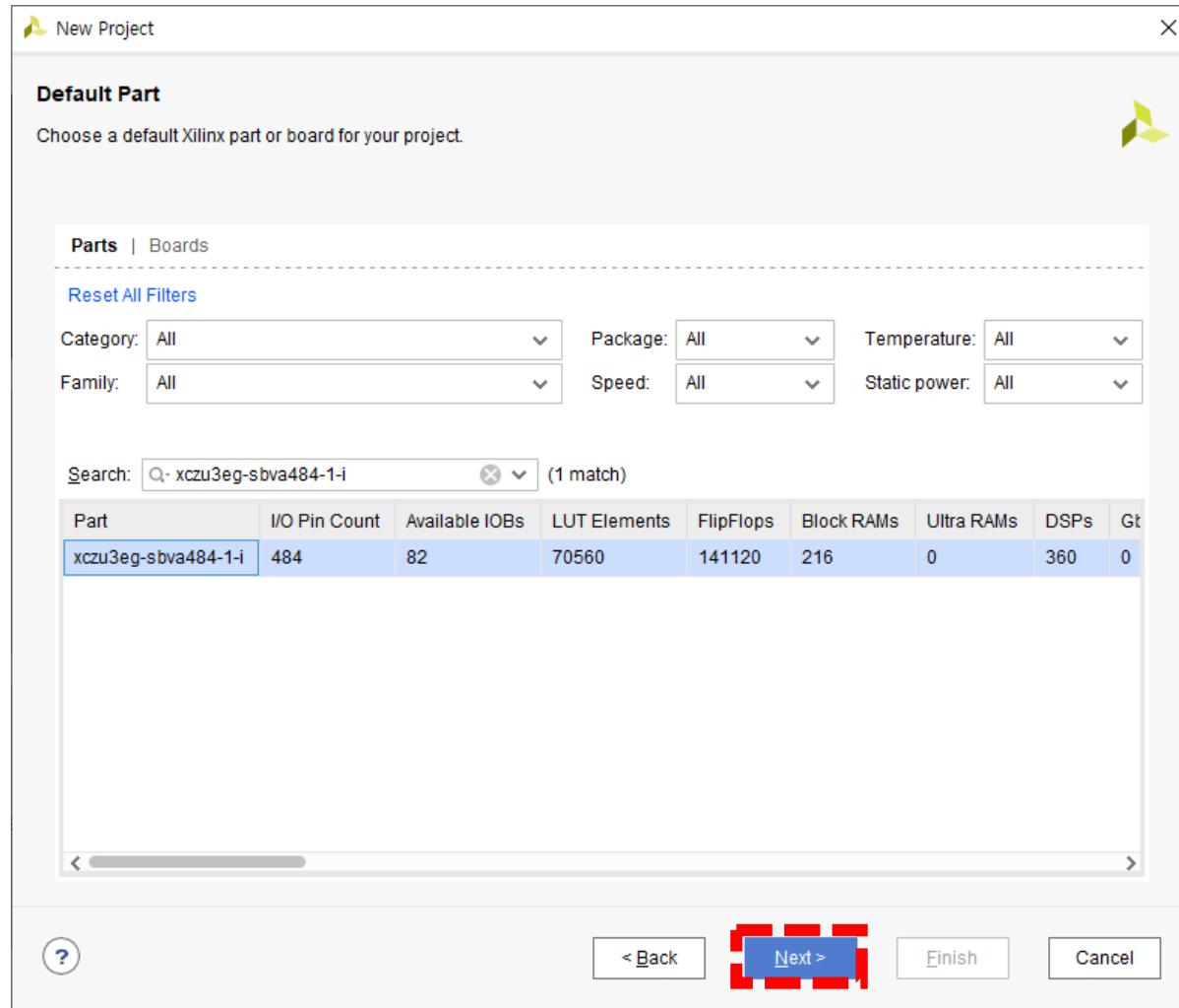
- ❖ Constraints를 추가하는 화면이 나온다.
- ❖ Constraints는 사전적 의미로 제한하다 라는 말로써 FPGA 설계에 필요한 제한조건을 만들기 위한 것이다.
- ❖ 예를 들어, 가장 기본적인 Constraints에는 Pin Constraints가 있는데 Pin Constraints란 FPGA Device에 HDL을 사용하여 설계한 하드웨어의 입출력 포트가 FPGA Device의 어떤 Pin을 사용할지 지정하는 것이다.
- ❖ 본 교육에서는 Ultra96 보드를 사용할 예정이니 Ultra96 보드에 맞게 Pin Constraints를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

# Step 1 Creating Vivado Project 7



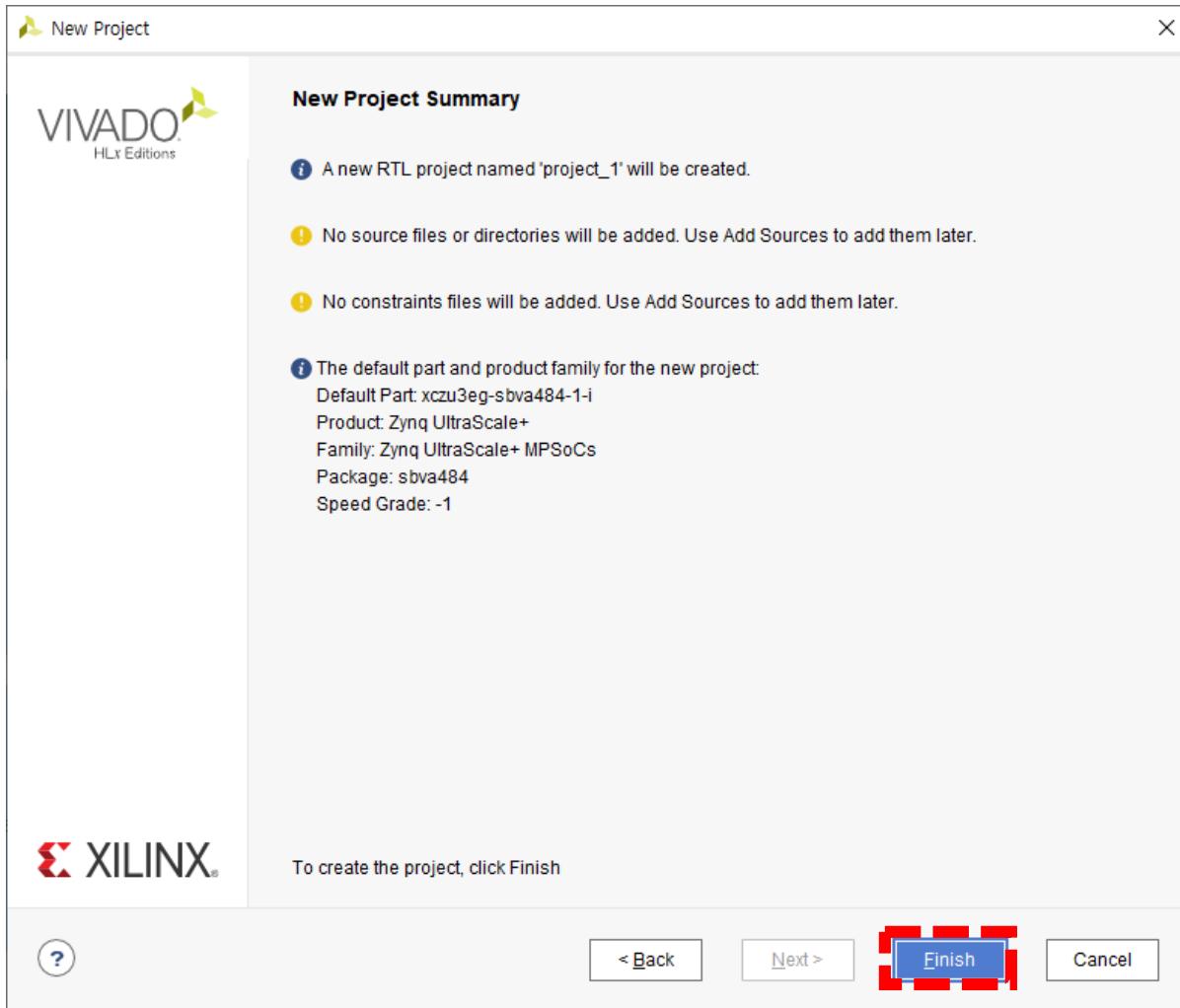
❖ 어떤 FPGA Device를 사용할지 선택하는 화면이 나오면 디바이스를 선택하기 위한 여러 가지 방법이 있지만, Search 창을 통해 디바이스의 파트번호를 입력하면 쉽게 선택할 수 있다.

# Step 1 Creating Vivado Project 8



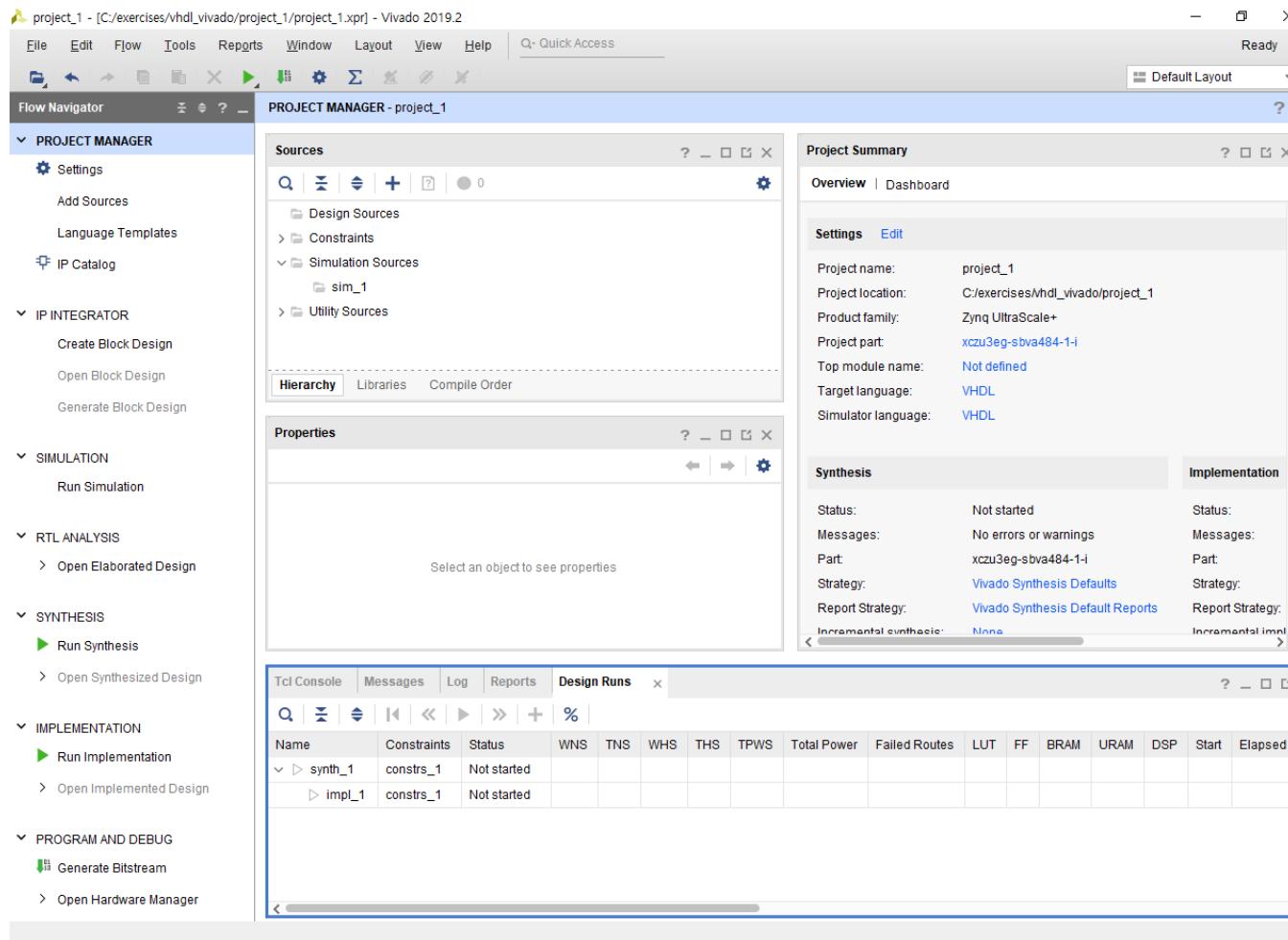
- ❖ Ultra96 보드에 탑재된 Device 인 xczu3eg-sbva484-1-i를 입력하면 디바이스가 하나만 남게 된다.
- ❖ 디바이스를 선택한 후 Next 버튼을 클릭한다.

# 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.

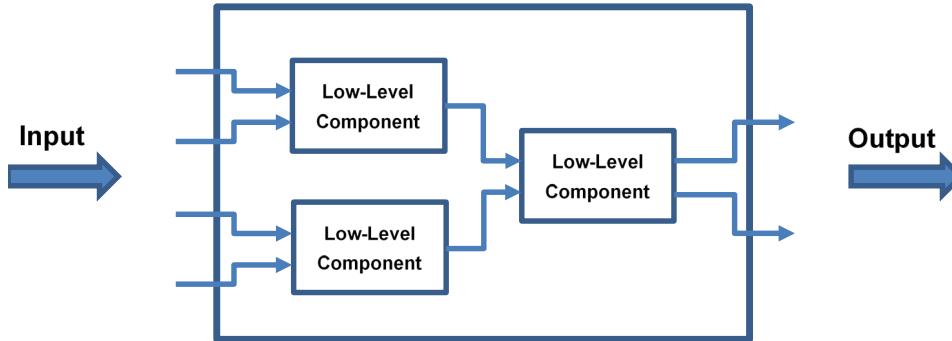
# Chapter 2 VHDL Basics

- **Hardware Modeling**
- **VHDL Basic Structure**
- **Signal Assignment**
- **Vivado Design Flow**

# Hardware Modeling

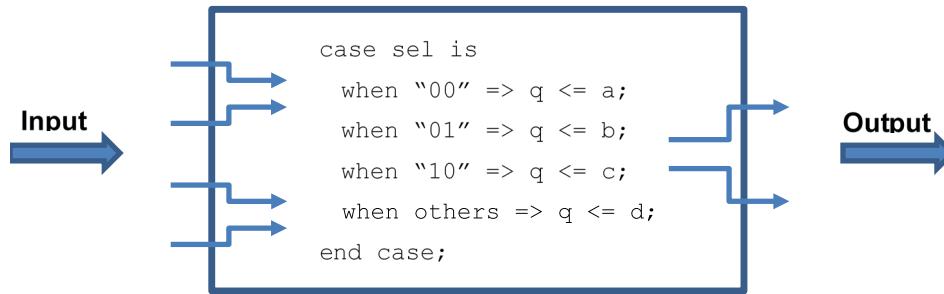
- ❖ Hardware Modeling이란 특정 하드웨어를 구현하기 위한 방법을 말한다.
- ❖ 이런 하드웨어 모델링 기법에는 Structural Modeling과 Behavioral Modeling의 두 가지 기법이 있다.

# 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 VHDL Basics

- Hardware Modelings
- VHDL Basic Structure
- Signal Assignment
- Vivado Design Flow

# VHDL Basics

- ❖ VHDL은 대소문자를 구분하지 않는다. (※ 대문자 A로 선언한 것과 소문자 a로 선언한 것은 같은 것으로 인식하며 참고로 Verilog HDL은 대소문자를 다른 것으로 인식한다.)
- ❖ 문장의 끝은 ; (세미콜론)으로 끝낸다.
- ❖ 주석처리는 --을 사용한다.

# VHDL Basic Structure

- ❖ VHDL에는 package가 있으며 package들을 모아 놓은 것이 library이다.
- ❖ 첫 번째 Statement와 두 번째 Statement는 사용 하려고 하는 package와 package를 포함하고 있는 library를 선언한 Statement이다.
- ❖ entity는 모듈의 이름과 포트를 표현하는 곳으로 이 모듈을 밖에서 보았을 때의 형태를 표현하는 곳이다.
- ❖ architecture는 architecture와 begin 사이의 선언부와 begin과 end 사이의 구현부의 두 부분으로 나누어지는데 선언부는 Data Type과 Signal 및 Component 등을 선언하는 부분이고 구현부는 회로의 구성을 표현하여 회로를 구현하는 부분이다.

```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity 엔터티이름 is  
    포트 선언  
end 엔터티이름;  
architecture 아키텍처이름 of 엔터티이름 is  
    데이터타입, 신호 및 컴포넌트 등 선언부  
begin  
    회로 구현부  
end 아키텍처이름;
```

# Chapter 2 VHDL Basics

- Hardware Modelings
- VHDL Basic Structure
- Signal Assignment
- Vivado Design Flow

# Signal Assignment 1

- ❖ Signal Assignment 란 회로상의 Wire를 연결하는 것을 말한다.
- ❖ 회로에서 불리는 Wire 또는 Net을 VHDL에서는 Signal로 표현하며 특정 Signal을 다른 Signal에 연결하는 것을 Signal Assignment라고 부른다.
- ❖ VHDL에서는 `<=` 을 사용하여 Signal Assignment를 한다.
- ❖ a, b 두 개의 포트로 들어오는 신호를 and 연산하여 result 포트로 Signal Assignment하여 출력하는 ander 모듈이다.

```
library ieee;
use ieee.std_logic_1164.all;

entity ander is
    port(
        a,b : in std_logic;
        result : out std_logic
    );
end ander;

architecture structural of ander is
begin
    result <= a and b;
end structural;
```

# Signal Assignment 2

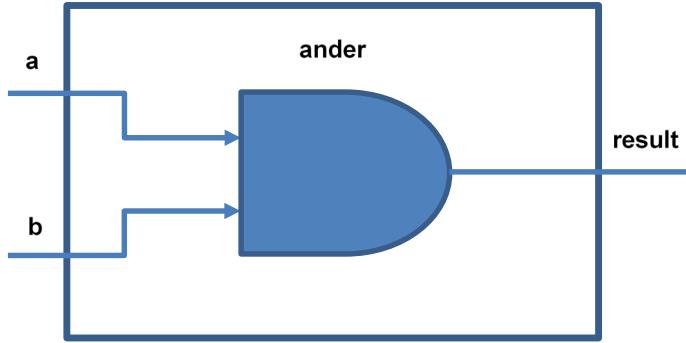
- ❖ Line 1과 Line 2는 IEEE 라이브러리 중 std\_logic\_1164 package를 사용하겠다는 선언이다. (※ 이 package 안에는 VHDL에서 가장 많이 사용하는 데이터 타입인 std\_logic 데이터 타입이 선언되어 있어서 항상 package로 선언된다.)
- ❖ ander라는 entity를 선언하고 ander 모듈의 port 선언한다. (※ a, b 두 개의 입력 포트와 result 한 개의 출력 포트가 선언되어 있다.)
- ❖ architecture는 ander 모듈의 내부 구조를 표현하는 곳이다. architecture는 앞에서 언급한 것과 같이 선언부와 구현부로 나누어지는데 여기서는 선언할 내용이 없으니 선언부인 architecture와 begin 사이에 아무런 코드가 없다.
- ❖ 구현부인 begin과 end사이에는 a와 b신호를 and 연산한 결과를 result로 Signal Assignment 한 Statement가 들어가 있다.

```
library ieee;
use ieee.std_logic_1164.all;

entity ander is
  port(
    a,b : in std_logic;
    result : out std_logic
  );
end ander;

architecture structural of ander is
begin
  result <= a and b;
end structural;
```

# Signal Assignment 3

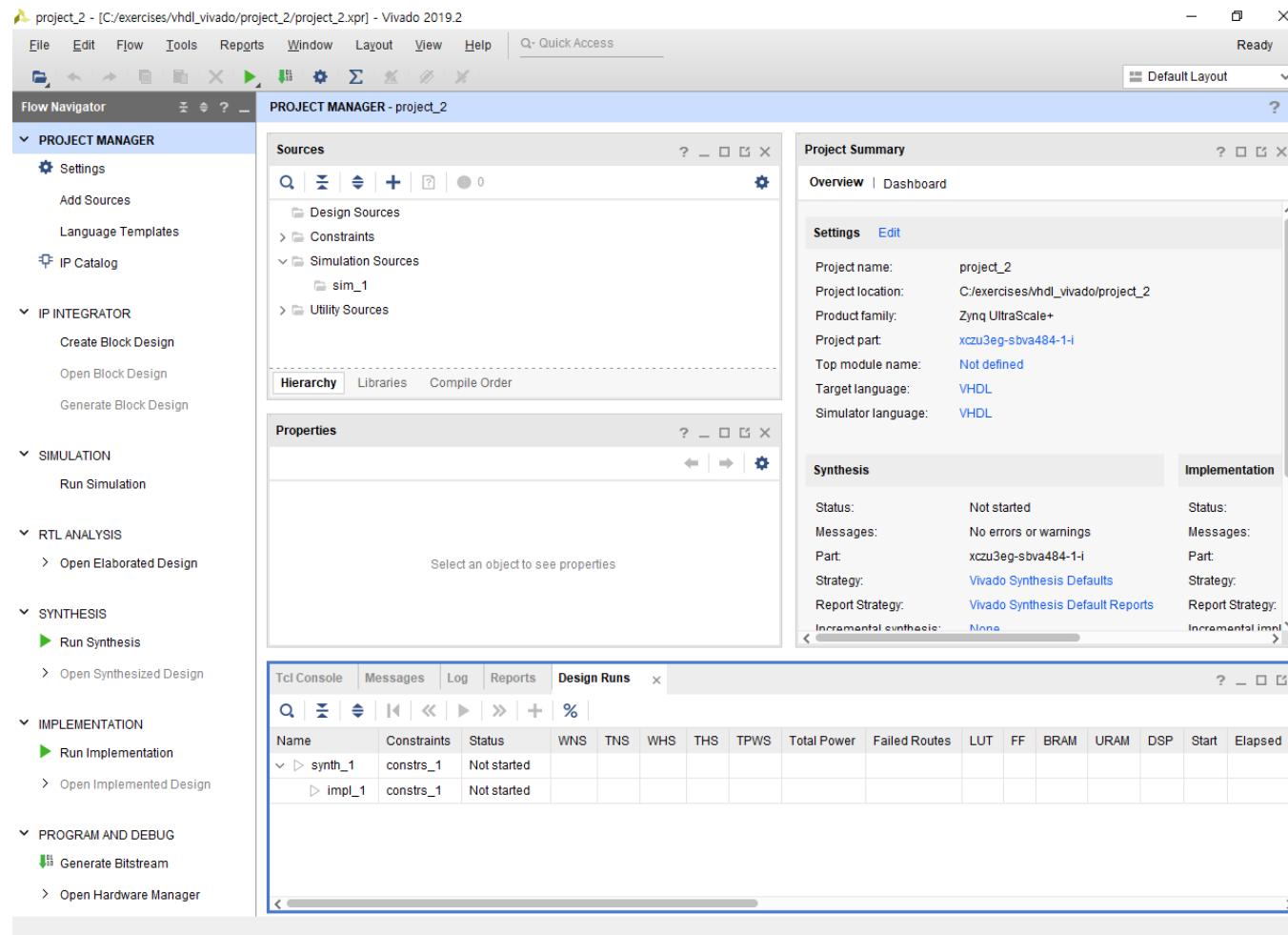


- ❖ Signal Assignment를 할 때 `<=`을 기준으로 왼쪽을 LHS(Left Hand Side) 오른쪽을 RHS(Right Hand Side)라고 부른다.
- ❖ VHDL에서 in port로 선언된 signal은 읽을 수만 있고 쓸 수는 없으며 out port로 선언된 signal은 쓸 수만 있고 읽을 수는 없다. (※ in port는 RHS에만 쓸 수 있고 out port는 LHS에만 쓸 수 있다.)
- ❖ LHS와 RHS는 데이터 타입과 비트 사이즈가 같아야 한다.
- ❖ ander 모듈에서 a, b, out은 모두 std\_logic으로 선언되어 있어서 데이터 타입이 같고 모두 1-bit로 비트 사이즈가 같다.

# Chapter 2 VHDL Basics

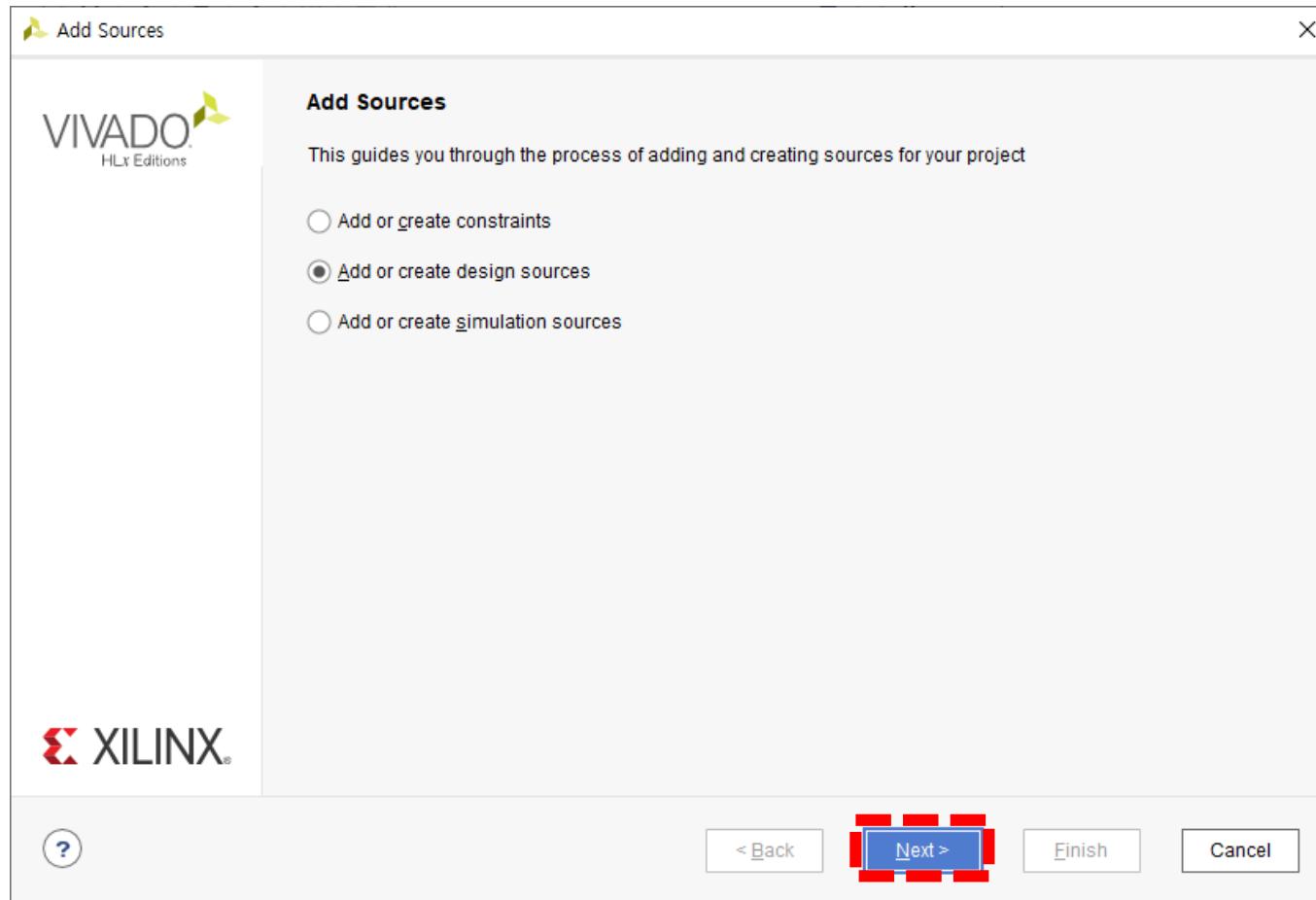
- Hardware Modelings
- VHDL Basic Structure
- Signal Assignment
- Vivado Design Flow

# Step 1 Creating Vivado Project



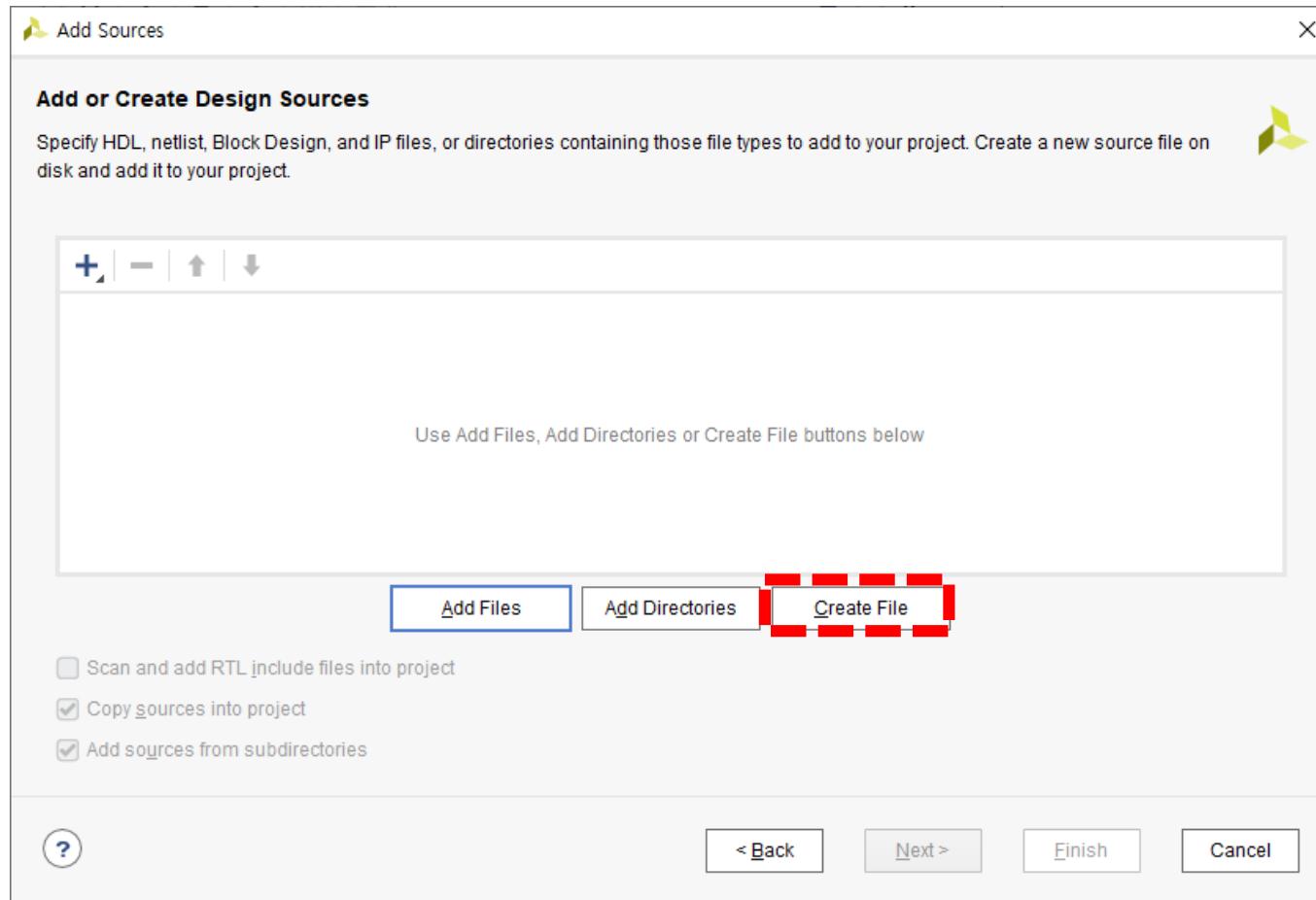
❖ Chapter 1의 Vivado 프로젝트 만들기를 참조하여 project\_2 프로젝트를 만든다.

# 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.

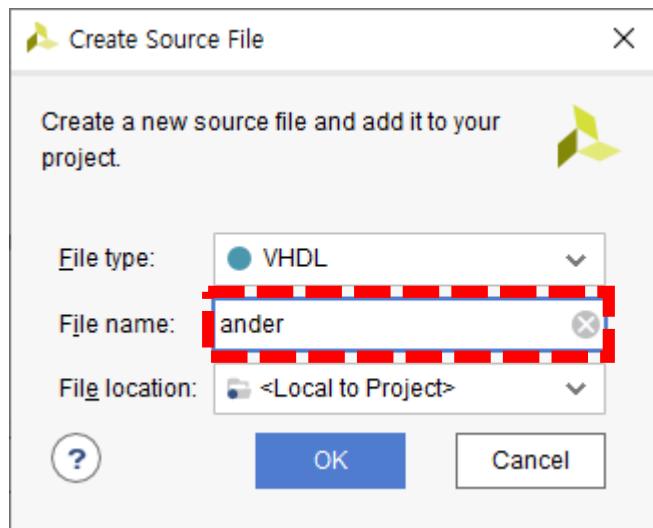
# Step 2 Add Source in Vivado Project 2



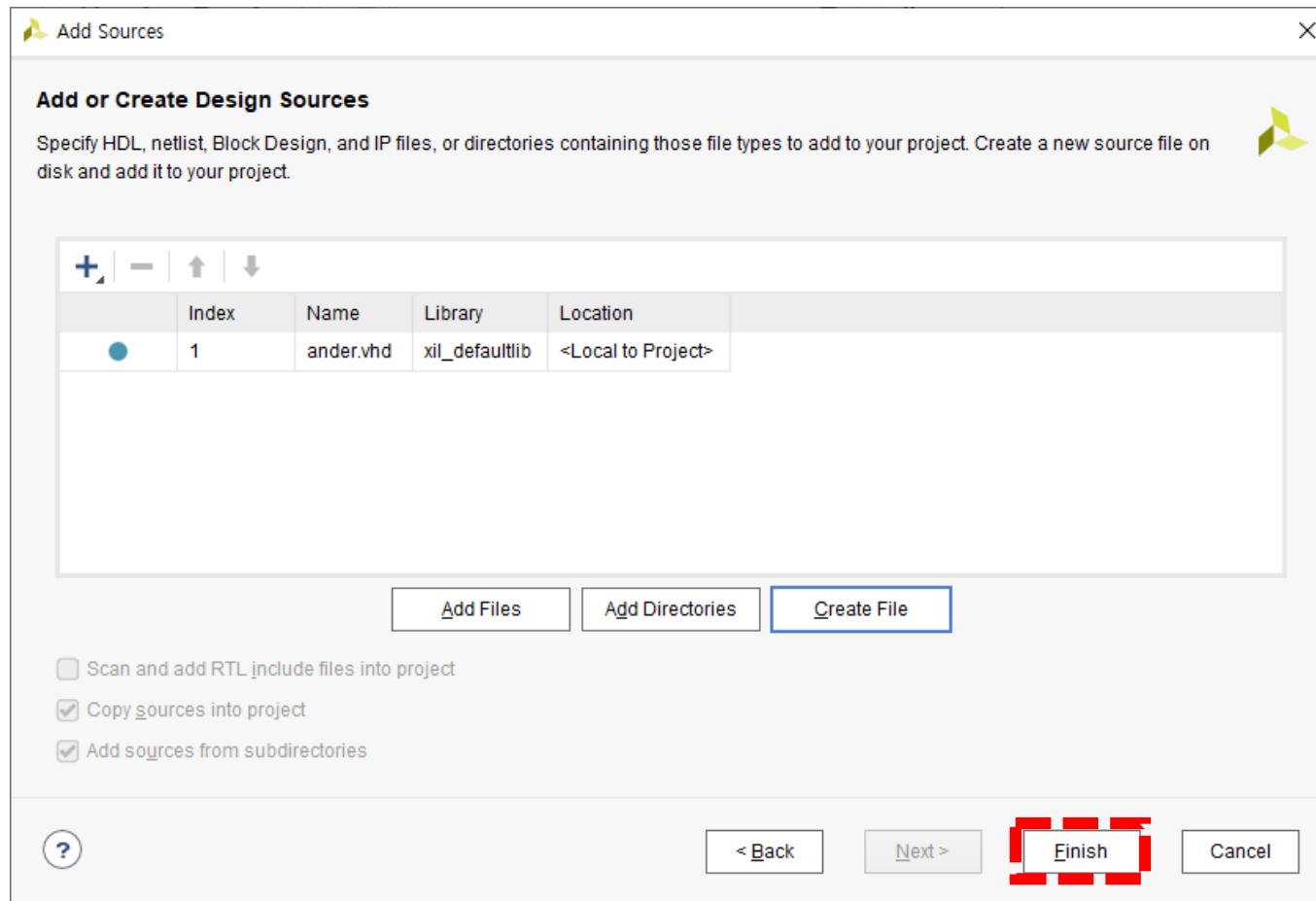
❖ 새로운 파일 생성하여 프로젝트에 추가하기 위해 Create File 버튼을 클릭한다.

## Step 2 Add Source in Vivado Project 3

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

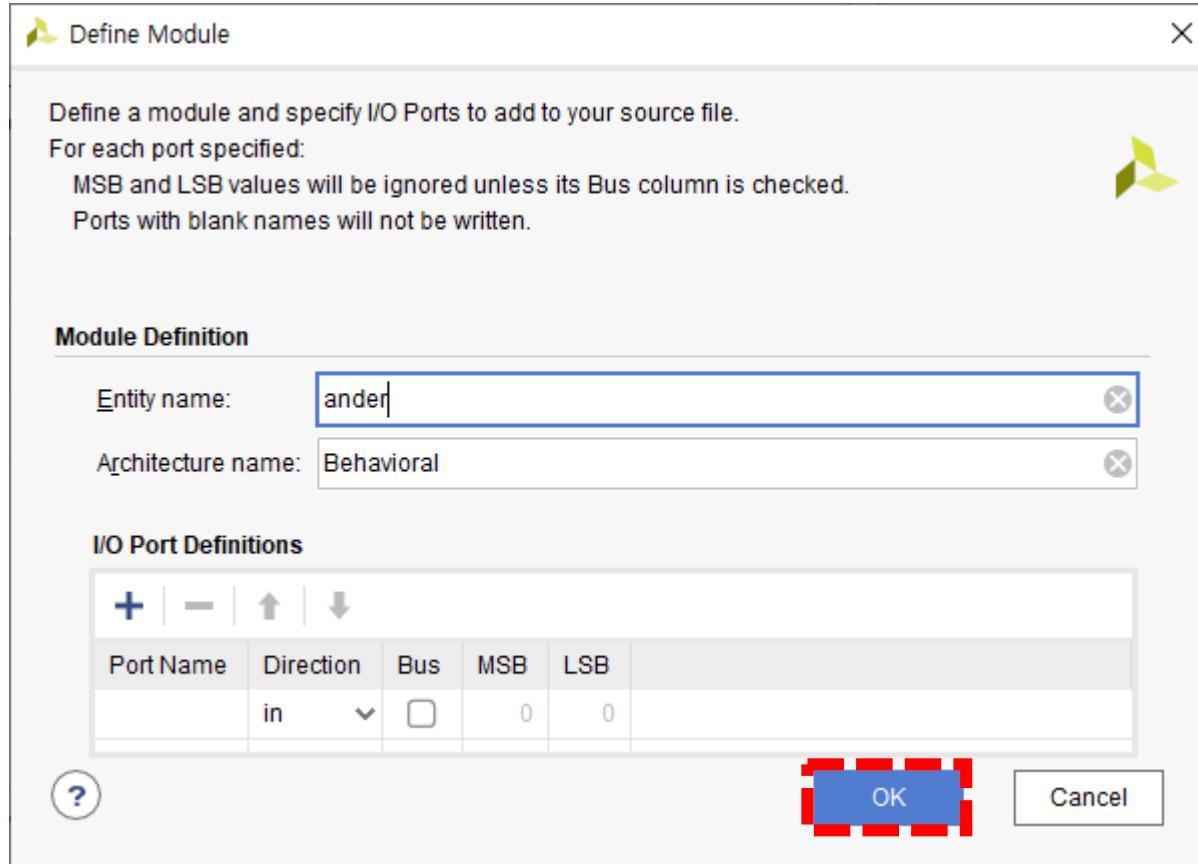


# Step 2 Add Source in Vivado Project 4



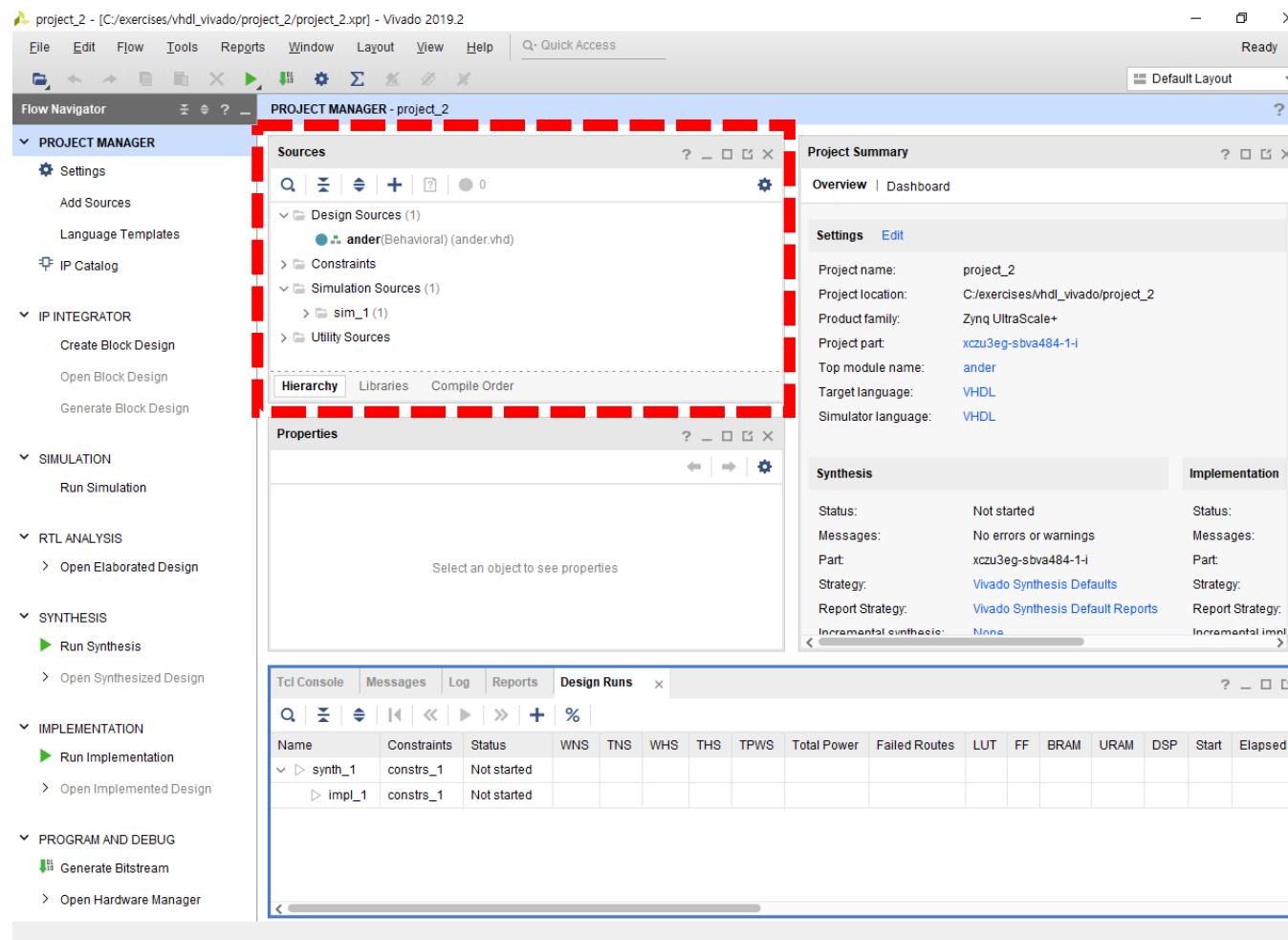
- ❖ 전에 생성한 ander.vhd 파일이 추가되어 있는 것을 확인할 수 있다.
- ❖ Finish 버튼을 클릭하여 소스 추가를 완료한다.

## Step 2 Add Source in Vivado Project 5



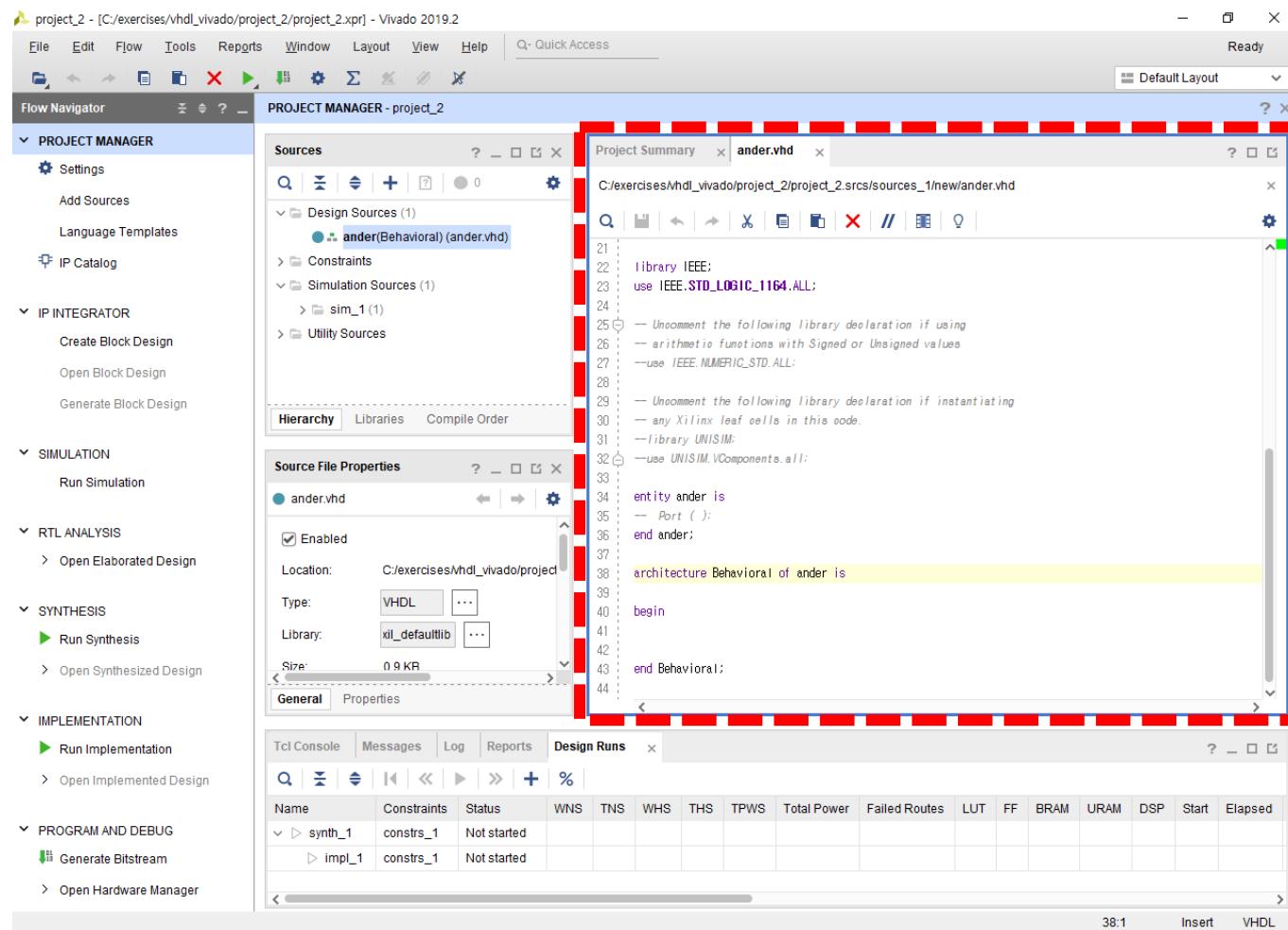
❖ 생성된 ander 모듈의 입출력 포트를 설정하는 화면이다. 여기서 포트를 설정하면 ander.vhd 파일의 포트 선언부를 자동으로 작성해준다. 여기서는 직접 코딩을 할 예정이니 아무 설정 없이 OK 버튼을 클릭한다.

# Step 2 Add Source in Vivado Project 6



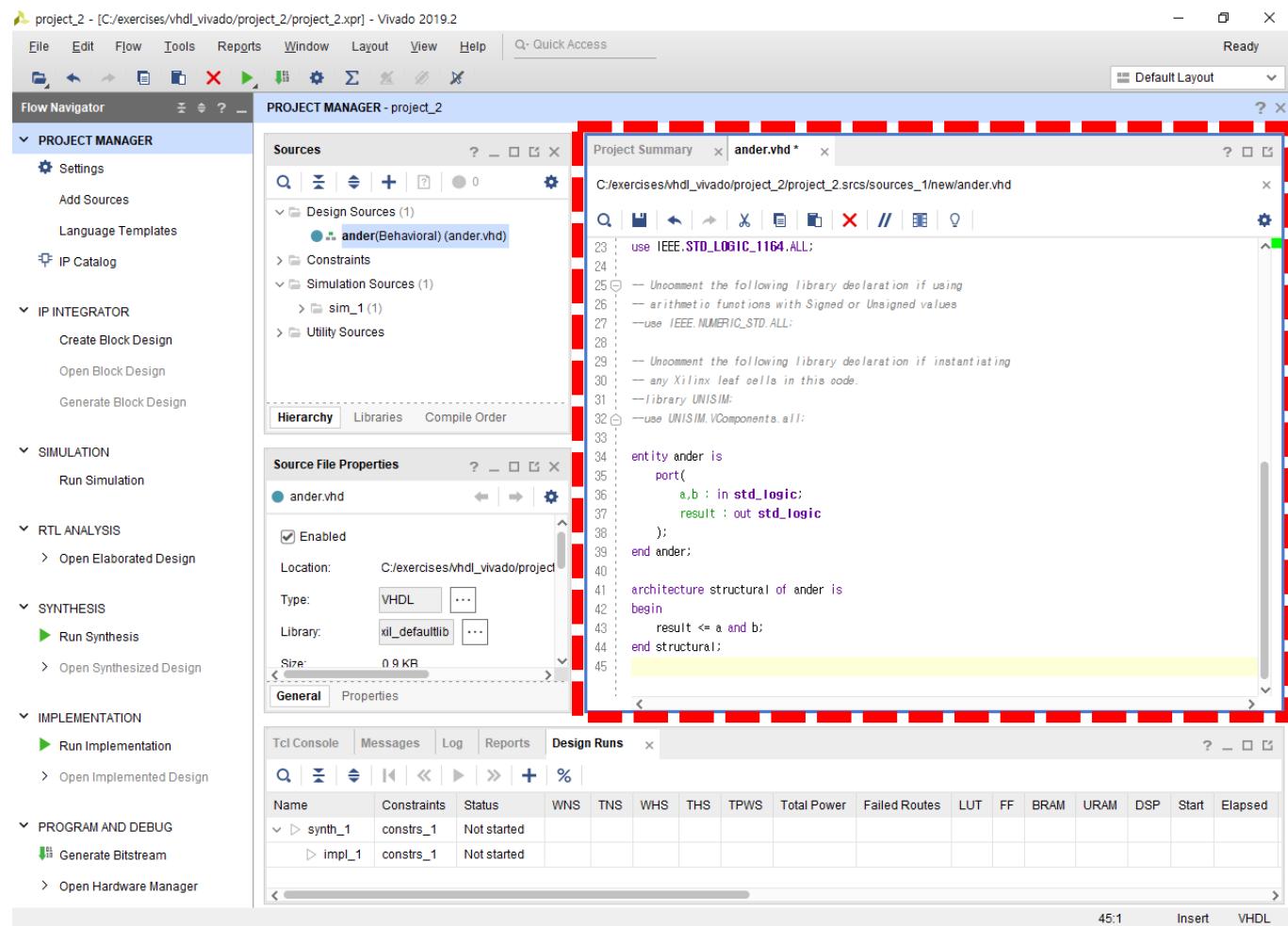
❖ 프로젝트 그림 2-13과 같이  
Sources window안에  
ander.vhd 소스파일이 추가된  
것을 확인할 수 있다.

# Step 3 Modifying Source Code 1



❖ Sources window 안에  
ander.vhd파일을 더블 클릭하  
면 Editor window에  
ander.vhd 파일의 소스코드를  
확인할 수 있다. (※ Vivado는  
VHDL의 기본적인 형태를 만들  
어 준다.)

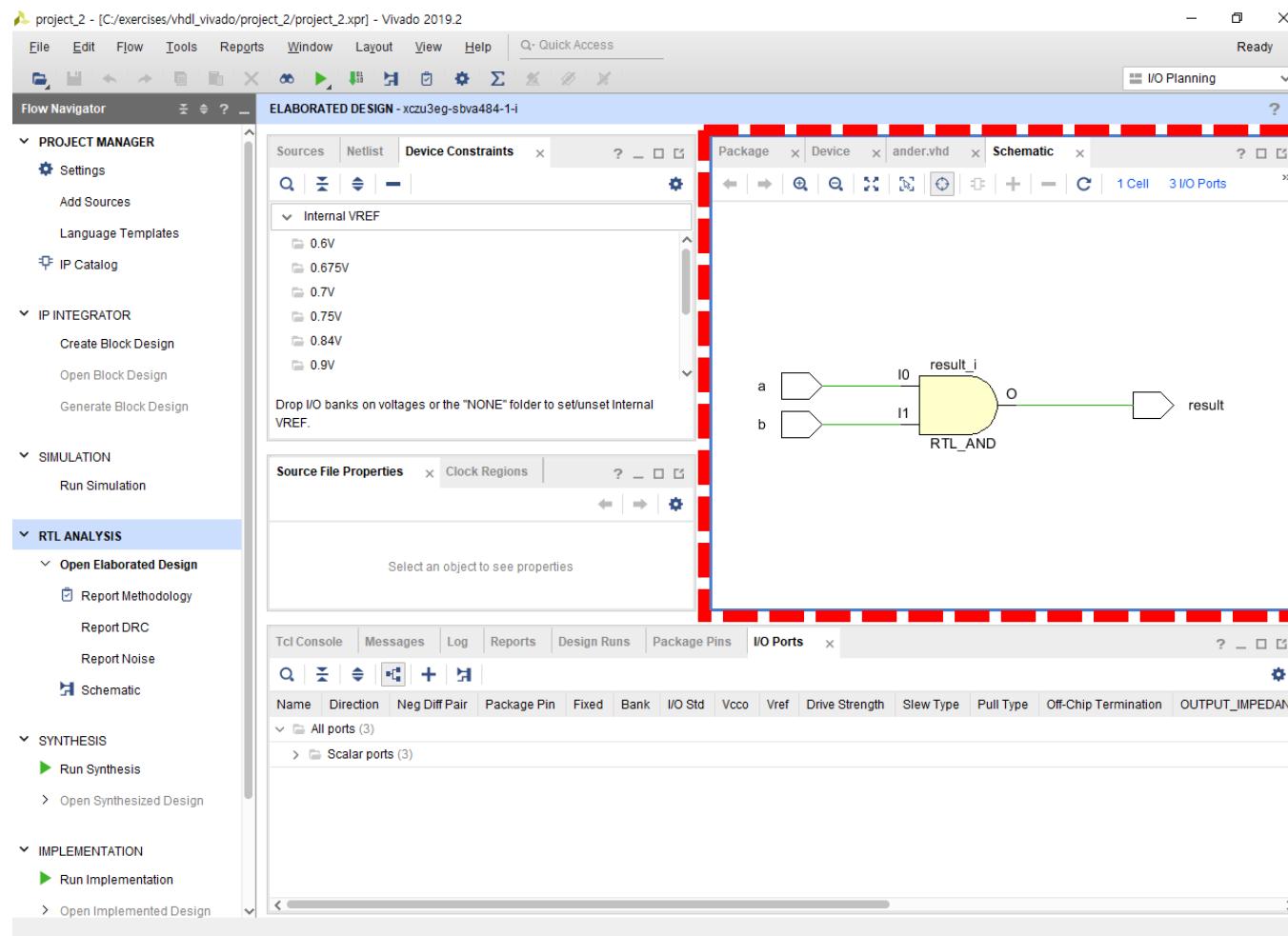
# Step 3 Modifying Source Code 2



❖ Editor Window 안에 ander 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

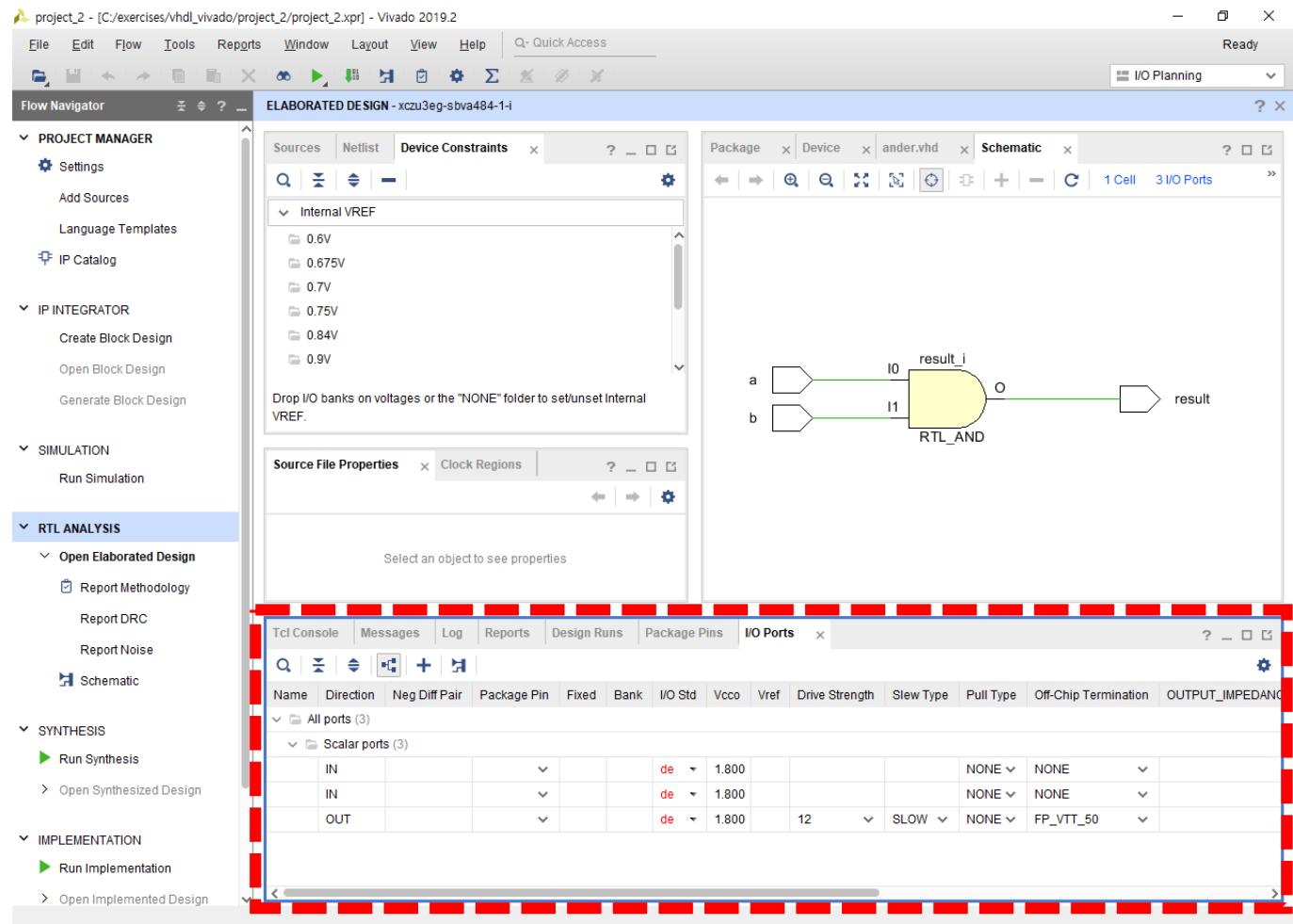


❖ Flow Navigator  $\Rightarrow$  RTL Analysis  $\Rightarrow$  Open Elaborated Design  $\Rightarrow$  Schematic을 클릭하면 Elaboration이 진행된 후 Schematic을 보여준다.

# Step 5 Pin Constraints 1

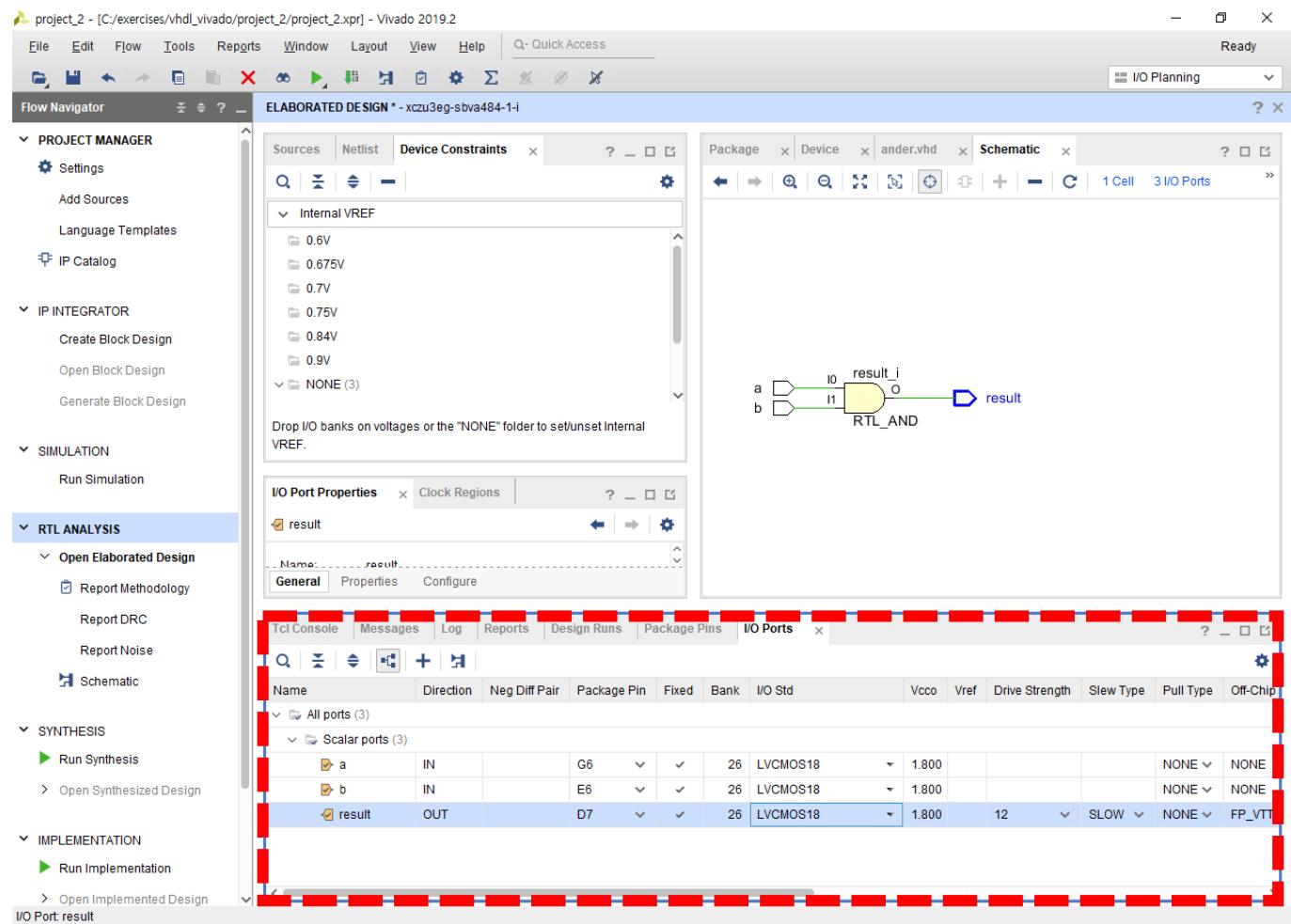
- ❖ Constraints는 사전적 의미로는 제한하다 라는 말로써 FPGA 설계에 필요한 제한조건을 만들기 위한 것이다.
- ❖ 예를 들어 가장 기본적인 Constraints에는 Pin Constraints가 있는데 Pin Constraints란 FPGA Device에 설계한 하드웨어의 입출력 포트가 FPGA Device의 어떤 Pin을 사용할지 지정하는 것이다.
- ❖ 보드에 다운로드 하지 않고 합성 또는 시뮬레이션만 한다면 Pin Constraints를 할 필요가 없지만 여기서는 Ultra96 보드에 다운로딩하여 구현할 예정이니 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



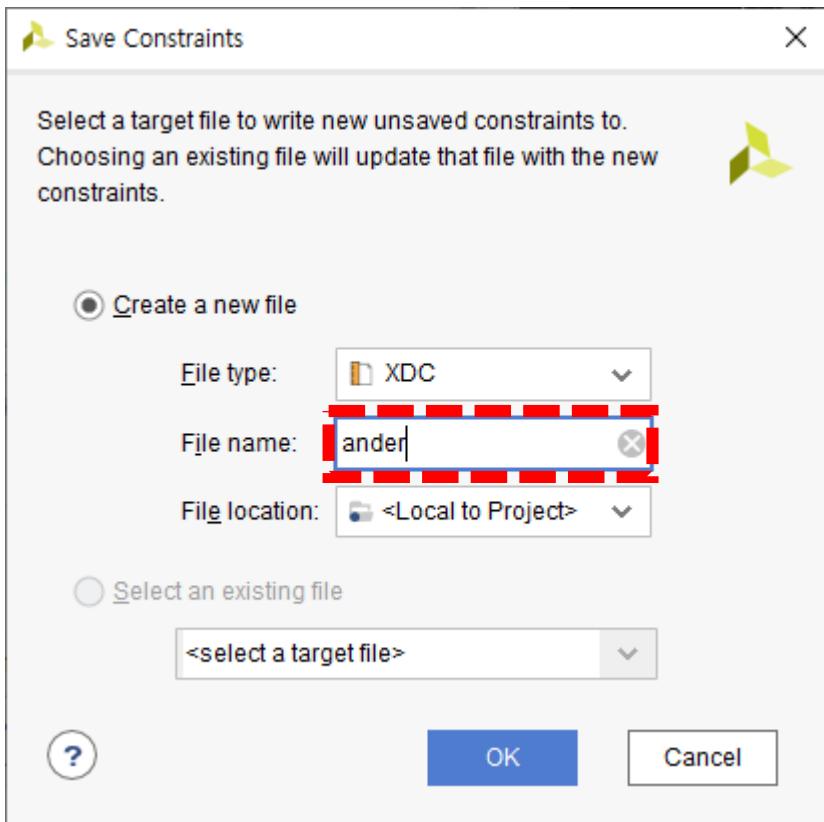
- ❖ Layout ⇒ I/O Planning을 클릭하면 하단에 I/O Ports 창이 나온다.
- ❖ I/O Ports 창 안에 Scalar ports를 열어보면 Ander 모듈의 포트들이 있는 것을 확인할 수 있다.

# Step 5 Pin Constraints 3



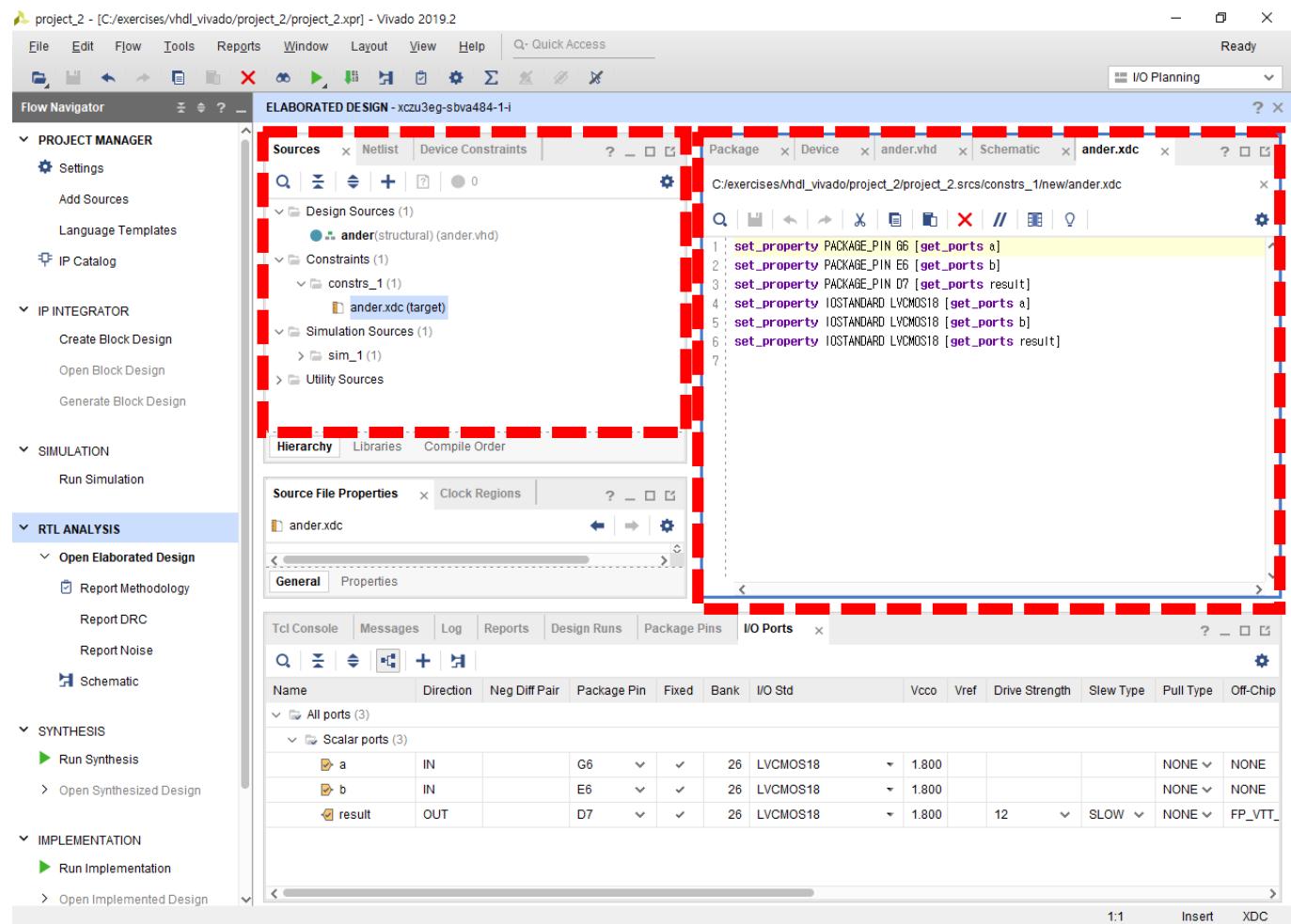
- ❖ Package Pin 열에 핀 번호를 입력하면 a, b, result 포트에 Pin Constraints를 할 수 있다.
- ❖ a → G6, b → E6, result → D7로 Pin Constraints를 하고 I/O Std열은 모두 LVCMS18을 선택한다.

# Step 5 Pin Constraints 4



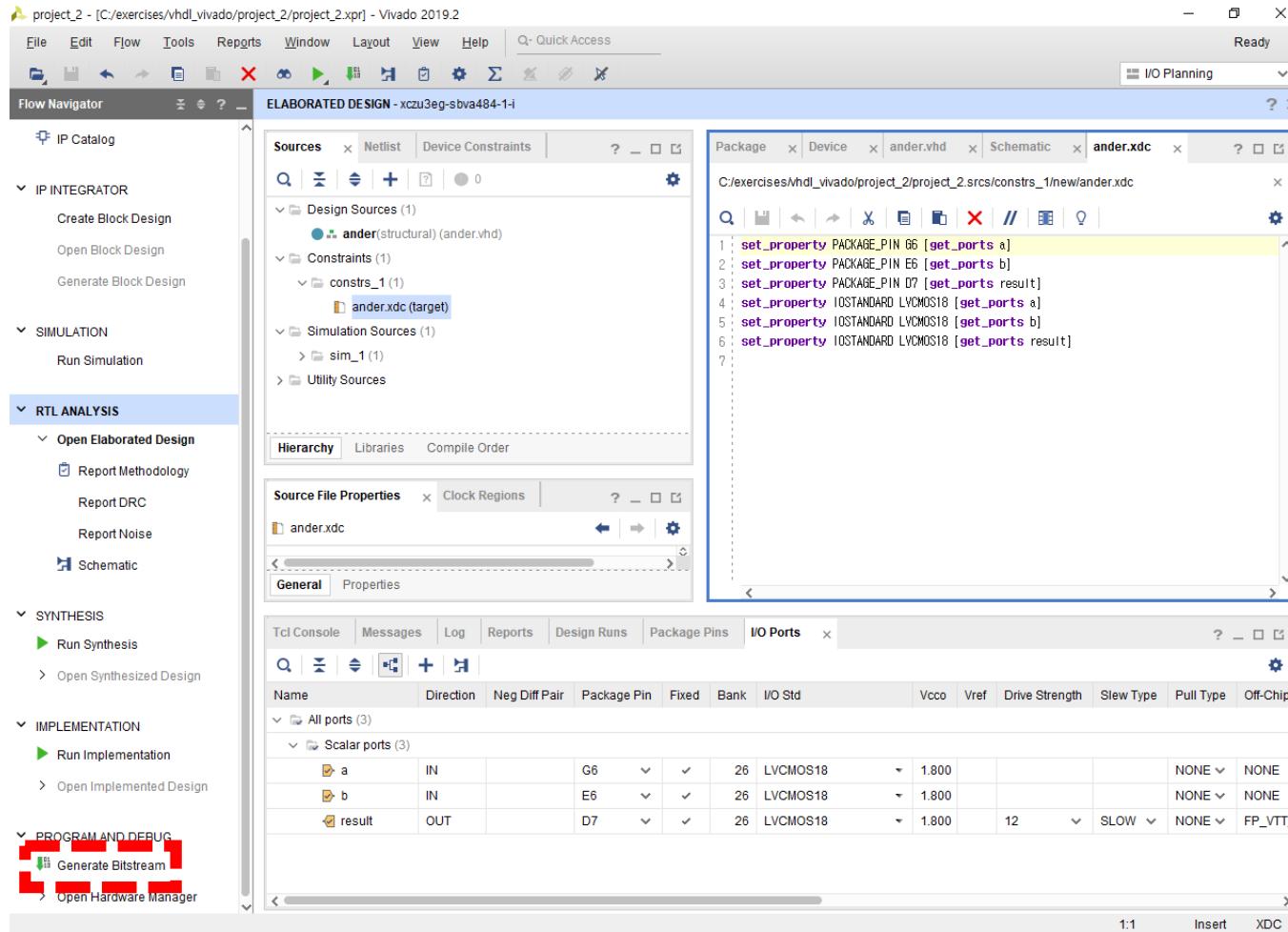
- ❖ File ⇒ Constraints ⇒ Save 메뉴 또는 Ctrl+S키를 사용하면 Constraints를 저장할 수 있다.
- ❖ Constraints를 저장할 파일 이름을 입력하는 창이 나오면 ander를 타이핑한 후 OK버튼을 클릭한다.

# Step 5 Pin Constraints 5



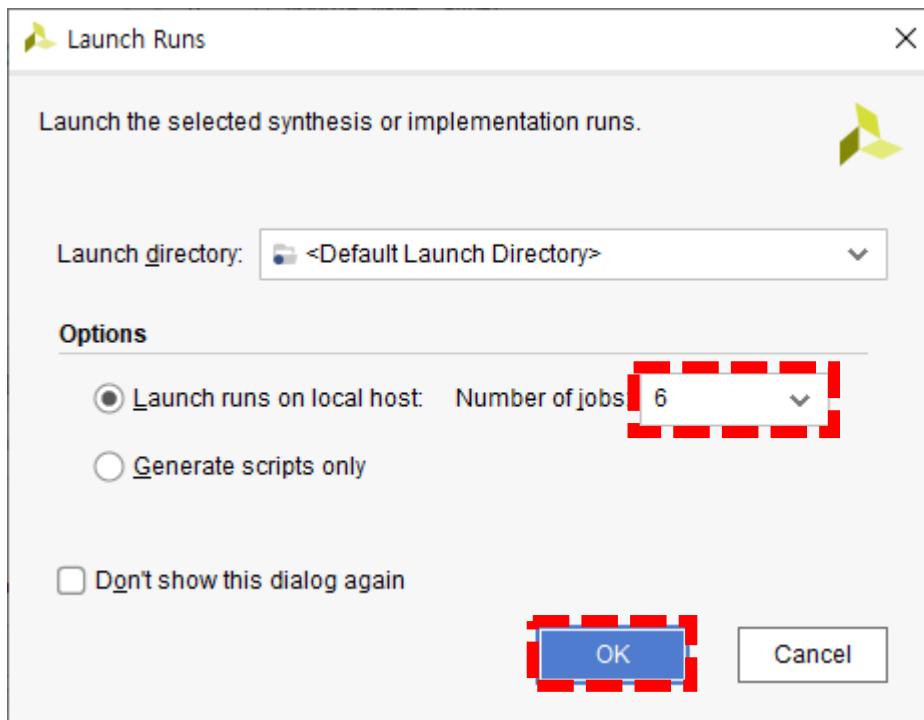
- ❖ Sources 탭을 클릭하면 Constraints 목록에 ander.xdc 파일이 추가되어 있는 것을 확인할 수 있다.
- ❖ ander.xdc 파일을 더블 클릭하면 Editor Window에서 Constraints 내용을 확인할 수 있다.

# Step 6 Generate Bitstream 1



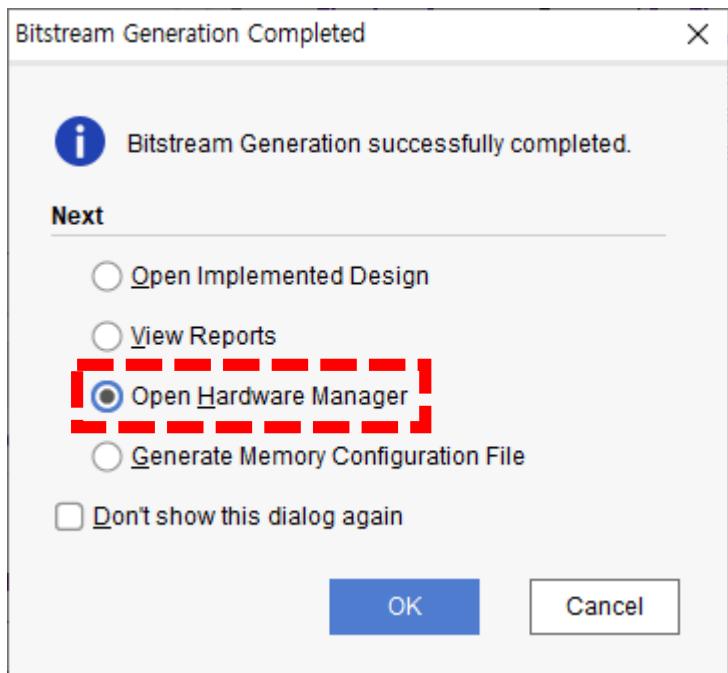
❖ Bitstream 생성을 위해 Flow Navigator ⇒ PROGRAM AND DEBUG ⇒ Generate Bitstream 을 클릭한다.

# Step 6 Generate Bitstream 2



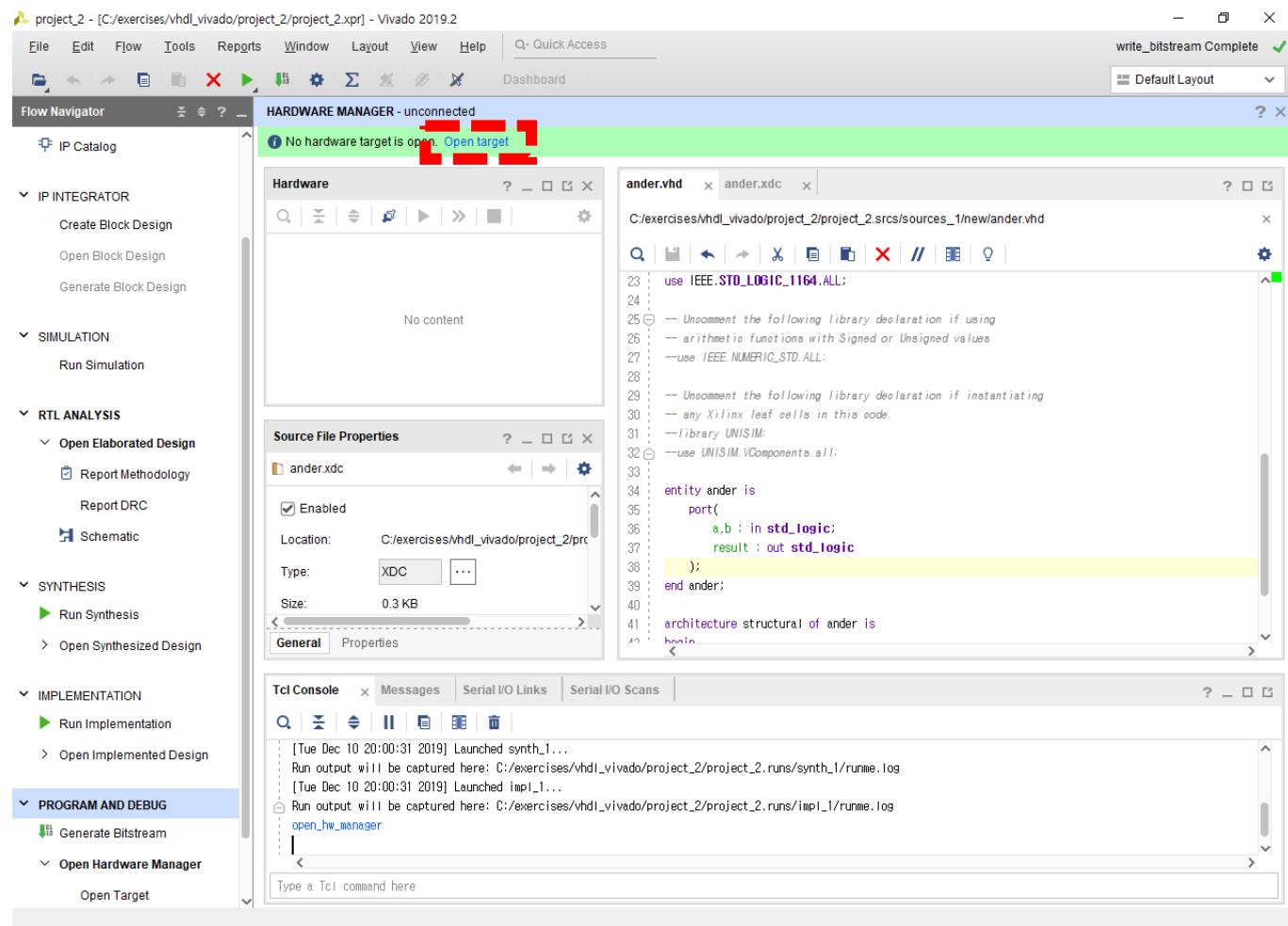
- ❖ Launch Runs 창은 Bitstream 을 생성하기 위해 CPU 코어를 몇 개 사용할 지 선택하는 창이다.
- ❖ 적당한 개수를 선택한 후 OK 버튼을 클릭한다.

# Step 7 Programming Device 1



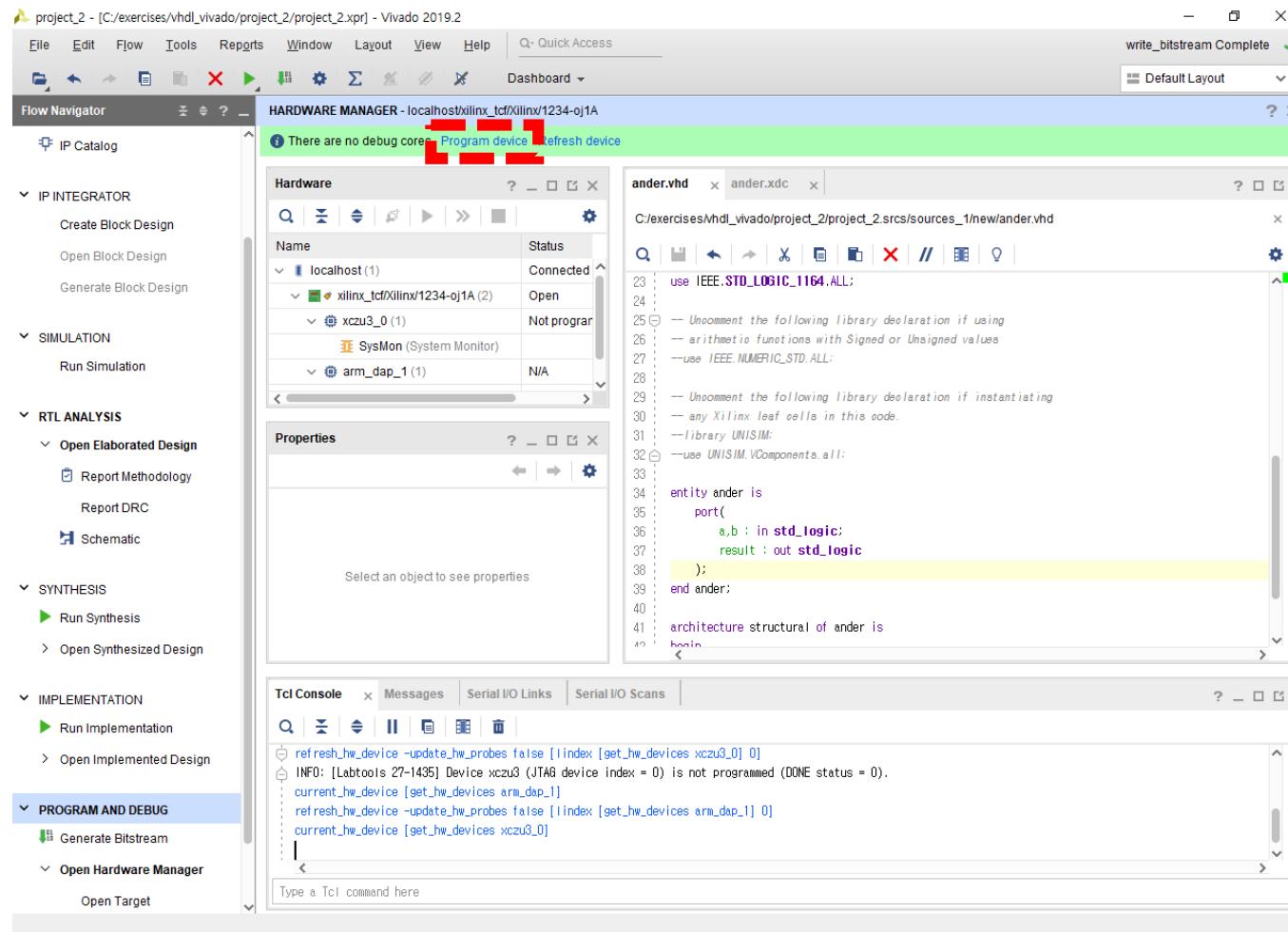
- ❖ Generate Bitstream이 정상적으로 완료되면 생성된 Bitstream을 디바이스에 프로그래밍하기 위해 하드웨어 매니저를 열 것인지 묻는 창이 나온다.
- ❖ Open Hardware Manager를 선택한 후 OK 버튼을 클릭한다.  
(※ Flow Navigator ⇒ PROGRAM AND DEBUG ⇒ Open Hardware Manager를 클릭해도 된다.)

# 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 Pmod 8LD, PMOD\_B 커넥터의 윗줄에 Pmod BTN을 연결한 후 POWER 버튼을 누른다. (※ Pmod Connector에는 모두 Pull-Up 저항이 연결되어 있어서 초기 상태에 LED가 연결되면 모두 켜져 있는 상태가 된다.)
- ❖ Hardware Window 위에 Open Target 이라고 되어 있는 부분을 클릭한다.
- ❖ 풀 다운 메뉴가 나오면 Auto Connect를 선택한다.

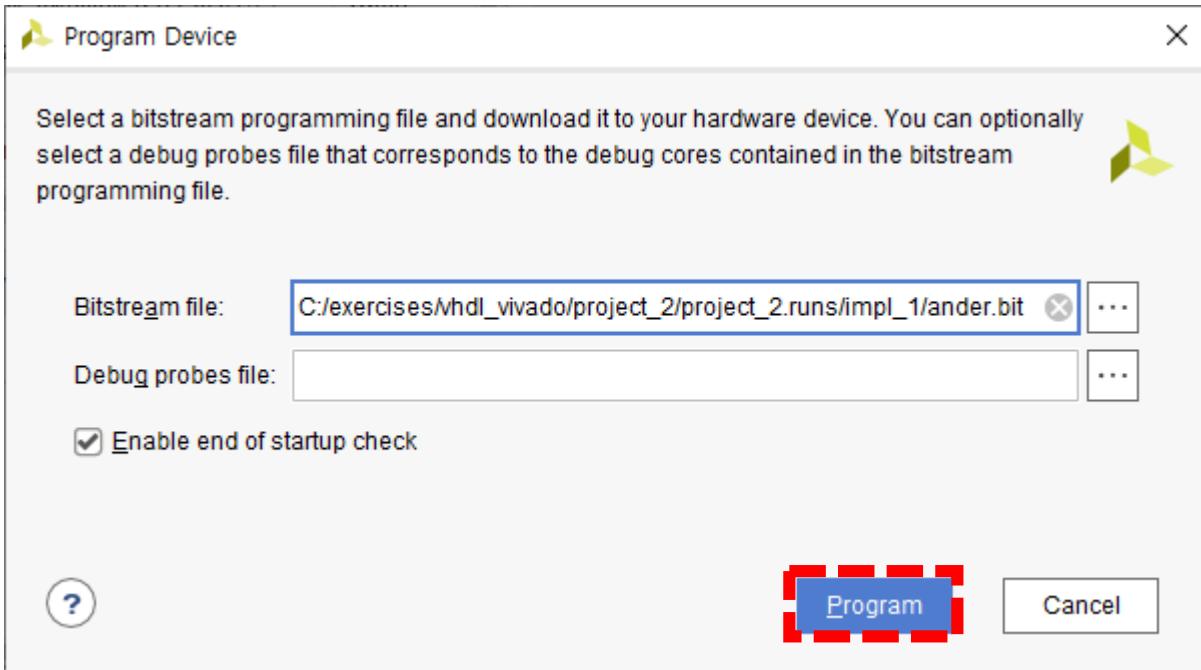
# Step 7 Programming Device 3



❖ 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 Pmod BTN의 BTN0와 BTN1을 동시에 누르면 Pmod 8LD의 LD0가 켜지는 것을 확인할 수 있다.
- ❖ 동작을 확인한 후 POWER 버튼을 10초 정도 눌러서 시스템을 종료한다.

# Chapter 3 VHDL Basic Syntax

- Constant & Signal
- VHDL Data Types
- Bit Slice
- Operators
- Ultra96 Training Kit Exercises 1

# Number Representations

- ❖ VHDL은 하드웨어를 표현하는 언어이어서 회로의 Wire의 상태를 표시하는 형태인 2진수(Binary)로 표현된다.
- ❖ 따옴표가 없다면 10진수(Decimal) 표현이 된다.
- ❖ 16-bit 또는 32-bit와 같이 사이즈가 커서 2진수로 표현하면 숫자가 길어져서 알아보기 어려울 경우 8진수(Octal) 또는 16진수(Hexadecimal)로 표현하거나 2진수의 중간에 \_ (underbar)와 같은 기호로 구분하여 표현할 수 있다.
- ❖ X는 16진수로 표현하는 것이고 O는 8진수로 표현하는 것이고 B는 \_ 기호를 사용하겠다는 것을 표현한 것이다. (※ 10진수로 표현하려면 따옴표를 쓰지 않으면 된다.)

- ▷ '1', '0' : 1-bit 에는 작은따옴표 사용
- ▷ "11", "100" : 2-bit 이상에는 큰따옴표 사용
- ▷ "111101001100000" : 16-bit, 2진수(Binary)
- ▷ 64096 : 16-bit, 10진수(Decimal)
- ▷ X"FA60" : 16-bit, 16진수(Hexadecimal)
- ▷ O"175140" : 16-bit, 8진수(Octal)
- ▷ B"1111\_1010\_0110\_0000" : 16-bit, 2진수(Binary), \_ 표현
- ▷ ( **others => '0'** ) : 사이즈와 상관없이 모두 0을 인가

# Constant

❖ Constant는 상수를 의미하는 것으로 코드의 가독성(readability)과 유연성 (flexibility)을 높이기 위해 사용한다.

❖ Constant는 Architecture와 Begin 사이의 선언부에 선언한 후 Begin과 End사이의 구현부에서 상수로 사용한다.

```
▷ Constant width : integer := 16;  
▷ Constant period : time := 10 ns;
```

# Signal

- ❖ Signal은 회로상의 wire를 표현한 것이며 컴포넌트, 포트 또는 프로세스들을 서로 연결하는 데 사용한다.
- ❖ 선언부에 c라는 Signal을 선언한 다음 a와 b를 and 연산한 결과를 c에 인가한 후 c의 값을 result에 인가하였다.
- ❖ 이와 같이 특정 신호를 받아서 내부적으로 사용해야 할 경우 Signal로 선언하여 사용한다.

```
library ieee;
use ieee.std_logic_1164.all;

entity ander2 is
  port(
    a,b : in std_logic;
    result : out std_logic
  );
end ander2;

architecture structural of ander2 is
  signal c : std_logic;
begin
  c <= a and b;
  result <= c;
end structural;
```

# Chapter 3 VHDL Basic Syntax

- Constant & Signal
- VHDL Data Types
- Bit Slice
- Operators
- Ultra96 Training Kit Exercises 1

# VHDL Data Types

- ❖ std\_logic은 VHDL의 여러 가지 Data Type 중 하나이며 VHDL의 여러 가지 Data Type 중 가장 많이 사용하는 Data Type이다.
- ❖ VHDL Data Types은 VHDL Package에 저장되어 있는데 아래 Directory 안에 std, std\_2008, ieee, ieee\_2008과 같은 Directory를 찾을 수 있다. Directory 이름들은 모듈 위에 선언했던 Library 이름이고 이 Directory 안에 Package들이 들어 있다.
  - ❖ Vivado 설치 디렉토리 > Vivado > Version > data > vhdl > src
- ❖ ieee\_2008 directory 안에 들어가면 IEEE Library 다음에 선언했던 Package인 std\_logic\_1164.vhdl이 있다. 이 Package에는 VHDL에서 가장 많이 사용하는 Data Type인 std\_logic이 선언되어 있어서 거의 모든 VHDL 코드의 제일 위에는 std\_logic\_1164 패키지를 선언한 코드가 나온다.
- ❖ 다른 Data Type들은 std 또는 std\_2008 Directory 안에 standard.vhd 파일에 선언되어 있는데 std Directory안의 Package들은 built-in Package라고 부르며 별도로 선언하지 않아도 되는 Package이다.
- ❖ std\_logic\_1164 package와 standard package에 선언되어 있는 Data Types이 VHDL의 기본 Data Types이다.

# STD\_ULOGIC & STD\_LOGIC Data Type

- ❖ VHDL에서 Data Type을 선언할 때는 type이라는 예약어를 사용하고 그 다음에는 Data Type Name을 쓰고 is라는 예약어가 오고 괄호 안에 Data Type이 가질 수 있는 값들을 나열한다.
- ❖ 우측에 std\_logic\_1164 패키지 안에 선언된 STD\_ULOGIC Data Type을 보면 STD\_ULOGIC은 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-'의 총 9개의 값을 가질 수 있는 Data Type임을 알 수 있다.
- ❖ STD\_LOGIC은 STD\_ULOGIC의 subtype으로 선언되어 있는데 STD\_ULOGIC Data Type을 resolved function으로 처리한 결과가 STD\_LOGIC Data Type이 된다.

```
type STD_ULOGIC is ( 'U', -- Uninitialized
                      'X',      -- Forcing Unknown
                      '0',      -- Forcing 0
                      '1',      -- Forcing 1
                      'Z',      -- High Impedance
                      'W',      -- Weak Unknown
                      'L',      -- Weak 0
                      'H',      -- Weak 1
                      '-');     -- Don't care
```

```
subtype STD_LOGIC is resolved STD_ULOGIC;
```

# INTEGER & NATURAL Data Type

- ❖ INTEGER Data Type 은 -2147483648부터 2147483647까지의 정수 값을 가질 수 있는 정수 Data Type이다. (※ 32-bit 데이터를 Signed Decimal로 값을 표현하면 위 값이 된다. 즉, X"80000000"는 Signed Decimal 값으로 -2147483648이고 X"7FFFFFFF"는 Signed Decimal 값으로 2147483647이 된다.)

```
type INTEGER is range -2147483648 to 2147483647;
```

- ❖ NATURAL Data Type은 0부터 INTEGER'HIGH까지의 0을 포함한 양수 값을 가질 수 있는 양수 Data Type이다. (※ 'HIGH는 VHDL Attribute인데 선언된 INTEGER range 중 큰 값을 의미한다. 즉, INTEGER'HIGH는 2147483647이 되므로 NATURAL Data Type은 0부터 2147483647까지의 값을 가질 수 있다.)

```
subtype NATURAL is INTEGER range 0 to INTEGER'HIGH;
```

# STD\_ULOGIC\_VECTOR & STD\_LOGIC\_VECTOR Data Type

- ❖ STD\_ULOGIC\_VECTOR Data Type은 STD\_ULOGIC의 array로 선언되어 있다.
- ❖ array를 사용하면 STD\_ULOGIC Data Type의 배열로 선언한 Data Type이 된다.
- ❖ STD\_ULOGIC\_VECTOR Data Type의 array range는 NATURAL range로 선언되어 있어서 0부터 2147483647까지의 range를 선언하여 사용할 수 있다.
- ❖ STD\_LOGIC\_VECTOR Data Type은 STD\_LOGIC과 같이 STD\_ULOGIC\_VECTOR의 subtype으로 resolved 타입으로 변형한 Data Type이다.
- ❖ 결과적으로 STD\_LOGIC\_VECTOR는 STD\_LOGIC 데이터 타입을 array로 선언한 것과 같다.
- ❖ STD\_LOGIC\_VECTOR Data Type은 VHDL에서 가장 많이 사용하는 Data Type이다.

```
type STD_ULOGIC_VECTOR is array (NATURAL range <>)
of STD_ULOGIC;
```

```
subtype STD_LOGIC_VECTOR is (resolved)
STD_ULOGIC_VECTOR;
```

# BOOLEAN & BIT Data Type

- ❖ BOOLEAN Data Type은 FALSE와 TRUE 값 중 하나를 가질 수 있는 데이터 타입이다.
- ❖ BIT Data Type은 그림 3-9에서 선언된 것처럼 0과 1 값 중 하나를 가질 수 있는데 이터 타입이다.

```
type BOOLEAN is (FALSE, TRUE);
```

```
type BIT is ('0', '1');
```

# BIT\_VECTOR & ENUMERATED Data Type

- ❖ BIT\_VECTOR Data Type은 BIT Data Type의 array type이다.
- ❖ ENUMERATED Data Type은 새로운 형태의 Data Type을 만들어서 사용하는 Data Type이다.
- ❖ 우측 예제 코드에서는 idle, coin\_in, ready, coffee, coin\_out 총 5 개의 값을 가질 수 있는 Data Type으로 만들었다.
- ❖ ENUMERATED Data Type은 State Machine 을 만들 때 필요한 State를 표현하기 위해 사용한다.

```
type BIT_VECTOR is array (NATURAL range <>) of BIT;
```

```
type state is (idle, coin_in, ready, coffee, coin_out);
```

# Chapter 3 VHDL Basic Syntax

- Constant & Signal
- VHDL Data Types
- Bit Slice
- Operators
- Ultra96 Training Kit Exercises 1

# Bit Slice

- ❖ VHDL Data Type 중 Vector Data Type은 각 Data Type의 Array Data Type으로 여러 bit를 하나로 묶은 것을 의미한다.
- ❖ Vector Data Type은 VHDL에서는 (High-Value downto Low-Value) 또는 (Low-Value to High-Value)로 표현한다.
- ❖ Low-Value는 일반적으로 0이 되며 High-Value는 사용하려고 하는 Bit Size에서 1을 뺀 값이 된다. (※ 16-bit size의 데이터 탑입을 만들고 싶다면 (15 downto 0) 또는 (0 to 15)로 표현한다.)
- ❖ Slice는 Vector Data Type으로 표현된 Data Type의 일부분을 의미한다.
- ❖ (15 downto 0)의 일부분에만 특정 값이나 신호를 Assignment하려면 우측 예제 코드와 같이 Slice 하여 표현 할 수 있다.
- ❖ Slice하여 Assignment를 할 때에도 Data Type과 Bit Size가 같아야 한다.

```
library ieee;
use ieee.std_logic_1164.all;

entity ander3 is
port(
    a,b : in std_logic_vector(15 downto 0);
    result : out std_logic_vector(15 downto 0)
);
end ander3;

architecture structural of ander3 is
    signal c : std_logic_vector(15 downto 0);
begin
    c(15 downto 8) <= a(15 downto 8) and b(15 downto 8);
    c(7 downto 0) <= a(7 downto 0) and b(7 downto 0);
    result <= c;
end structural;
```

# Chapter 3 VHDL Basic Syntax

- Constant & Signal
- VHDL Data Types
- Bit Slice
- Operators
- Ultra96 Training Kit Exercises 1

# Arithmetic Operators

연산자	설명	예 ( a="1001", b="0011" )
+	더하기	$a + b = 12$
-	빼기	$a - b = 6$
*	곱하기	$a * b = 27$
/	나누기	$a / b = 3$
abs	절대값	$Abs(b-a) = 6$

# Relational Operators

연산자	설명	예 ( a="1001", b="0011" )
=	같다	a = b -- false
/=	다르다	a /= b -- true
<	작다	a < b -- false
>	크다	a > b -- true
<=	작거나 같다	a <= b -- false
>=	크거나 같다	a >= b -- true

# Logical Operators

연산자	설명	예 ( a="1001", b="0011" )
and	and	a and b = "0001"
or	or	a or b = "1011"
not	not	not a = "0110"
nand	nand	a nand b = "1110"
nor	nor	a nor b = "0100"
xor	xor	a xor b = "1010"
xnor	xnor	a xnor b = "0101"

# Concatenation Operator

- ❖ 표현된 식을 연결해주는 연산자이다.
- ❖  $a = "1001"$ ,  $b = "0011"$  일 때  $a \& b = "10010011"0$  된다.

```
library ieee;
use ieee.std_logic_1164.all;

entity ander4 is
    port(
        a,b : in std_logic_vector(15 downto 0);
        result : out std_logic_vector(15 downto 0)
    );
end ander4;

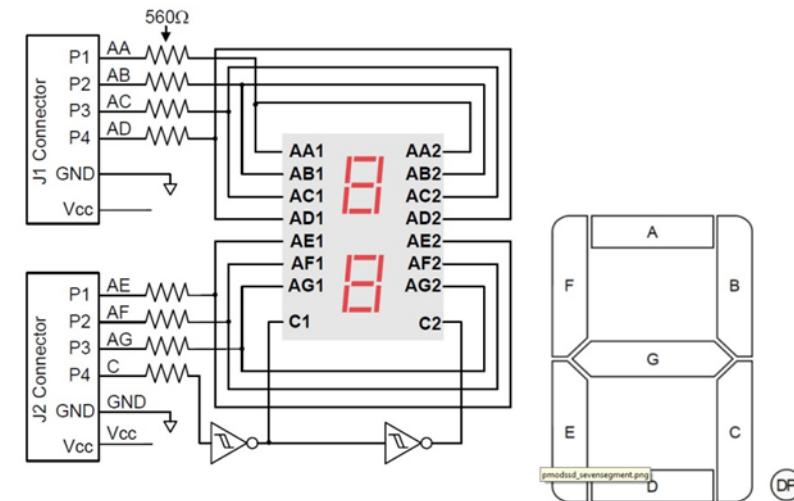
architecture structural of ander4 is
    signal c : std_logic_vector(7 downto 0);
    signal d : std_logic_vector(7 downto 0);
begin
    c <= a(15 downto 8) and b(15 downto 8);
    d <= a(7 downto 0) and b(7 downto 0);
    result <= c & d;
end structural;
```

# Chapter 3 VHDL Basic Syntax

- Constant & Signal
- VHDL Data Types
- Bit Slice
- Operators
- Ultra96 Training Kit Exercises 1

# Ultra96 Training Kit Exercises 1

- ❖ EX1-1 Pmod 8LD와 Pmod BTN을 활용하여 4개의 버튼이 각각 하나의 LED를 제어하도록 프로그래밍 하시오.
- ❖ EX1-2 Pmod BTN의 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 Pmod SSD가 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 Pmod SSD 한 개를 연결하여 7-Segment에 특정 숫자 또는 문자가 표시되도록 프로그래밍 하시오.

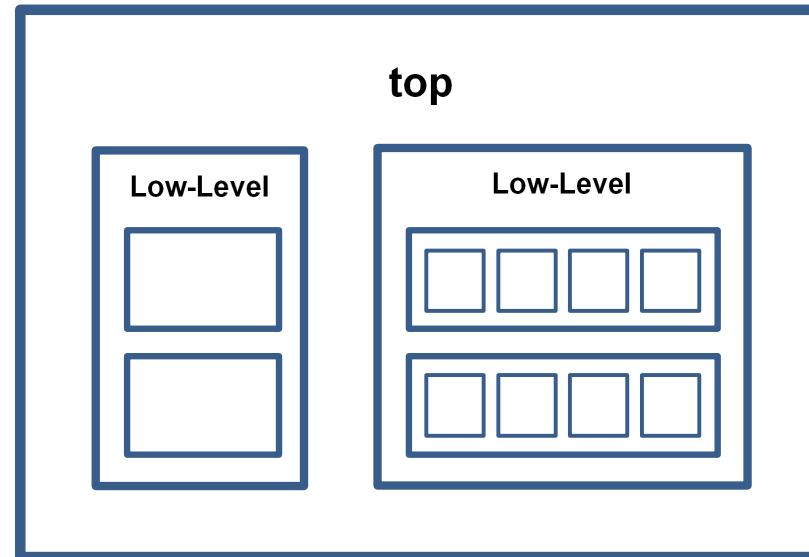


# Chapter 4 Instantiation

- 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

```
architecture 아키텍처이름 of 상위레벨엔터티이름 is
  component 하위레벨엔터티이름 is
    port(
      하위레벨포트리스트
    );
  end component;
begin
  인스턴스이름 : 하위레벨엔터티이름 port map(
    포트연결
  );
end 아키텍처이름;
```

- ❖ 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.
- ❖ Instantiation은 선언부와 구현부 두 부분으로 나누어진다. (※ 선언부에서 컴포넌트를 선언한 후 구현부에서 선언된 컴포넌트를 인스턴스로 생성한다.)

# Port Connection by Order

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

```
library ieee;
use ieee.std_logic_1164.all;

entity top is
    port( top_a,top_b : in std_logic_vector(3 downto 0);
          top_result : out std_logic_vector(3 downto 0));
end top;

architecture structural of top is
    component ander is
        port(a,b : in std_logic;
             result : out std_logic);
    end component;
begin
    ander_inst_0 : ander port map (top_a(0), top_b(0), top_result(0));
    ander_inst_1 : ander port map (top_a(1), top_b(1), top_result(1));
    ander_inst_2 : ander port map (top_a(2), top_b(2), top_result(2));
    ander_inst_3 : ander port map (top_a(3), top_b(3), top_result(3));
end structural;
```

# Port Connection by Name

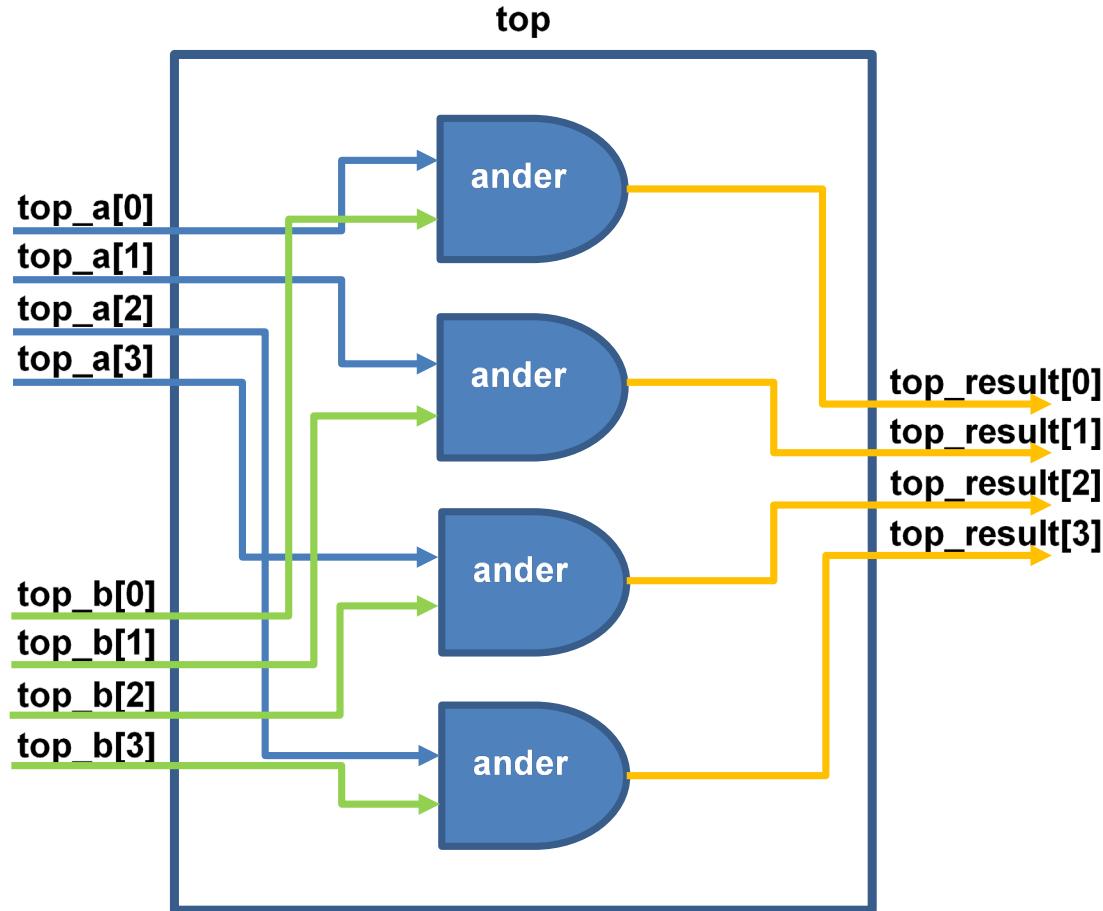
❖ 이름에 의한 포트 연결 방식은 하위레벨 모듈의 포트이름과 상위레벨 모듈의 넷을 명시하여 각각이 연결되도록 표현하는 방식이다.

```
library ieee;
use ieee.std_logic_1164.all;

entity top2 is
    port( top_a,top_b : in std_logic_vector(3 downto 0);
          top_result : out std_logic_vector(3 downto 0));
end top2;

architecture structural of top2 is
    component ander is
        port(a,b : in std_logic;
             result : out std_logic);
    end component;
begin
    ander_inst_0 : ander port map (a=>top_a(0),b=>top_b(0),result=>top_result(0));
    ander_inst_1 : ander port map (a=>top_a(1),b=>top_b(1),result=>top_result(1));
    ander_inst_2 : ander port map (a=>top_a(2),b=>top_b(2),result=>top_result(2));
    ander_inst_3 : ander port map (a=>top_a(3),b=>top_b(3),result=>top_result(3));
end structural;
```

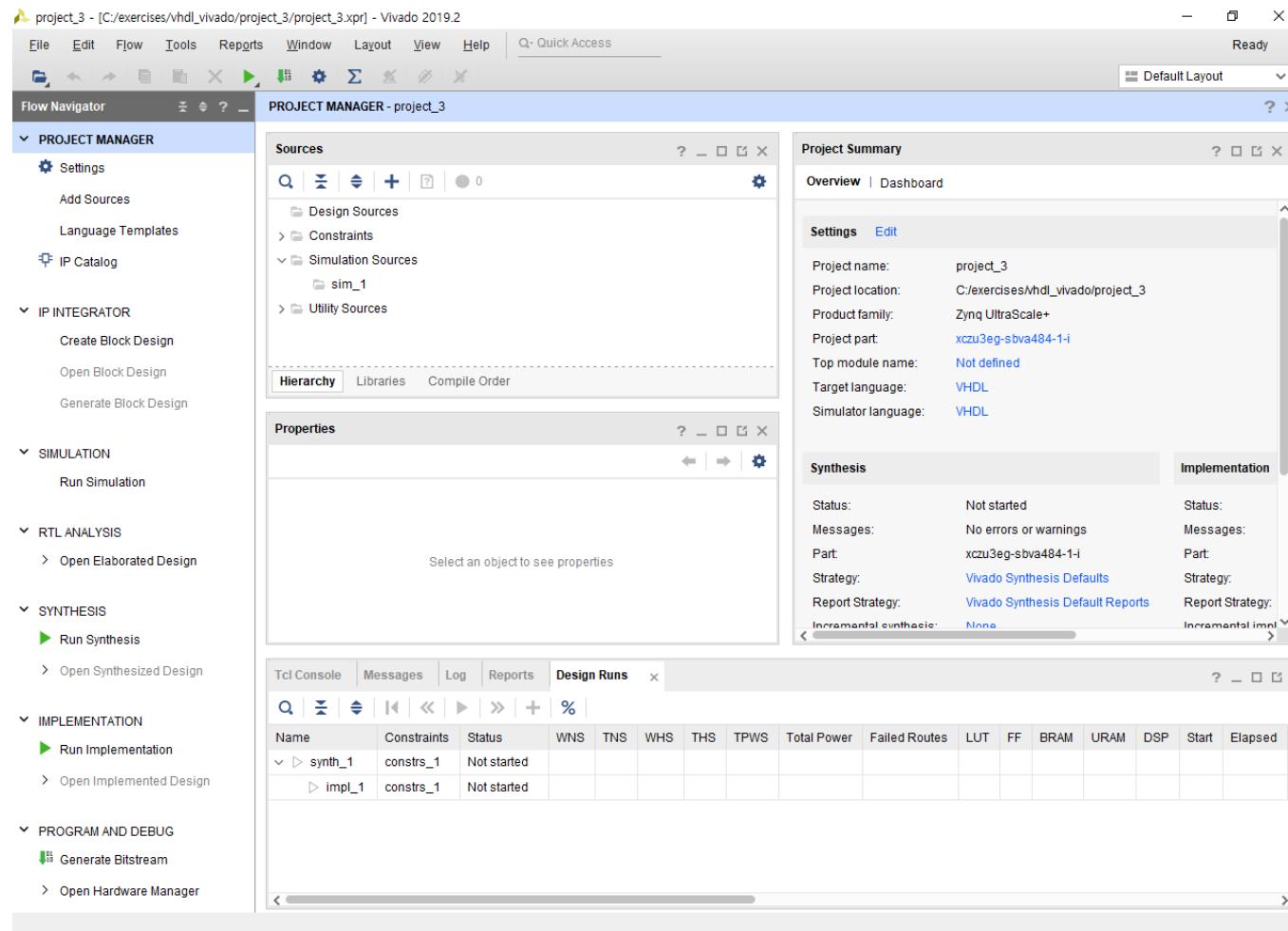
# Schematic for Instantiation Example



# Chapter 4 Instantiation

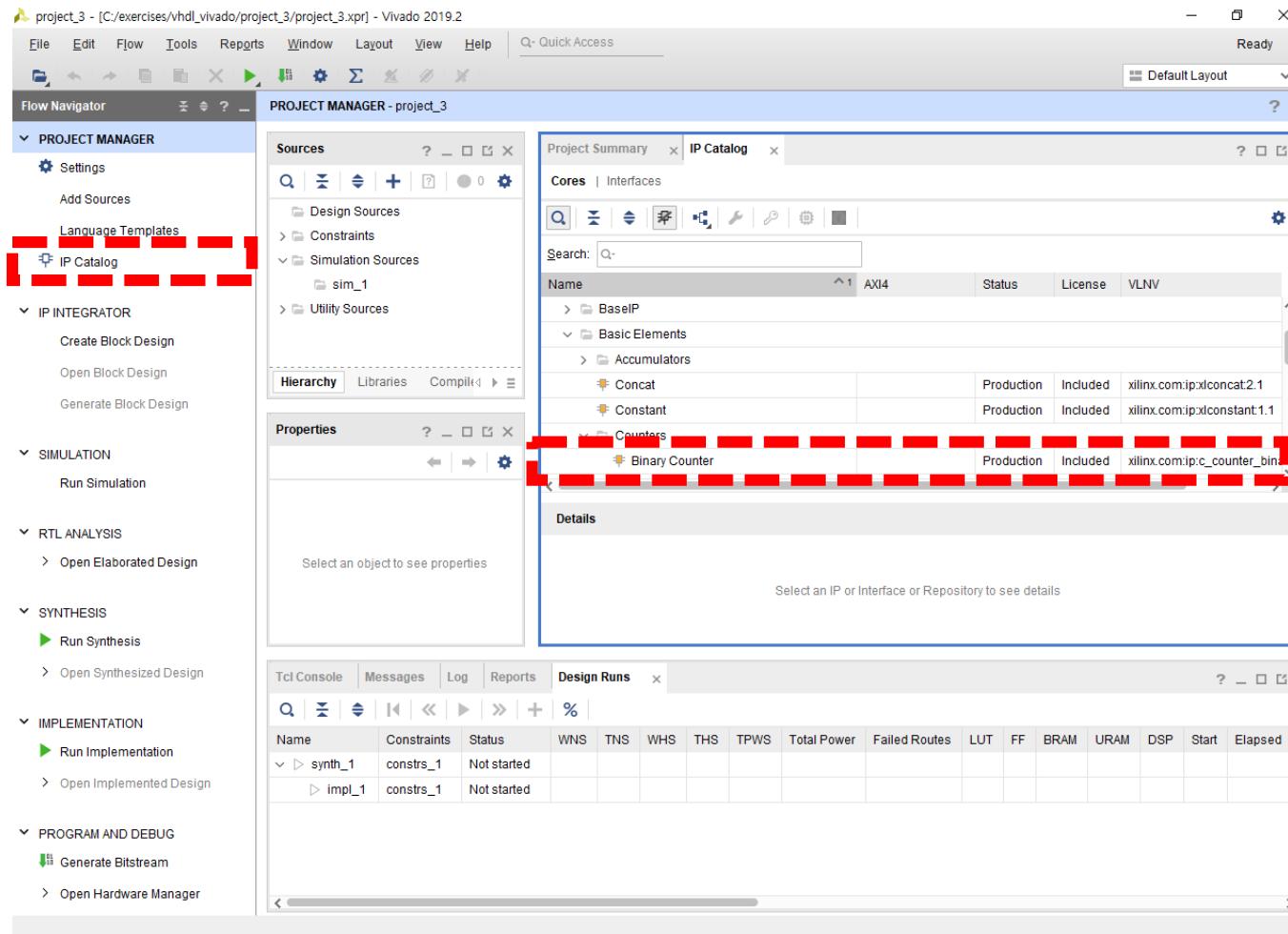
- Hierarchical Design
- Instantiation
- IP Core Instantiation using Vivado
- Clocking Wizard
- Ultra96 Training Kit Exercises 2

# Step 1 Creating Vivado Project



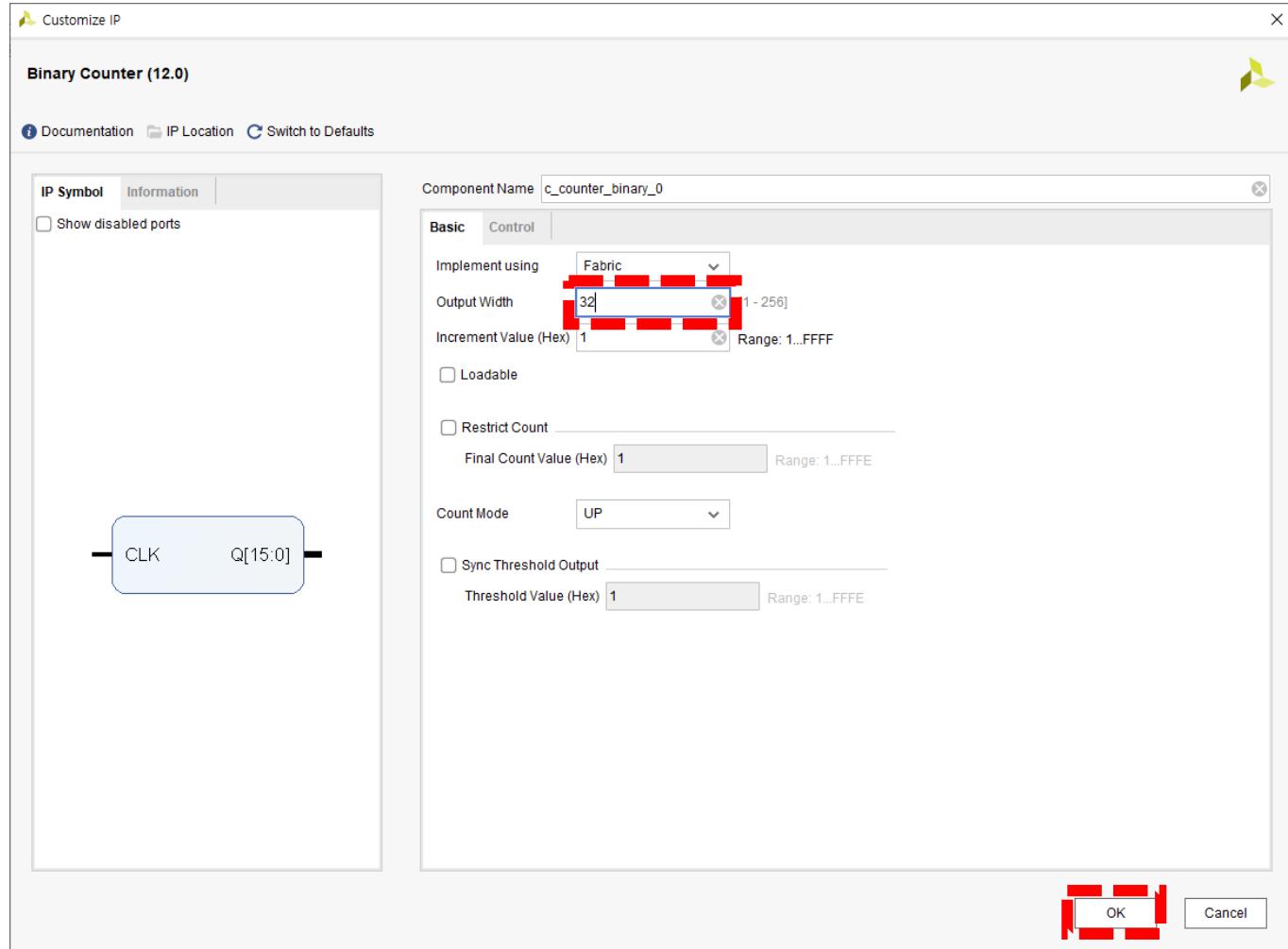
❖ Chapter 1의 Vivado 프로젝트 만들기를 참조하여 project\_3 프로젝트를 만든다.

# Step 2 Creating IP Core 1



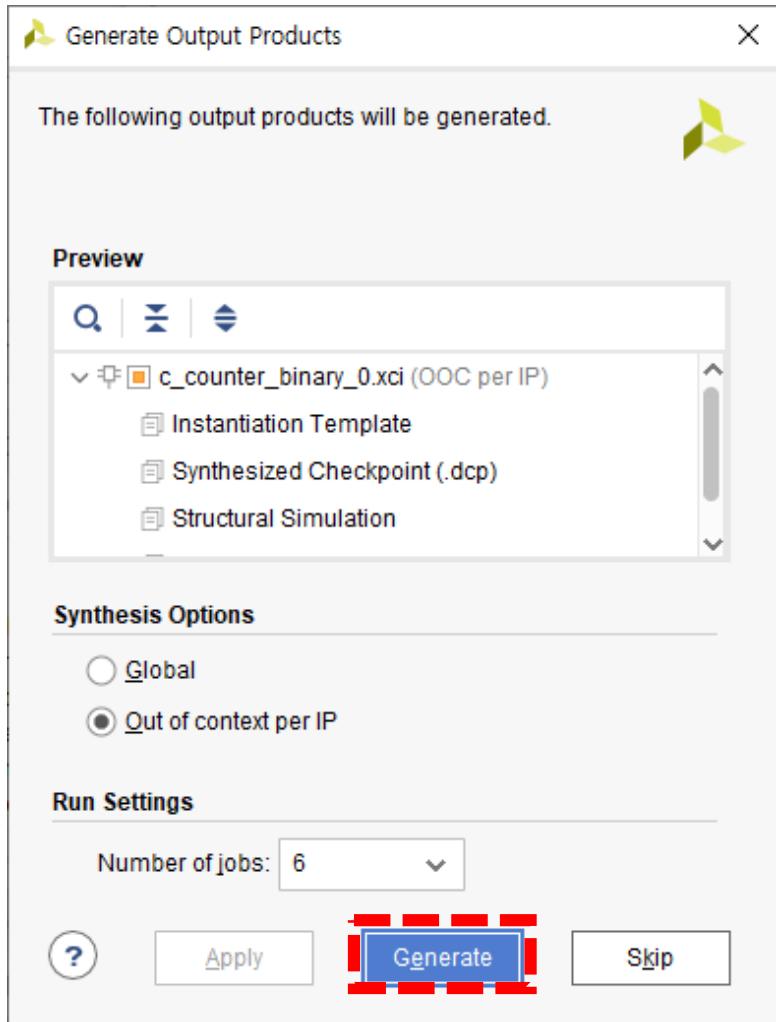
- ❖ Flow Navigator안에 Project Manager 아래 IP Catalog를 클릭한다.
- ❖ IP Catalog가 나오면 IP Catalog안에 Basic Elements ⇒ Counters ⇒ Binary Counter를 더블클릭 한다.

# Step 2 Creating IP Core 2



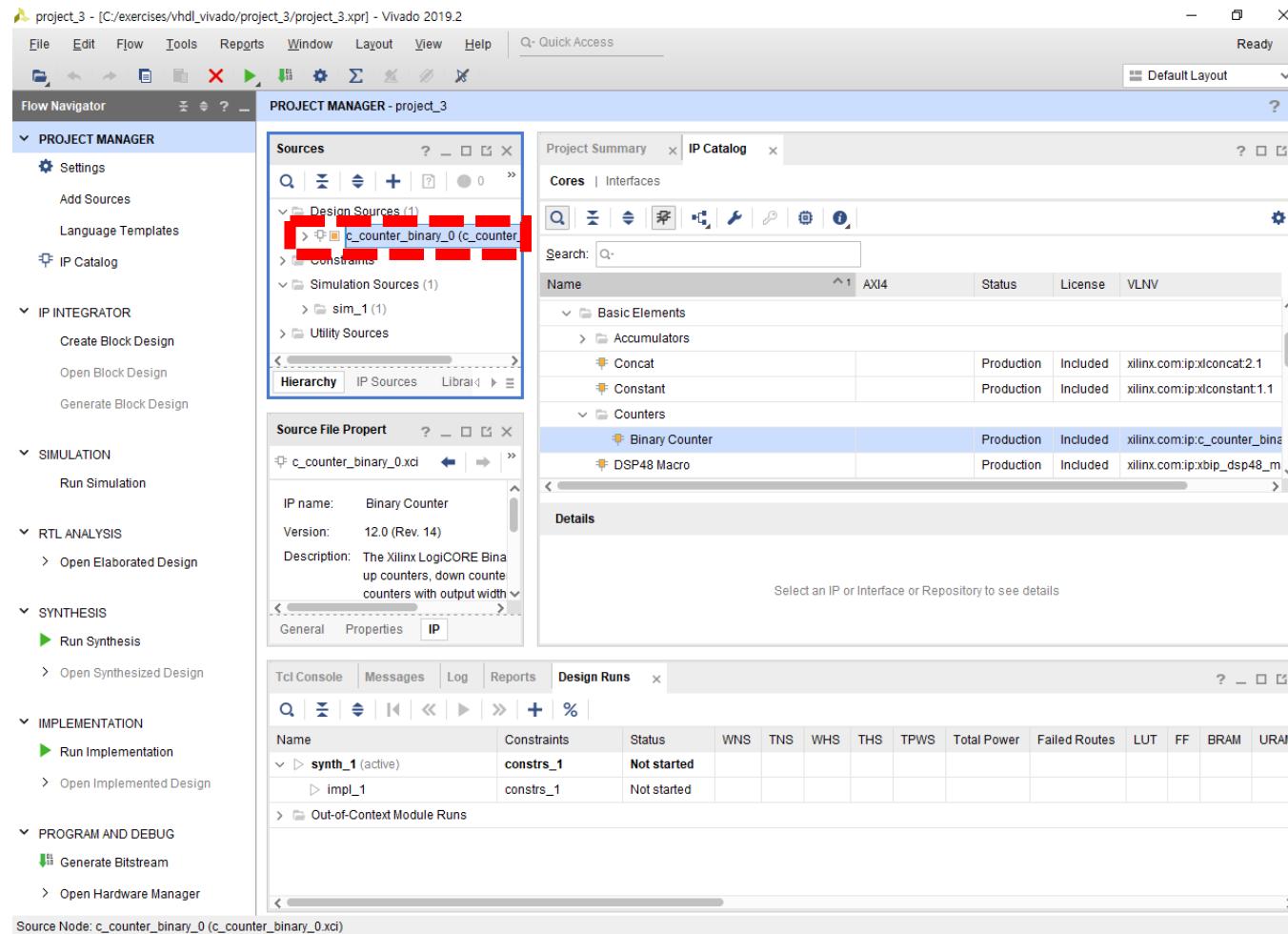
❖ Binary Counter IP를 Customize하는 Window가 나으면 Output Width를 32로 수정하고 OK버튼을 클릭한다.

# Step 2 Creating IP Core 3



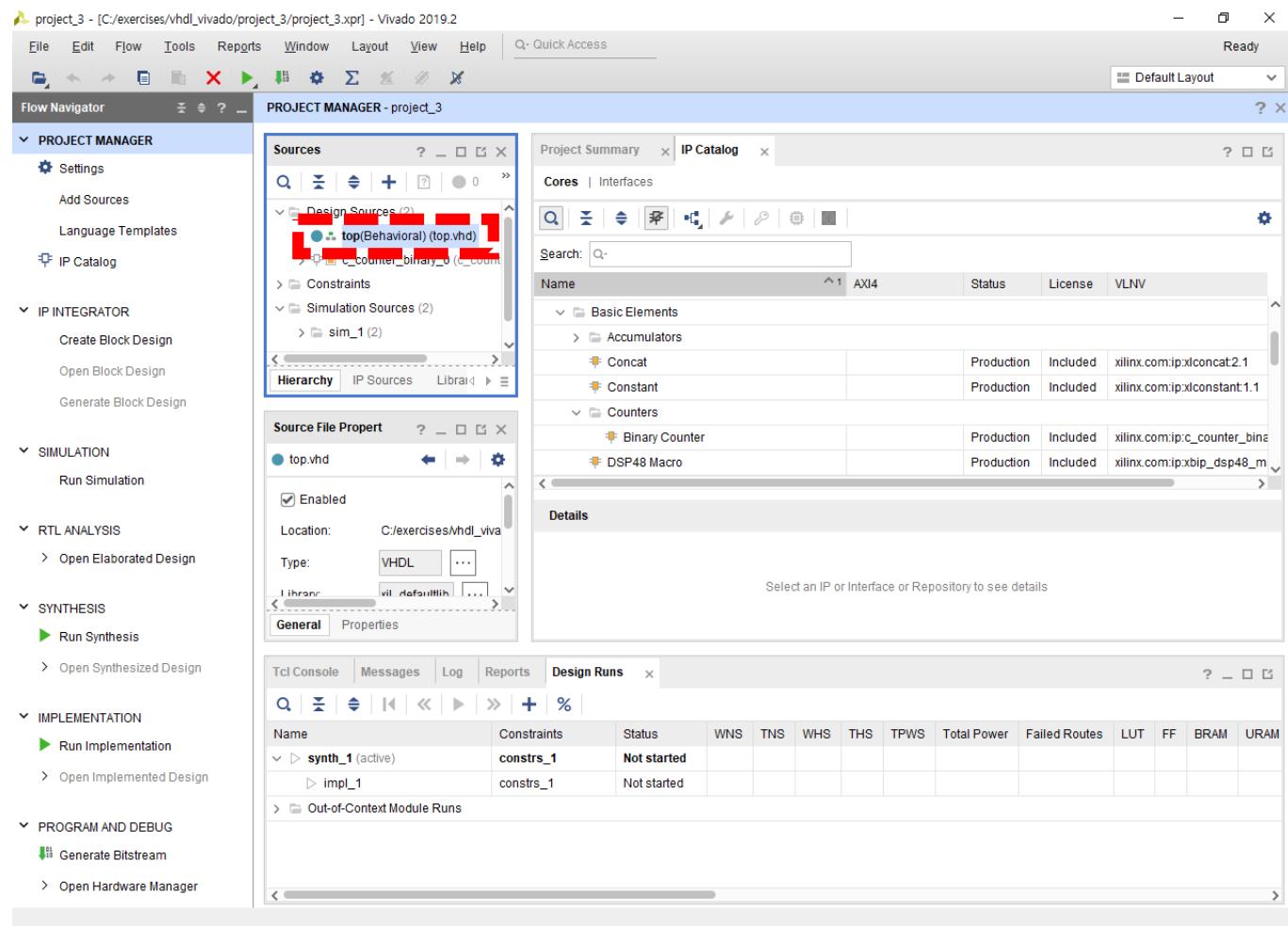
❖ Generate Output Products  
Window가 나오면 디폴트 그대로 두고 Generate 버튼을 클릭한다.

# Step 2 Creating IP Core 4



❖ Sources Window 안에 Design Sources를 보면  
c\_counter\_binary\_0로 Binary Counter IP가 생성된 것을 확인할 수 있다.

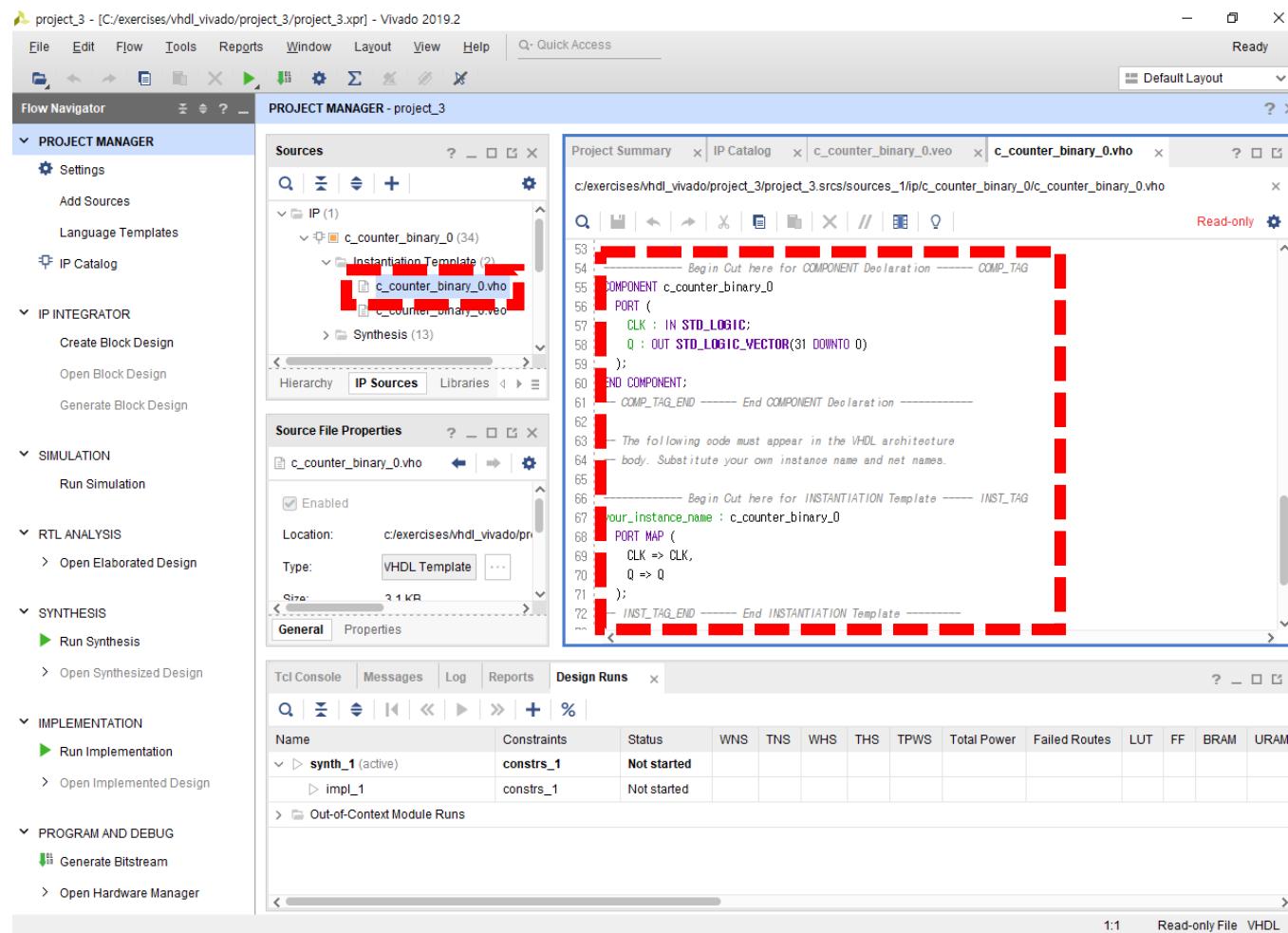
# Step 3 Add Top Module



## ❖ Chapter 2 Vivado Design

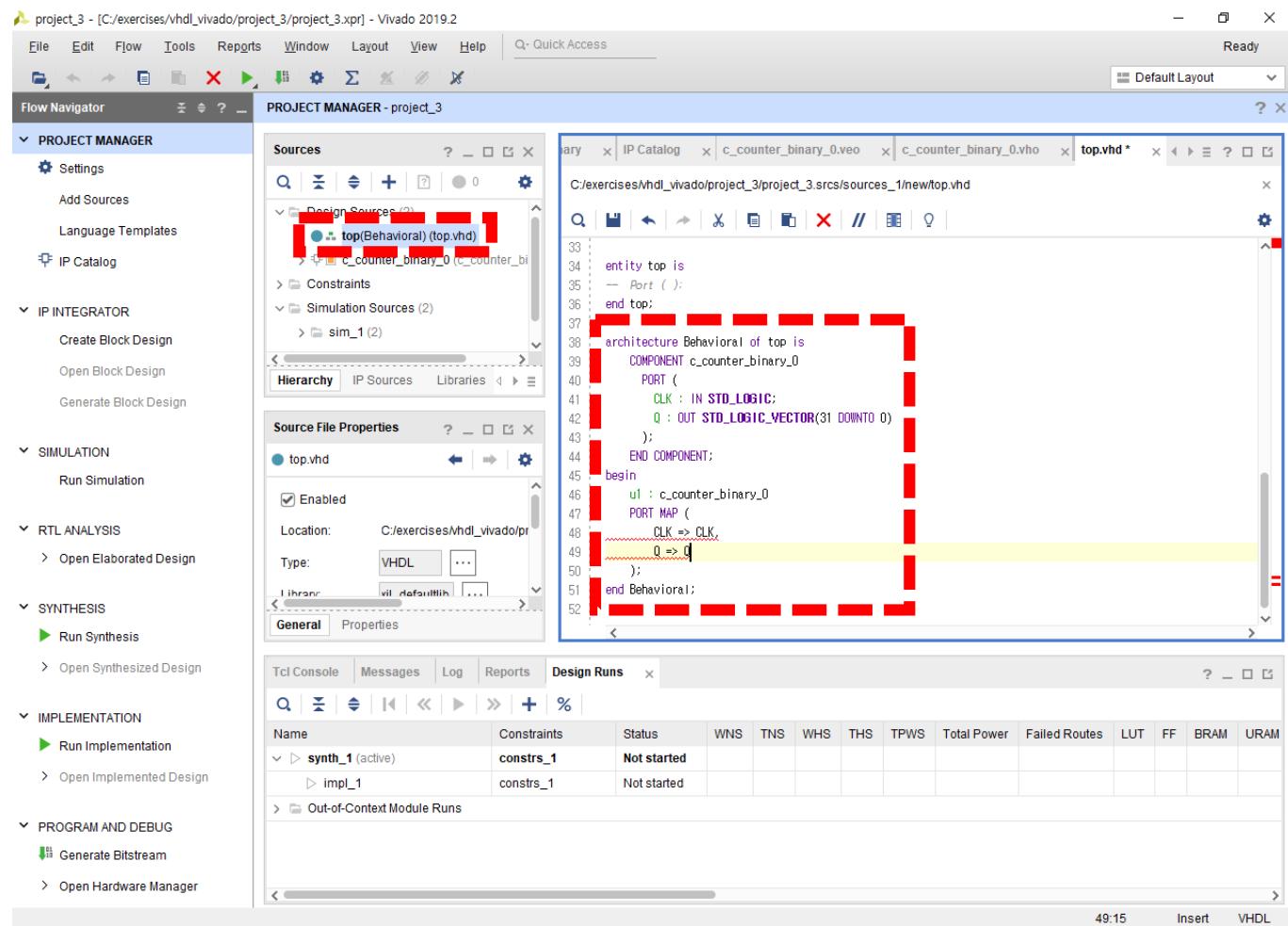
Flow의 Step 2에서 디자인 소스를 추가한 과정을 참고하여 top.vhd 소스파일을 추가한다.

# Step 4 IP Core Instantiation 1



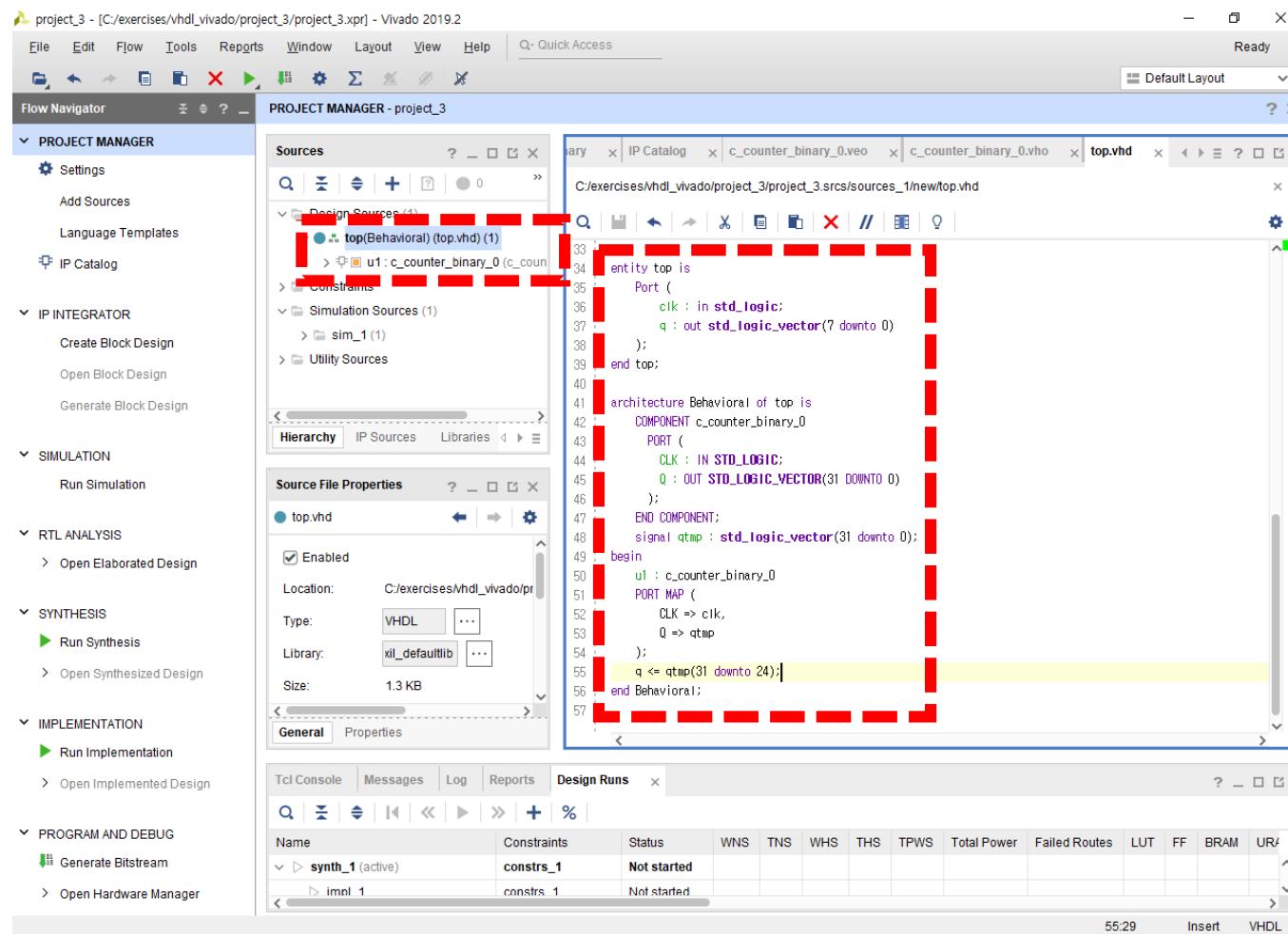
- ❖ Sources 창안에 IP Sources 탭을 클릭한 후 IP아래 **c\_counter\_binary\_0** ⇒ Instantiation Template ⇒ **c\_counter\_binary\_0.vho** 파일을 더블 클릭한다.
- ❖ **c\_counter\_binary\_0.veo** 파일 안에 주석처리 되지 않은 부분을 복사하여 top.vhd파일에 붙여 넣기 한다.

# Step 4 IP Core Instantiation 2



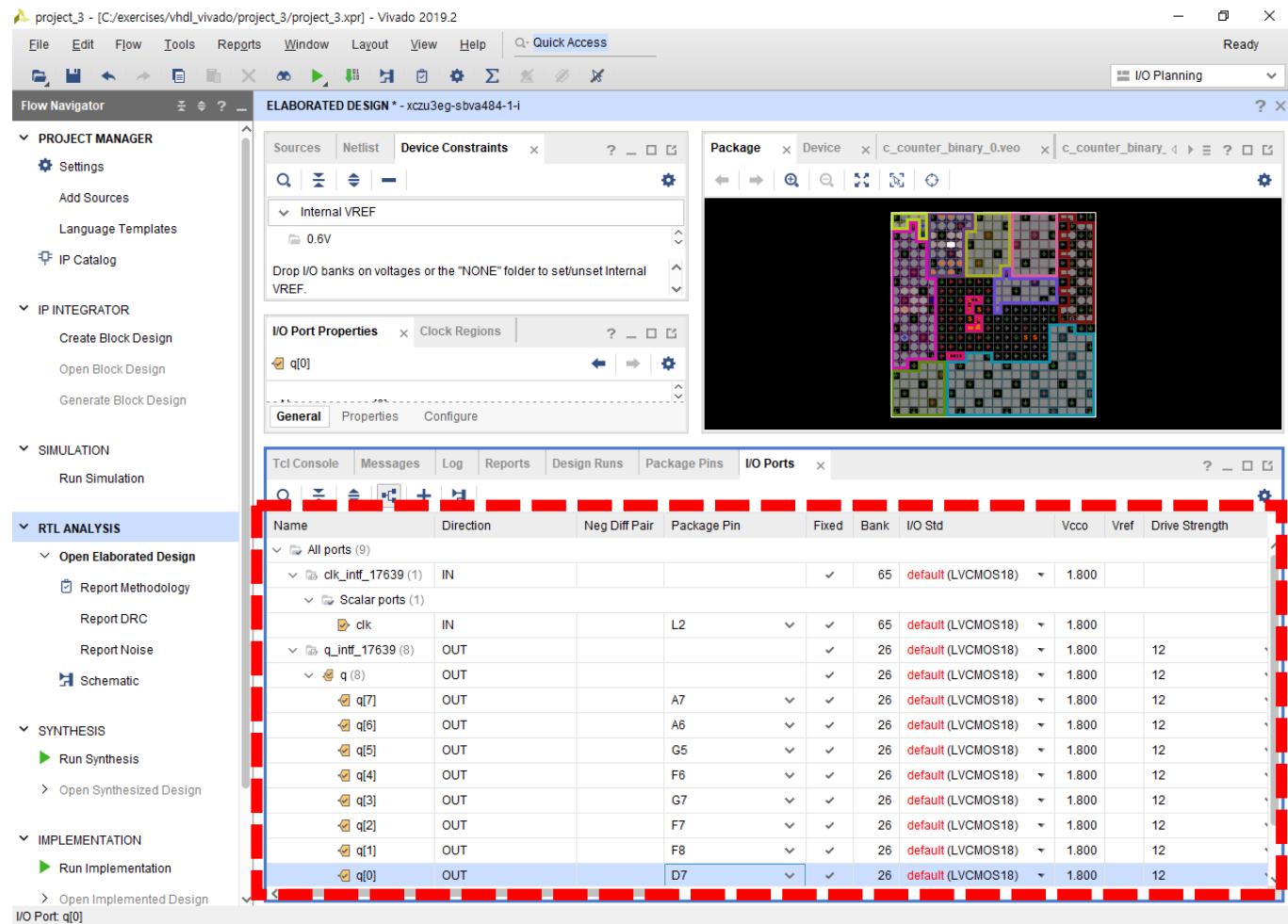
- ❖ Component 선언부는 architecture 와 begin사이에 붙여 넣기를 하고 Instantiation부분은 begin과 end 사이에 붙여 넣기를 한다.
- ❖ 붙여 넣기 한 내용 중 your\_instance\_name을 u1로 수정한다. (※ 해당 인스턴스의 기능에 따라서 인스턴스 이름을 만들어주는 것이 가독성을 위해서 좋지만 여기서는 편의상 인스턴스 이름을 u1으로 수정하였다.)

# Step 5 Write Top Module Source Code



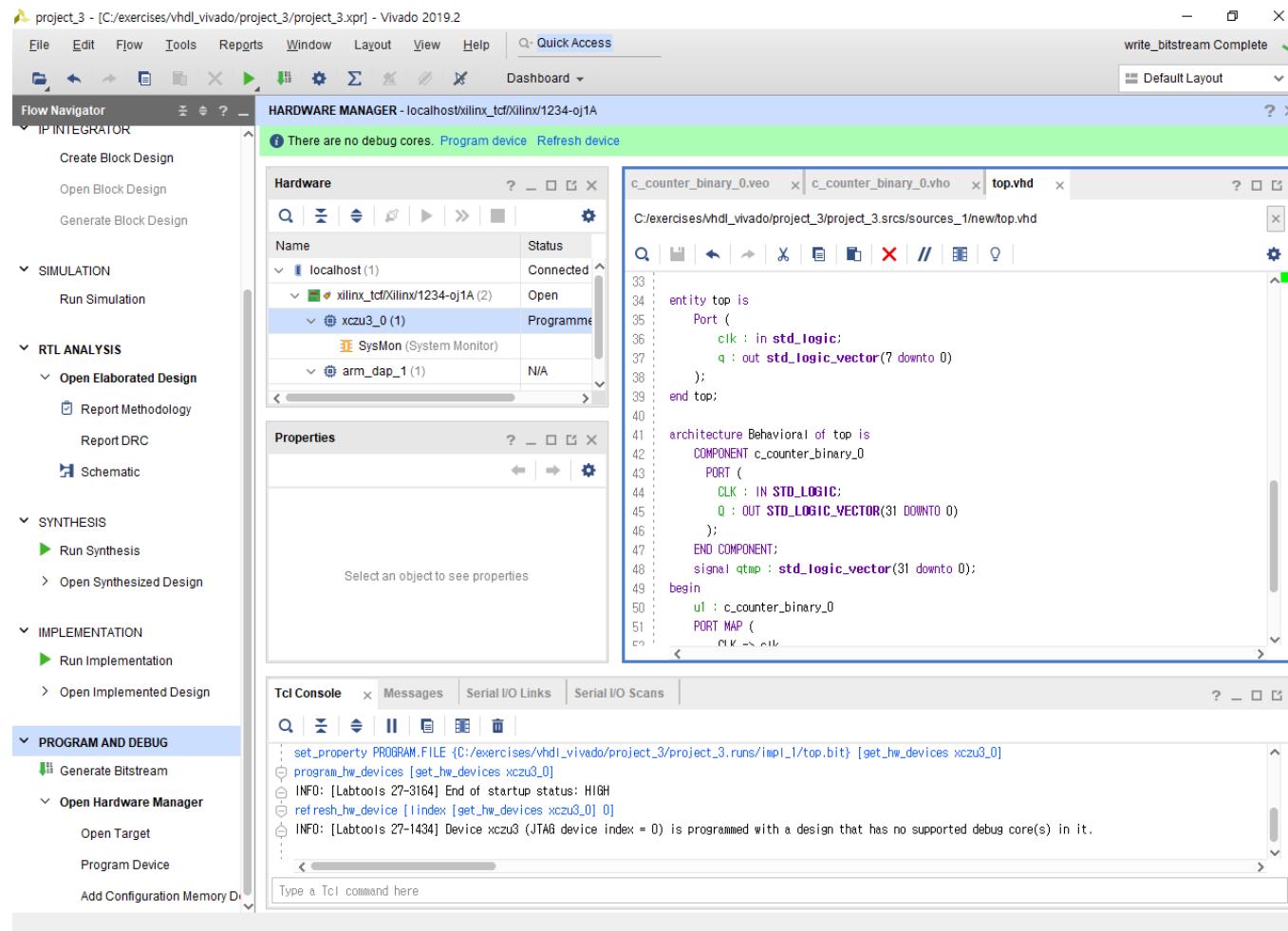
- ❖ top entity의 포트리스트에 clk와 q를 추가 한다.
- ❖ 포트선언부에 clk : in std\_logic; 와 q : std\_logic\_vector(7 downto 0)을 추가하고 선언부에 signal qtmp : std\_logic\_vector(31 downto 0);를 선언한다.
- ❖ Counter IP Core Instantiation을 위해 포트를 clk와 qtmp로 수정하여 연결하고 qtmp의 상위 8-bit를 q에 assign 한다.
- ❖ top모듈을 저장하면 Sources 창에 생성한 Binary 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.

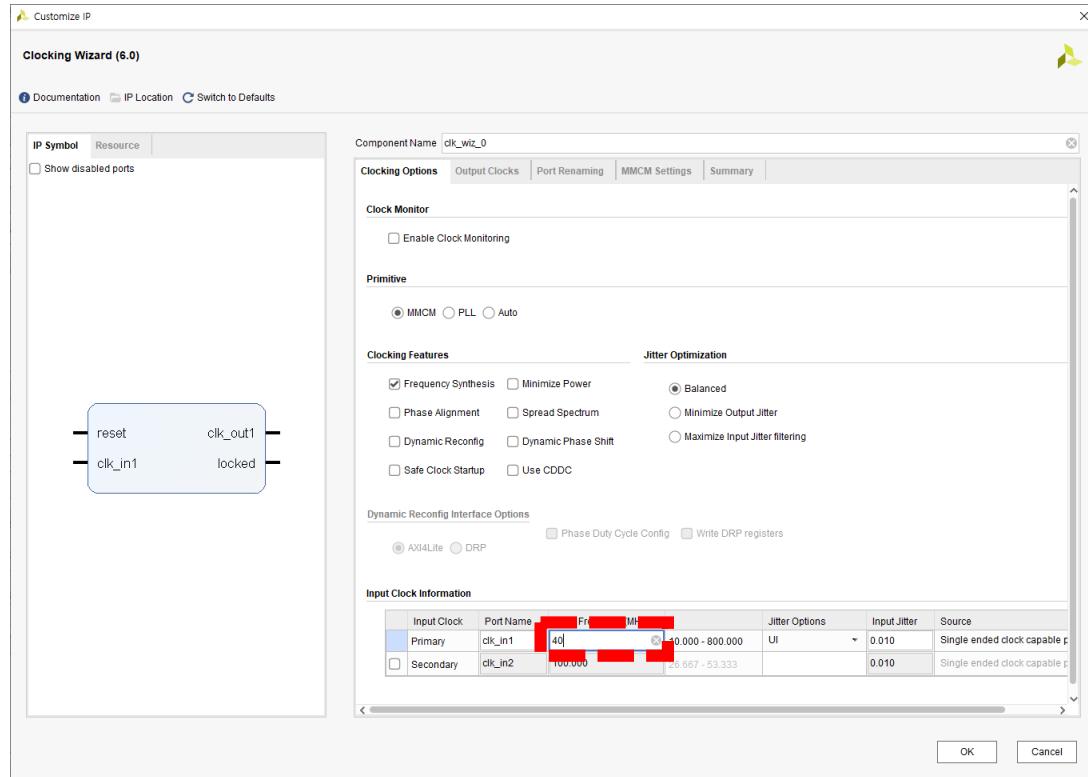
# Chapter 4 Instantiation

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

# Clocking Wizard 1

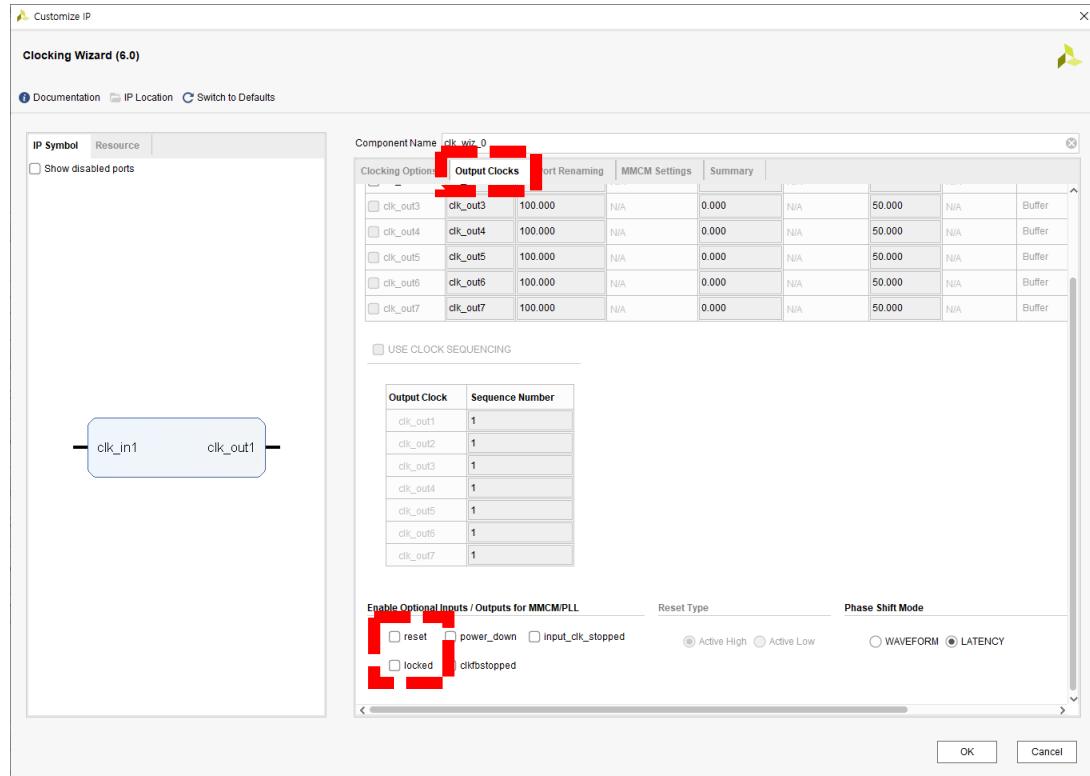
- ❖ 디지털 회로를 설계할 때 필요한 clock signal을 모두 FPGA 외부에서 공급받아 사용할 수는 없다.
- ❖ Xilinx에서 제공하는 Clocking Wizard IP Core를 사용하면 다양한 주파수, 위상, duty cycle을 가지는 clock signal을 클럭을 만들어 낼 수 있다.
- ❖ Clocking Wizard는 Mixed-mode clock manager(MMCM)나 phase-locked loop(PLL) primitive를 사용하여 클럭 생성 회로를 만들어낸다.
- ❖ Xilinx 디바이스를 사용할 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 Pmod 8LD를 PMOD\_B 커넥터의 윗줄에 Pmod BTN을 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

- **Process**
- **Behavioral Statements**
- **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.
- ❖ Process밖에 있는 Statement들은 Concurrent Statement이고 Process안에 있는 Statement들은 Sequential Statement이다.

# Process

프로세스라벨(옵션) : **process**(Sensitivity List)

프로세스 선언부

**begin**

프로세스 구현부

**end process** 프로세스라벨(옵션);

- ❖ 프로세스라벨은 여러 개의 프로세스를 사용했을 때 프로세스를 구분해 주기 위해 사용한다.  
(※ 프로세스 라벨은 옵션으로 생략 가능하여 일반적으로 쓰지 않는다.)
- ❖ Process의 괄호 안에 있는 Sensitivity List는 우리말로 하면 감지 목록으로 Sensitivity List에 포함된 신호들을 감지하고 있다가 신호가 바뀌면 Process를 실행한다는 의미이다. (※ 일반적으로 Sensitivity List에는 Process에서 입력으로 사용하는 신호들을 모두 넣어준다.)
- ❖ 프로세스 선언부에는 Variable을 선언하여 Process안에서 사용할 수 있다.
- ❖ 프로세스 구현부에는 Sequential Statement를 사용하여 프로세스의 동작을 구현한다.

# Signal vs Variable

- ❖ Signal은 실제 회로를 연결하는데 사용하는 와이어(Wire) 같은 역할을 하지만 Variable은 코딩 할 때 필요한 임시 저장 장소와 같은 역할을 한다.
- ❖ Signal은 Architecture 선언부에서 선언하여 Architecture내에서 모두 사용할 수 있지만 Variable은 Process 선언부에서 선언하므로 Variable을 선언한 Process안에서만 사용 가능하다.
- ❖ Signal은 `<=` 를 사용하여 Assignment를 하지만 Variable은 Assignment를 위해 `:=` 를 사용한다.
- ❖ Signal에 Assignment를 하면 Combinational Circuit인 경우에는 바로 Value가 Update되지만 Sequential Circuit인 경우에는 1 Clock Cycle 후에 Update 된다.
- ❖ Variable에 Assignment를 하면 항상 바로 Value가 Update된다.

# Process Example

- ❖ 우측의 코드 예제는 Chapter 3에서 사용한 ander3 또는 ander4 모듈을 Process를 사용하여 표현한 것으로 Chapter 3의 ander3 또는 ander4 모듈과 동일한 기능을 하는 모듈이다.
- ❖ 우측의 코드 예제에서는 하나의 Process만 사용하였지만 여러 개의 Process 사용이 가능하면 여러 개를 사용할 경우 각각의 Process는 Concurrent한 것으로 해석하고 Process 안의 Statement들은 Sequential한 것으로 해석한다.

```
library ieee;
use ieee.std_logic_1164.all;

entity ander5 is
port(
    a,b : in std_logic;
    result : out std_logic
);
end ander5;

architecture behavioral of ander5 is
begin
process_ex : process(a,b)
    variable temp : std_logic;
begin
    temp := a and b;
    result <= temp;
end process process_ex;
end behavioral;
```

# Chapter 5 Behavioral Modeling

- Process
- Behavioral Statements
- Ultra96 Training Kit Exercises 3

# Behavioral Statements

- ❖ Behavioral Modeling을 위해 순차적으로 해석되어 지는 Procedural Block안에 코딩하는 Statement를 의미하며 앞서 이야기한 Sequential Statement와 같은 의미이다.
- ❖ VHDL에는 If, Case, Loop와 같은 Behavioral Statements가 있다.

# If Statement

- ❖ if Statement는 if안에 if를 중첩하여 사용하는 것이 가능하다.
- ❖ 여러 개의 표현을 if의 조건으로 사용할 때 and 또는 or 논리 연산자를 사용한다.
- ❖ if Statement를 사용할 때는 else Statement를 사용하여 모든 조건에 대해서 기술해 주는 것이 좋다.

```
process(a,b,c,d,sel)
begin
    if sel="00" then
        y <= a;
    elsif sel="01" then
        y <= b;
    elsif sel="10" then
        y <= c;
    else
        y <= d;
    end if;
end process;
```

# 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.)

```
process(a,b,c,d,sel)
begin
    case sel is
        when "00" =>
            y <= a;
        when "01" =>
            y <= b;
        when "10" =>
            y <= c;
        when others =>
            y <= d;
    end case;
end process;
```

# Infinite Loop Statement

- ❖ Infinite Loop는 exit statement의 조건이 거짓이면 계속 루프를 돌다가 조건이 참이 되면 루프를 빠져나오도록 동작한다.
- ❖ 루프라벨은 프로세스라벨과 같이 여러 개의 루프를 사용할 때 구분하기 위해서 사용한다. (※ 프로세스라벨과 같이 루프라벨도 옵션이어서 생략 가능하기 때문에 일반적으로 잘 쓰이지 않는다.)

```
process(din)
    variable i : integer;
begin
    i := 0;
    L1 : loop
        dout(31-i) <= din(i);
        i := i + 1;
    exit L1 when i>31;
end loop L1;
end process;
```

# While & For Loop Statement

- ❖ While Loop는 While statement의 조건이 참이면 루프를 계속 돌다가 조건이 거짓이 되면 루프를 빠져나오도록 동작한다.
- ❖ For Loop는 For statement의 식별자가 range만큼 돌다가 루프를 빠져나오도록 동작한다.

```
process(din)
    variable i : integer;
begin
    i := 0;
    L2 : while i<=31 loop
        dout(31-i) <= din(i);
        i := i + 1;
    end loop L2;
end process;
```

```
process(din)
    variable i : integer;
begin
    L3 : for i in 0 to 31 loop
        dout(31-i) <= din(i);
    end loop L3;
end process;
```

# Chapter 5 Behavioral Modeling

- Process
- Behavioral Statements
- 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 모듈을 Component로 선언하고 Instantiation에 필요한 신호들을 선언한다.
- ❖ 검증하고자 하는 모듈을 Instantiation 한다.
- ❖ 검증하고자 하는 모듈의 입력포트에 인가할 적절한 입력 신호들을 만들어 준다. (※ 모듈에 자극을 준다고 하여 Stimulus라고 부른다.)

```
library ieee;
use ieee.std_logic_1164.all;
entity ander_tb is
end ander_tb;
architecture Behavioral of ander_tb is
component ander is
port( a,b : in std_logic;
      result : out std_logic );
end component;
signal a,b,result : std_logic;
begin
  uut : ander port map(a=>a,b=>b,result=>result);
  Stimulus : process
  begin
    a<='0';  b<='0';  wait for 200 ns;
    a<='0';  b<='1';  wait for 200 ns;
    a<='1';  b<='1';  wait;
  end process;
end Behavioral;
```

# Stimulus with Sequential Statement

- ❖ Process 안에서 Sequential Statement를 사용하여 Stimulus를 줄 때 begin으로 시작하면 0초로 시작하여 end process까지 실행되고 다시 begin부터 다시 반복하는 형태로 실행된다.
- ❖ 우측 예제 코드는 Clock Signal을 주기 위한 Stimulus를 Process를 사용하여 구현한 예제이다.

```
architecture Behavioral of tb is
    signal clk : std_logic;
begin
    Stimulus : process
    begin
        clk<='0';
        wait for 10 ns;
        clk<='1';
        wait for 10 ns;
    end process;
end Behavioral;
```

# Stimulus with Concurrent Statement

- ❖ Process 밖에서 Concurrent Statement를 사용하여 Stimulus를 줄 때는 우측 코드 예제와 같이 after라는 keyword를 사용한다.
- ❖ After 뒤에 오는 시간은 절대 값으로 앞에 시간 값과 상관 없이 0초로부터 해당 시간 후에 Value가 Update된다.
- ❖ 우측 코드 예제 중 아래 코드는 Clock Signal을 Concurrent Statement를 사용하여 Stimulus를 준 예제이다.
- ❖ 우측 코드 예제와 같이 signal 선언부에 초기값을 주지 않으면 not clk 구문의 값이 무엇이 될지 모르기 때문에 Simulation을 실행하면 Clock Signal은 Unknown이 되므로 주의해야 한다.

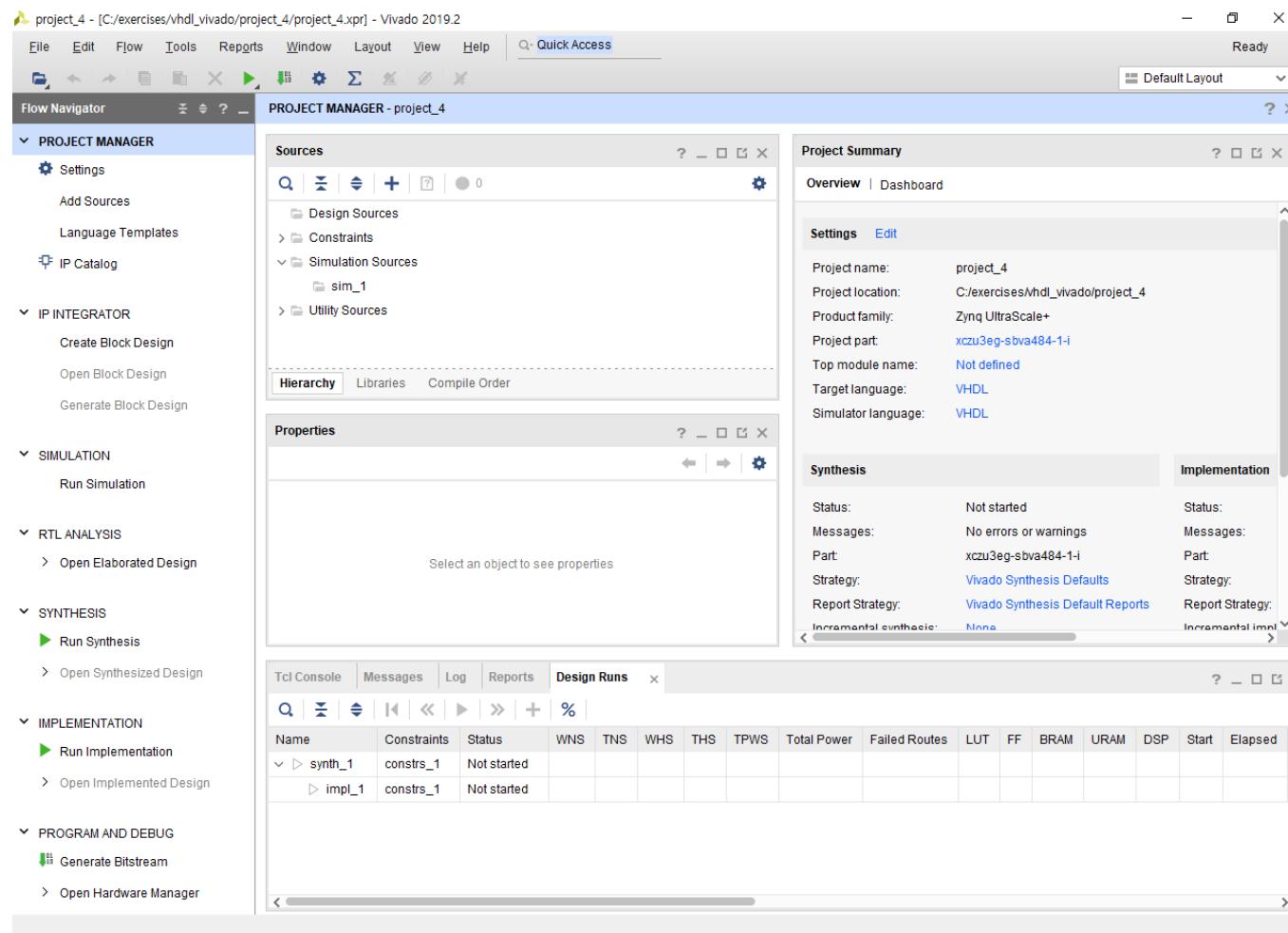
```
a <= '0', '1' after 400 ns;  
b <= '0', '1' after 200 ns;
```

```
architecture Behavioral of tb is  
    signal clk : std_logic := '0';  
begin  
    clk <= not clk after 10 ns;  
end Behavioral;
```

# Chapter 6 Simulation

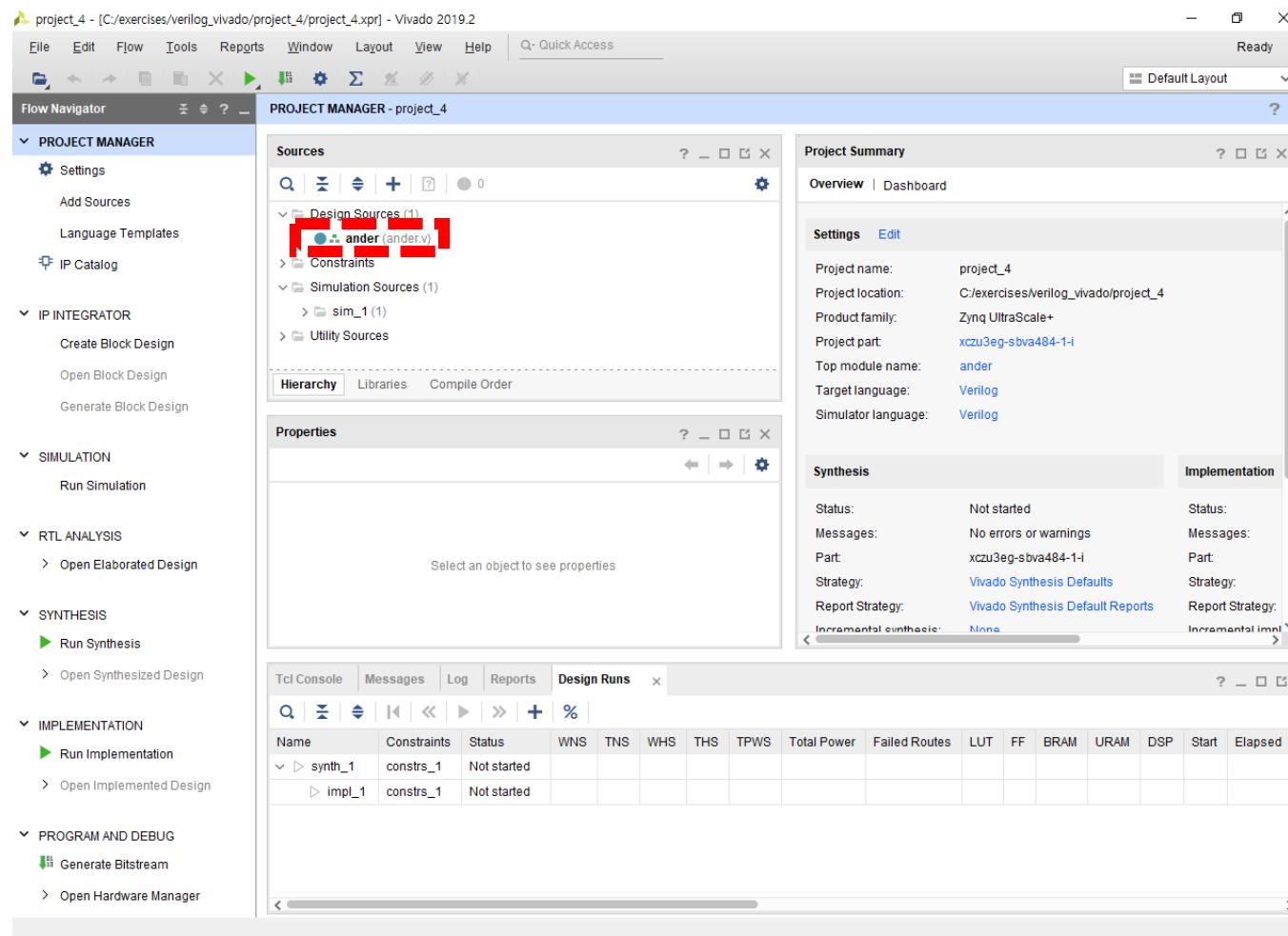
- Testbench
- Vivado Simulation
- Simulation Exercises

# Step 1 Creating Vivado Project



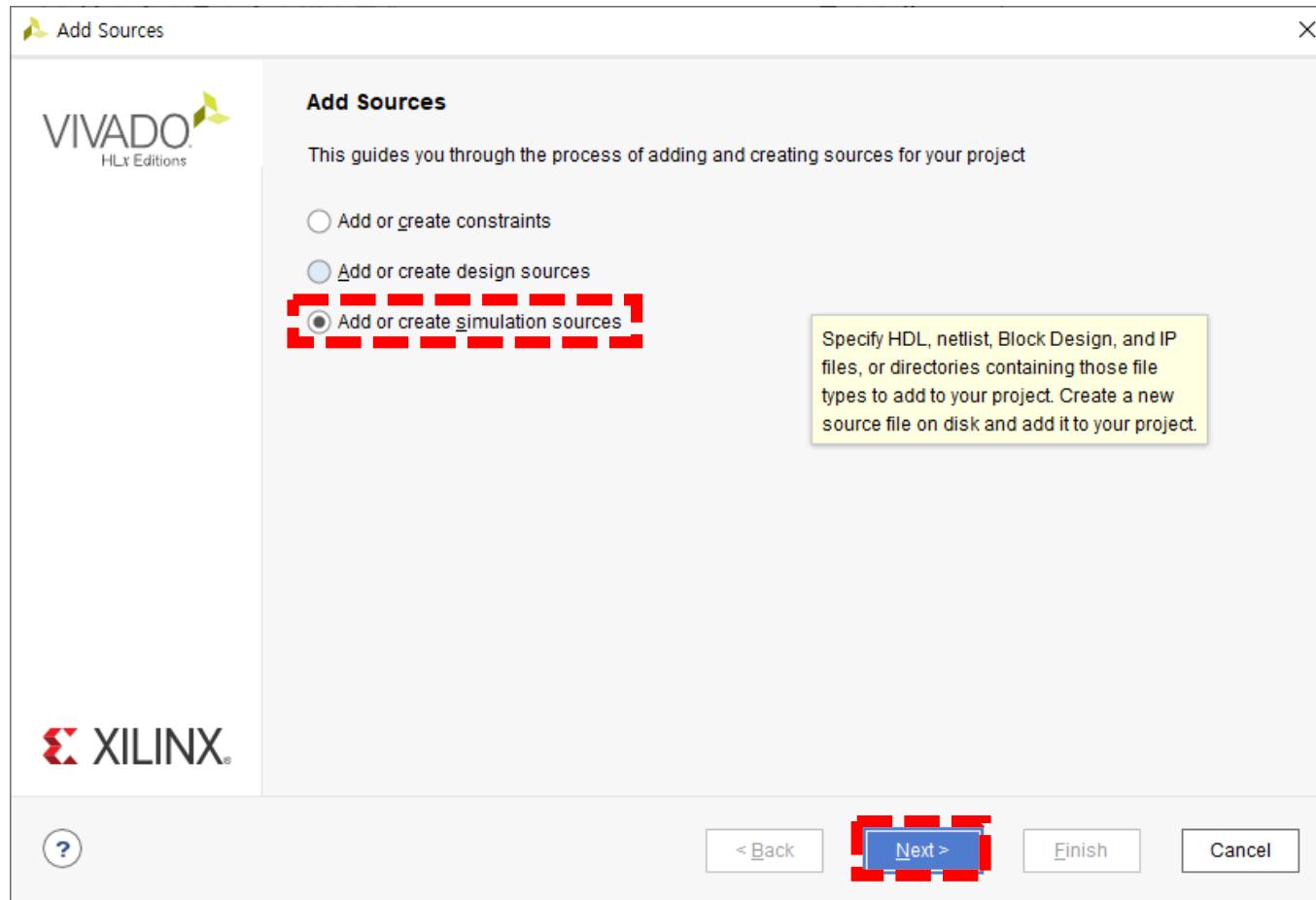
❖ Chapter 1의 Vivado 프로젝트 만들기를 참조하여 project\_4 프로젝트를 만든다.

# Step 2 Add Design Source in Vivado Project



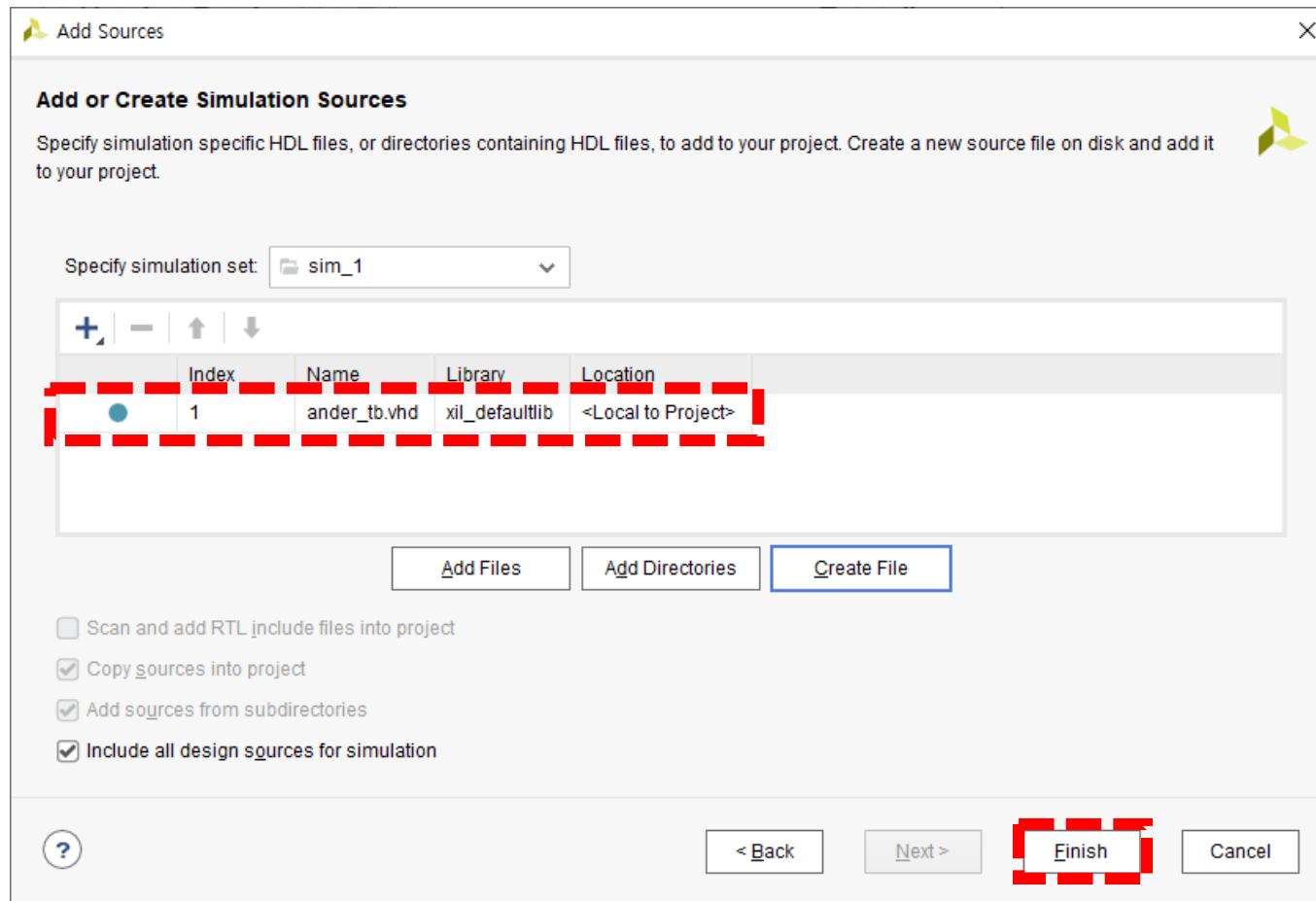
- ❖ Chapter 2 Vivado Design Flow의 Step 2를 참고하여 ander.vhd 소스파일을 프로젝트에 추가한다.
- ❖ ander.vhd파일에 Chapter 2의 ander 모듈 소스 코드를 코딩 한다.

# Step 2 Add Simulation Source in Vivado Project 1



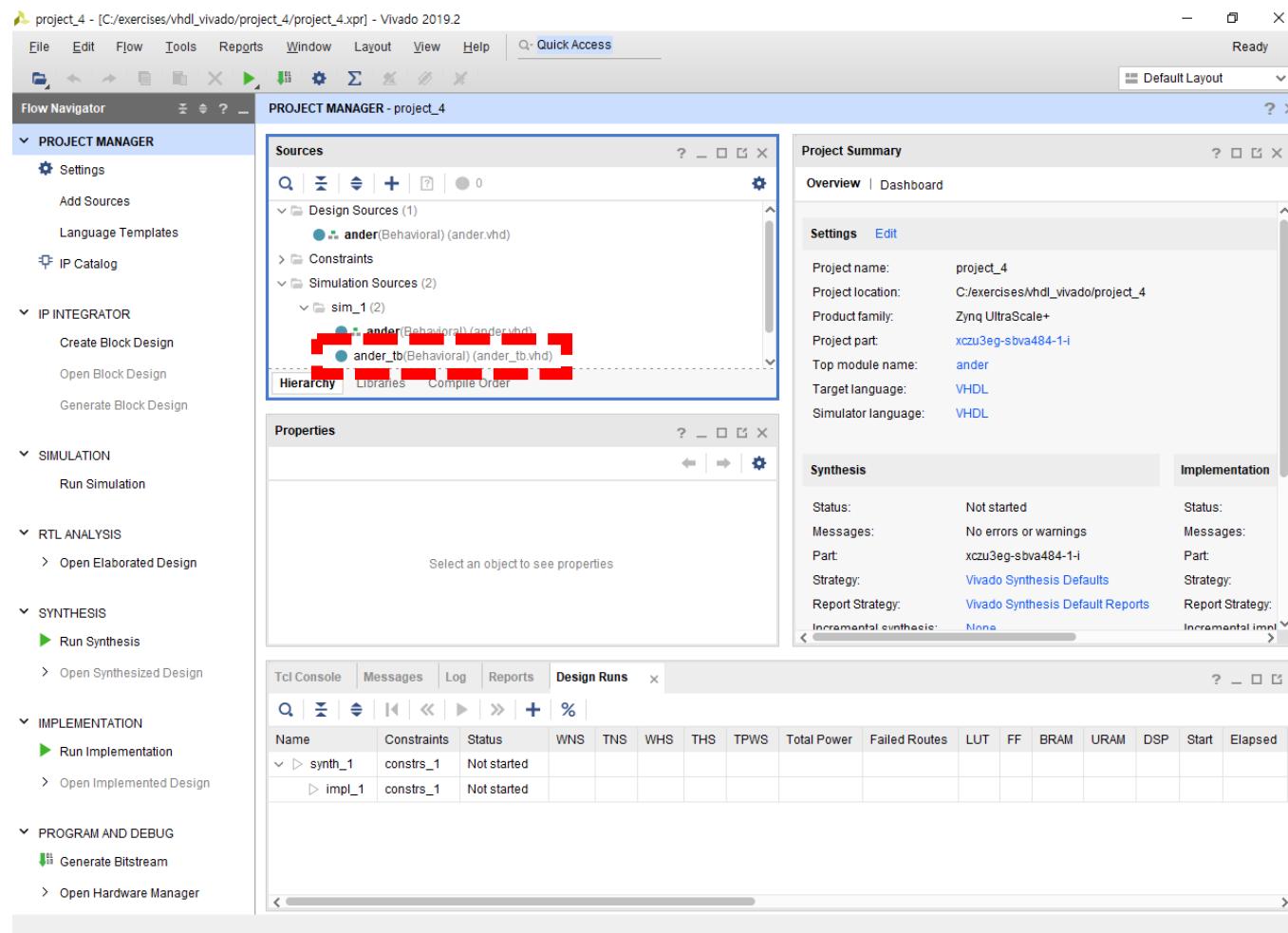
- ❖ Flow Navigator ⇒ Project Manager ⇒ Add Sources를 클릭한다.
- ❖ Add or create simulation sources를 클릭하여 선택한 후 Next 버튼을 클릭한다.

# Step 2 Add Simulation Source in Vivado Project 2



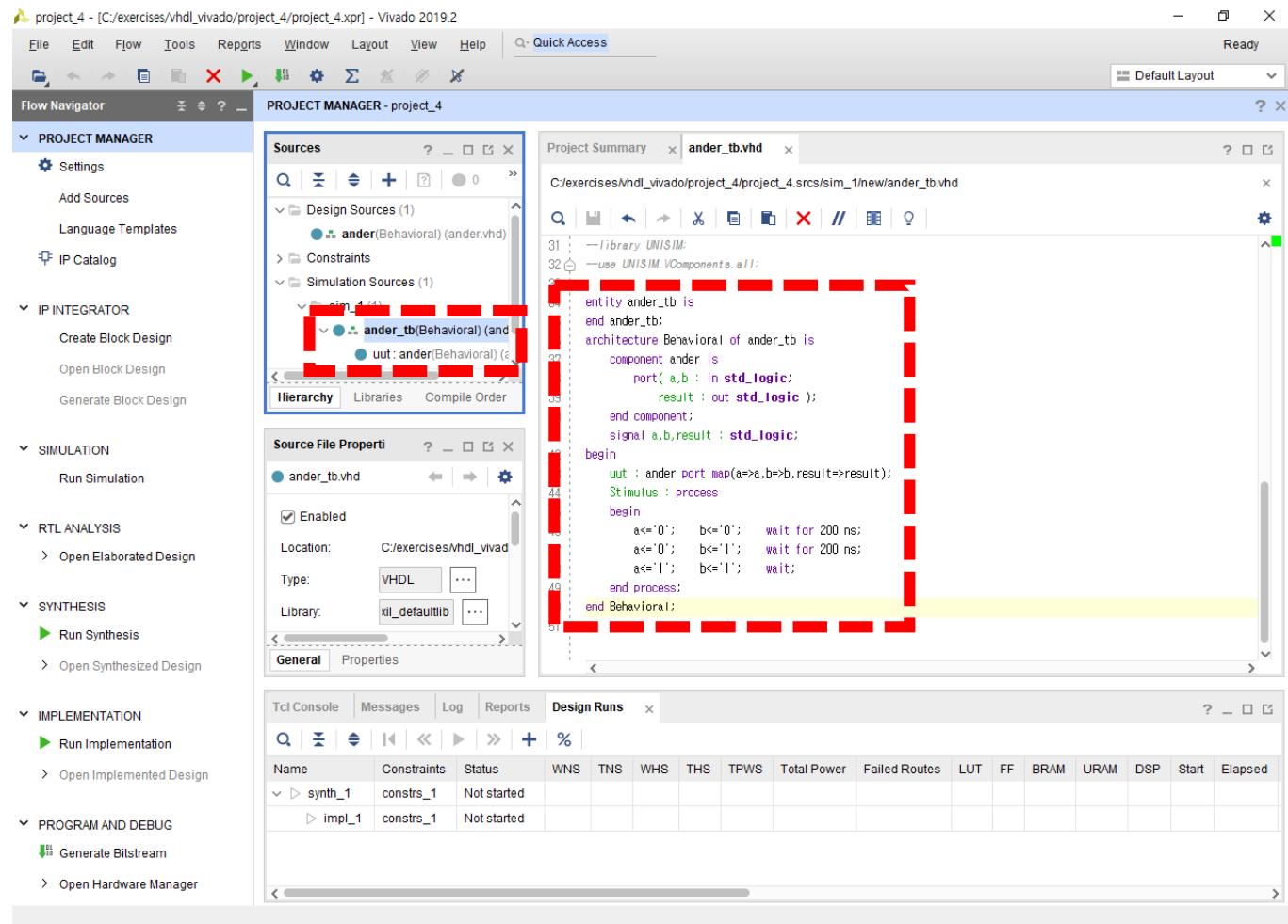
- ❖ Create File버튼을 클릭하여 Create Source File Window가 나오면 File name에 ander\_tb를 입력한 후 OK버튼을 클릭 한다.
- ❖ ander\_tb.vhd파일이 추가된 것을 확인 후 Finish 버튼을 클릭 한다.

# Step 2 Add Simulation Source in Vivado Project 3



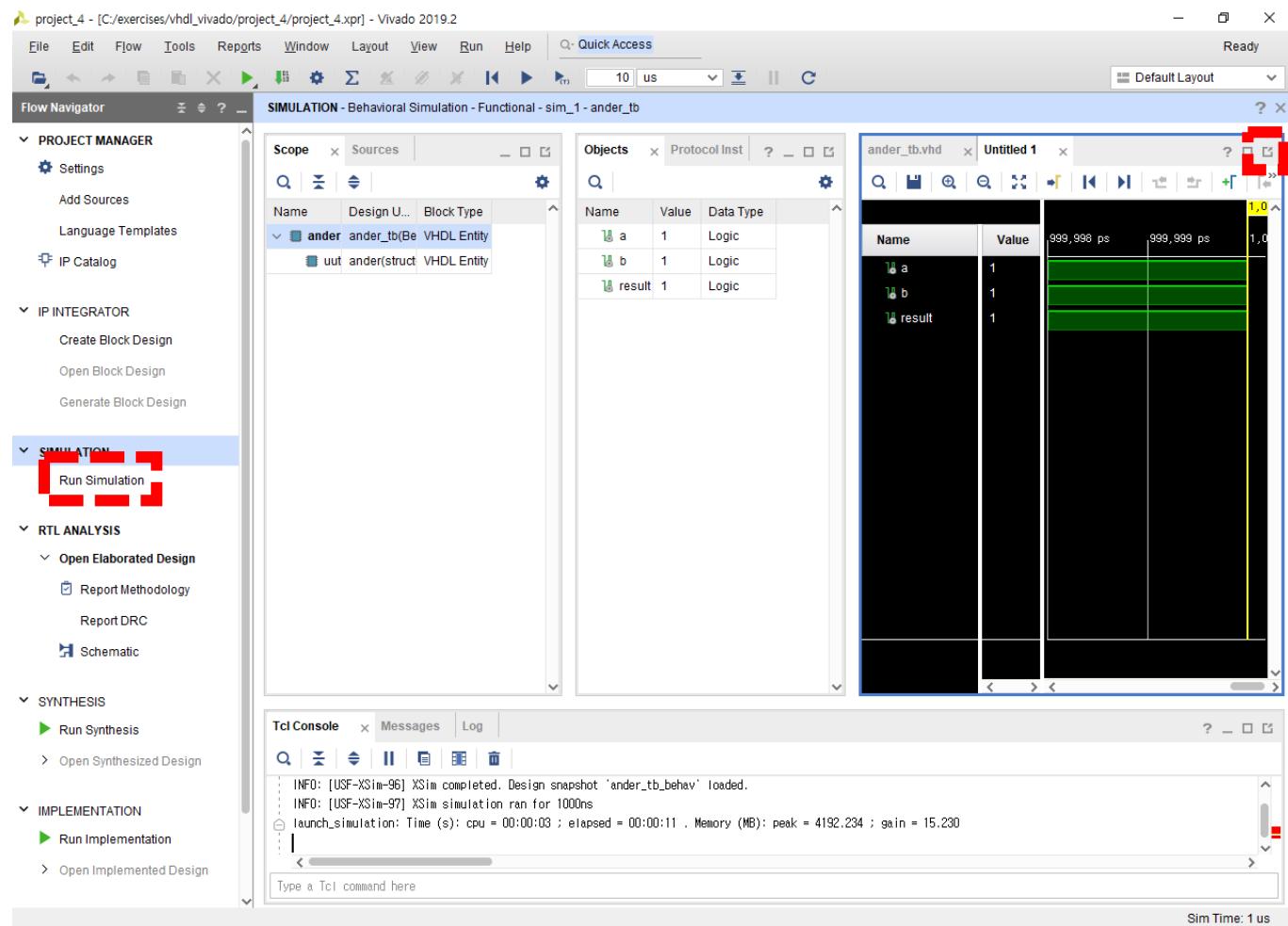
❖ Sources Windows 안에  
Simulation Sources 아래에  
ander\_tb가 추가된 것을 확인  
할 수 있다.

# Step 3 Write Testbench Code



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

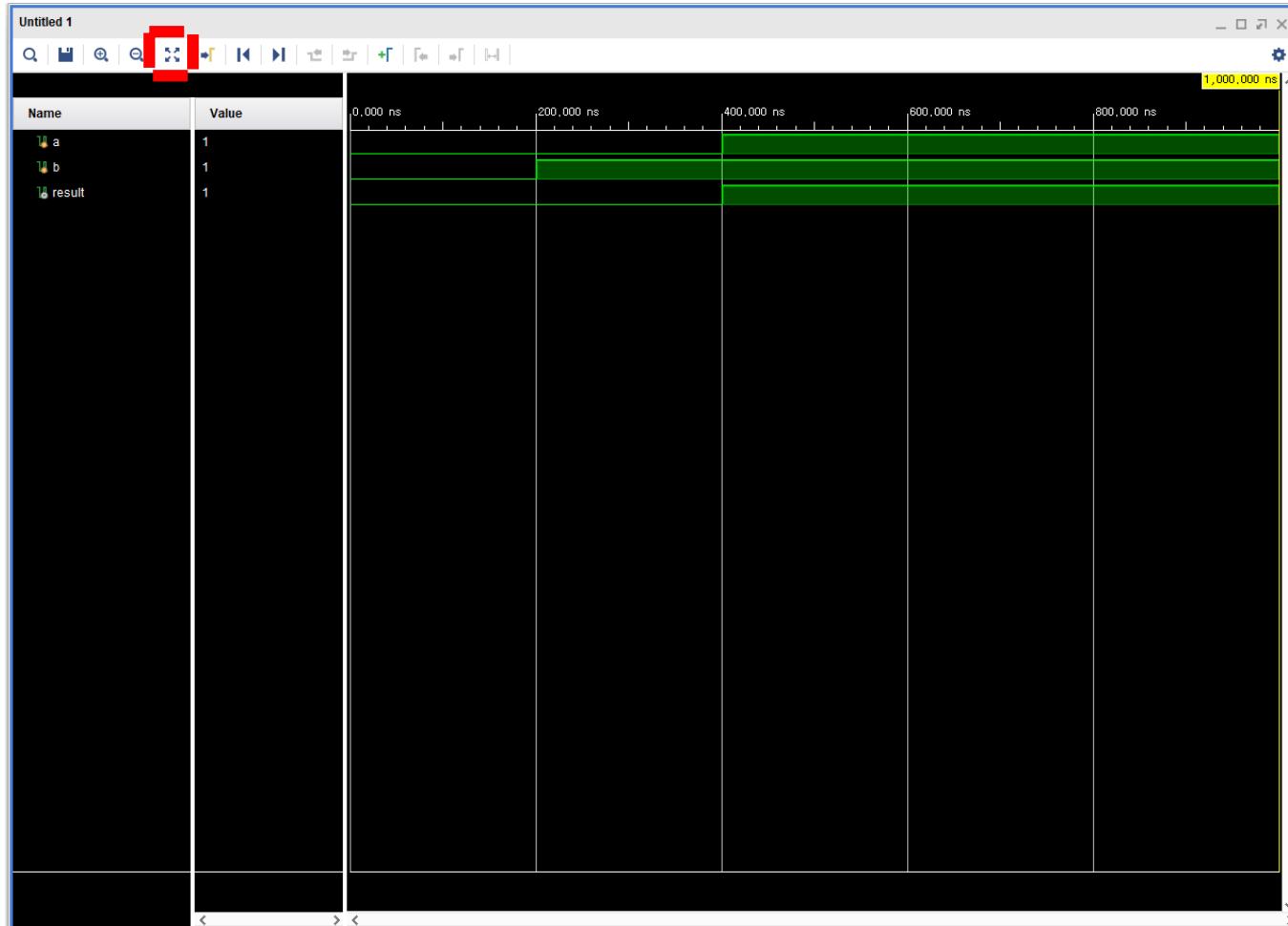
# Step 4 Run Simulation 1



❖ Flow Navigator ⇒ Simulation  
⇒ Run Simulation을 클릭한  
후 Run Behavioral Simulation  
을 클릭한다.

❖ Simulation 결과를 확인하기  
쉽도록 우측에 있는  
Simulation 결과 창의 우측 상  
단에 있는 Float 버튼을 클릭한  
다.

# 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.
- ❖ 우측 예제 코드는 Clock의 Rising Edge순 간에 입력 신호 d가 출력 신호 q에 인가되도록 설계하였다.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity flipflop is
port(
    clk,d : in std_logic;
    q : out std_logic
);
end flipflop;
architecture Behavioral of flipflop is
begin
process(clk)
begin
    if rising_edge(clk) then
        q <= d;
    end if;
end process;
end Behavioral;
```

# Register

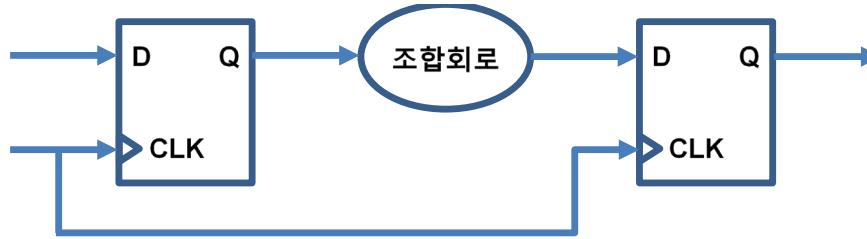
- ❖ Flip-Flop을 데이터 크기에 따라서 16개 또는 32개의 Flip-Flop이 하나의 Clock 신호에 동기를 맞추어서 동작하는 디지털 회로를 Register라고 한다.
- ❖ 우측 예제 코드를 보면 Clock의 Edge순간에 입력 신호 d가 출력 신호 q에 인가되도록 설계하였다. (※ 코드 상에서 Flip-Flop과 다른 점은 입출력신호의 크기만 다르고 다른 부분은 동일하다.)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity reg16_ex1 is
    port(
        clk : in std_logic;
        d : in std_logic_vector(15 downto 0);
        q : out std_logic_vector(15 downto 0)
    );
end reg16_ex1;

architecture Behavioral of reg16_ex1 is
begin
    process(clk)
    begin
        if rising_edge(clk) then
            q <= d;
        end if;
    end process;
end Behavioral;
```

# 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 clr이 추가되어 clr신호가 변하면 프로세스를 실행하여 출력 값을 업데이트하고 clr신호가 '1'(High)이 되면 출력 신호 q는 16'h0000으로 초기화 된다.
- ❖ if clr = '1' then 문장이 if rising\_edge(clk) then 문장보다 먼저 오게 되면 Asynchronous Circuit이 되고 클럭의 Rising Edge를 먼저 확인하고 clr신호가 1(High)인지를 확인하도록 표현하면 Synchronous 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.)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity reg16_ex2 is
port(
    clk, clr : in std_logic;
    d : in std_logic_vector(15 downto 0);
    q : out std_logic_vector(15 downto 0)
);
end reg16_ex2;

architecture Behavioral of reg16_ex2 is
begin
process(clk,clr)
begin
    if clr = '1' then
        q <= x"0000";
    elsif rising_edge(clk) then
        q <= d;
    end if;
end process;
end Behavioral;
```

# Register 3

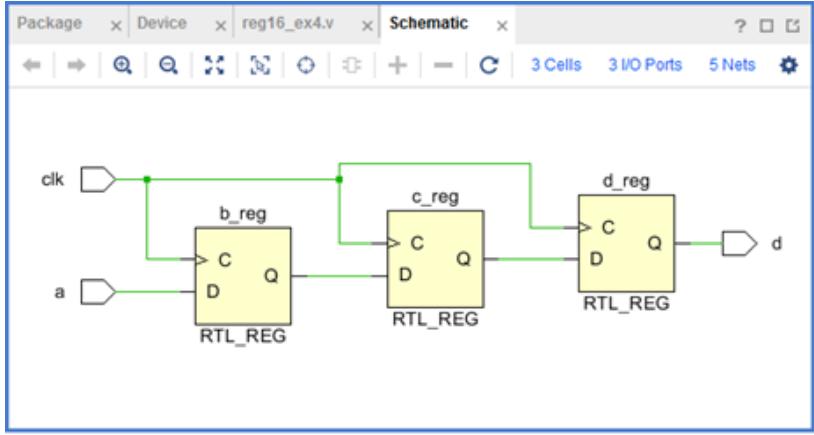
- ❖ Clock Enable 신호를 추가한 Register이다.
- ❖ 이 코드 예제에서 Clear 신호(clr)와 Clock Enable 신호(clken)들은 Synchronous하게 동작하도록 설계되어 있다.
- ❖ Clock Enable 신호(clken)가 '0'(Low)이면 입력 신호(d)가 출력 신호(q)에 인가되지 않고 '1'(High)일 때만 인가되도록 하여 Clock Enable을 설계하였다.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity reg16_ex3 is
    port( clk, clr, clken : in std_logic;
          d : in std_logic_vector(15 downto 0);
          q : out std_logic_vector(15 downto 0));
end reg16_ex3;

architecture Behavioral of reg16_ex3 is
begin
    process(clk,clr,clken)
    begin
        if rising_edge(clk) then
            if clr = '1' then
                q <= x"0000";
            elsif clken = '1' then
                q <= d;
            end if;
        end if;
    end process;
end Behavioral;
```

# Register 4



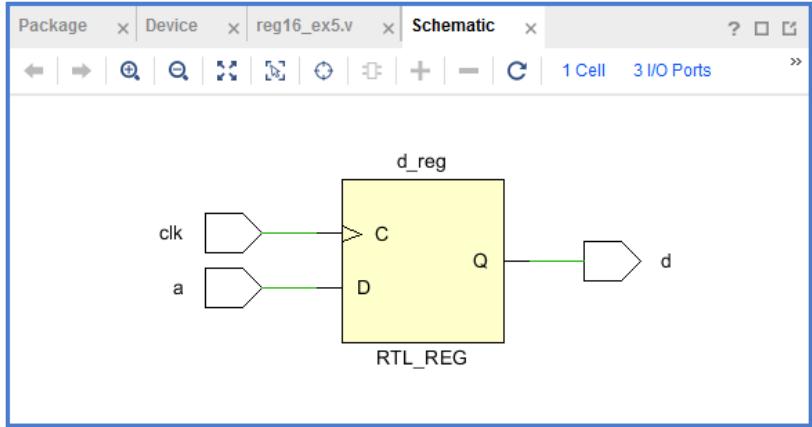
- ❖ Signal을 사용하여 b와 c신호를 선언한 후 입력포트 a를 b에 인가하고 b를 c에 인가하고 c를 출력포트 d로 인가하도록 표현한 예이다.
- ❖ Flow Navigator ⇒ RTL Analysis ⇒ Elaborated Design ⇒ Schematic을 클릭하면 위 Schematic 그림을 확인할 수 있다.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity reg16_ex4 is
port(
    clk, a : in std_logic;
    d : out std_logic
);
end reg16_ex4;

architecture Behavioral of reg16_ex4 is
signal b,c : std_logic;
begin
process(clk)
begin
    if rising_edge(clk) then
        b <= a;
        c <= b;
        d <= c;
    end if;
end process;
end Behavioral;
```

# Register 5



❖ Variable을 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.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity reg16_ex5 is
port(
    clk, a : in std_logic;
    d : out std_logic
);
end reg16_ex5;

architecture Behavioral of reg16_ex5 is
begin
    process(clk)
        variable b,c : std_logic;
    begin
        if rising_edge(clk) then
            b <= a;
            c <= b;
            d <= c;
        end if;
    end process;
end Behavioral;
```

# 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.
- ❖ 우측 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.
- ❖ 이 카운터에서 + 연산자를 사용하여 값을 증가시키기 때문에 우측 코드 예제의 Line 3과 같이 + 연산자가 정의되어 있는 STD\_LOGIC\_UNSIGNED 또는 STD\_LOGIC\_SIGNED을 선언해야 한다.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity counter_ex1 is
    port(clk, clr : in std_logic;
        q : out std_logic_vector(15 downto 0));
end counter_ex1;

architecture Behavioral of counter_ex1 is
    signal cnt : std_logic_vector(15 downto 0);
begin
    process(clk,clr)
    begin
        if clr = '1' then
            cnt <= x"0000";
        elsif rising_edge(clk) then
            cnt <= cnt + 1;
        end if;
    end process;
    q <= cnt;
end Behavioral;
```

# Counter 2

- ❖ 우측 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.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity counter_ex2 is
    port(clk, clr : in std_logic;
        q : out std_logic_vector(15 downto 0));
end counter_ex2;
architecture Behavioral of counter_ex2 is
    signal cnt : std_logic_vector(15 downto 0);
begin
    process(clk,clr)
    begin
        if clr = '1' then
            cnt <= x"0000";
        elsif rising_edge(clk) then
            cnt <= cnt + 1;
            if cnt = 999 then
                cnt <= x"0000";
                -- 특정시간마다 해야 할 일
            end if;
        end if;
    end process;
    q <= cnt;
end Behavioral;
```

# Counter 3

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

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity counter_ex3 is
    port(clk, clr, cnten : in std_logic;
        q : out std_logic_vector(15 downto 0));
end counter_ex3;
architecture Behavioral of counter_ex3 is
    signal cnt : std_logic_vector(15 downto 0);
begin
    process(clk,clr,cnten)
    begin
        if clr = '1' then
            cnt <= x"0000";
        elsif rising_edge(clk) then
            if cnten = '1' then
                cnt <= cnt + 1;
                if cnt = 999 then
                    cnt <= x"0000";
                    -- 특정시간마다 해야 할 일
                end if;
            end if;
        end if;
    end process;
    q <= cnt;
end Behavioral;
```

# Counter 4

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

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity counter_ex4 is
    port(clk, clr, cnten, updown : in std_logic;
          q : out std_logic_vector(15 downto 0));
end counter_ex4;
architecture Behavioral of counter_ex4 is
    signal cnt : std_logic_vector(15 downto 0);
begin
    process(clk,clr,cnten,updown)
    begin
        if clr = '1' then
            cnt <= x"0000";
        elsif rising_edge(clk) then
            if cnten = '1' then
                if updown = '1' then
                    cnt <= cnt + 1;
                else
                    cnt <= cnt - 1;
                end if;
            end if;
        end if;
    end process;
    q <= cnt;
end Behavioral;
```

# 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 VHDL로 Counter 를 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.

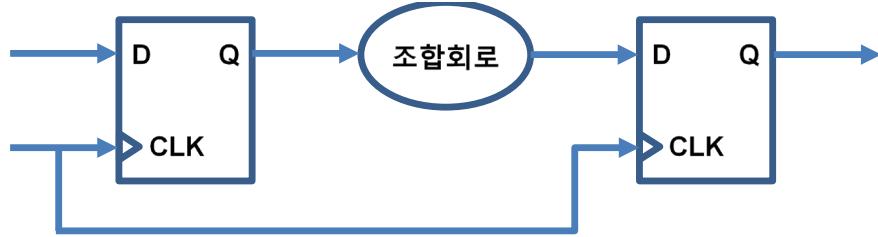
# Synthesis & Implementation

- ❖ 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.)
- ❖  $10\text{ns}$ 에  $0.3\text{m}(30\text{cm})$ 를 움직이는 속도로 계산된다. (※ 회로상에 전기신호가 안정된 후  $30\text{cm}$  도선 거리에 전달되려면  $10 \text{ ns}$ 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.

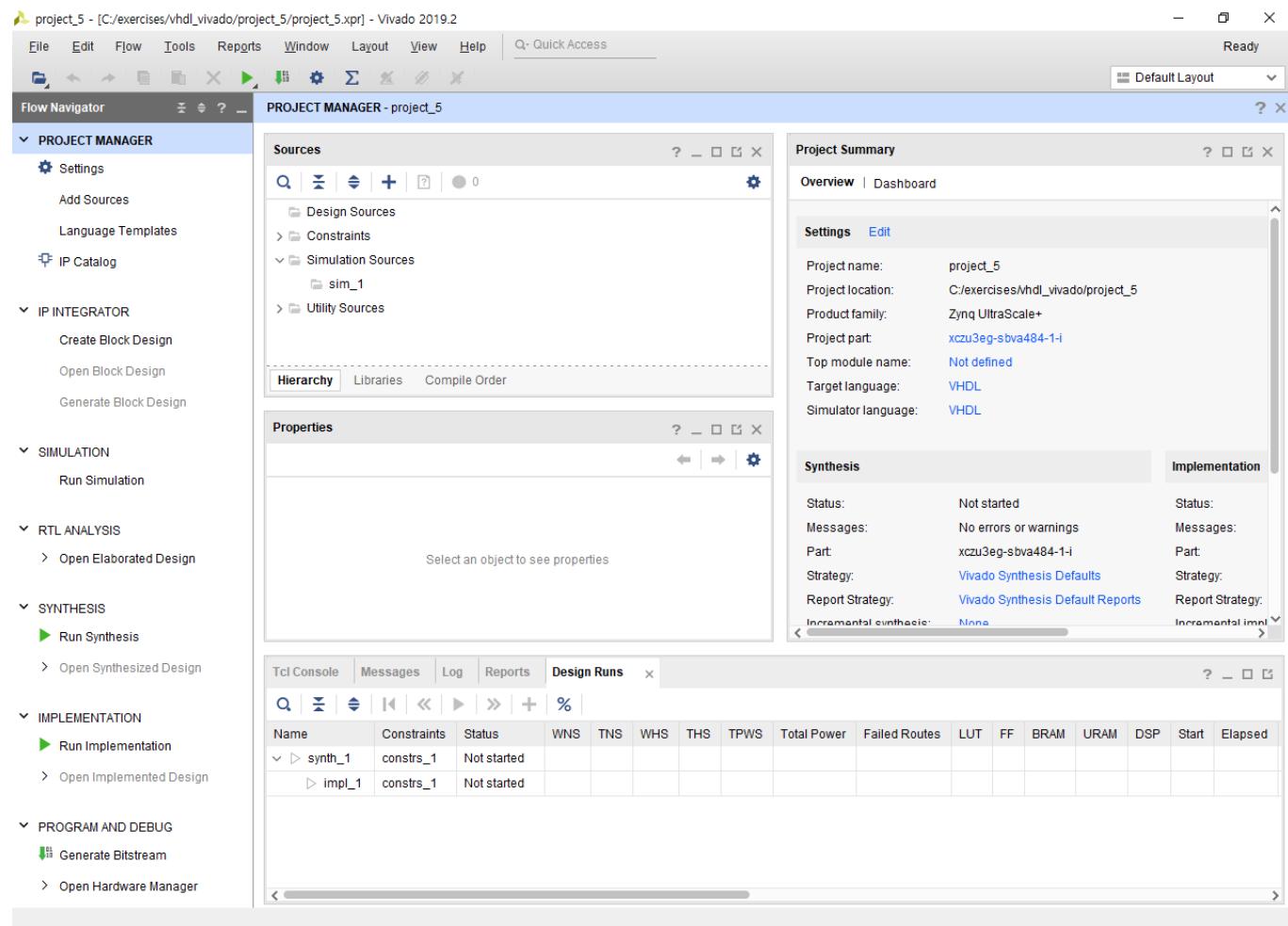
# Timing Constraints

- ❖ 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

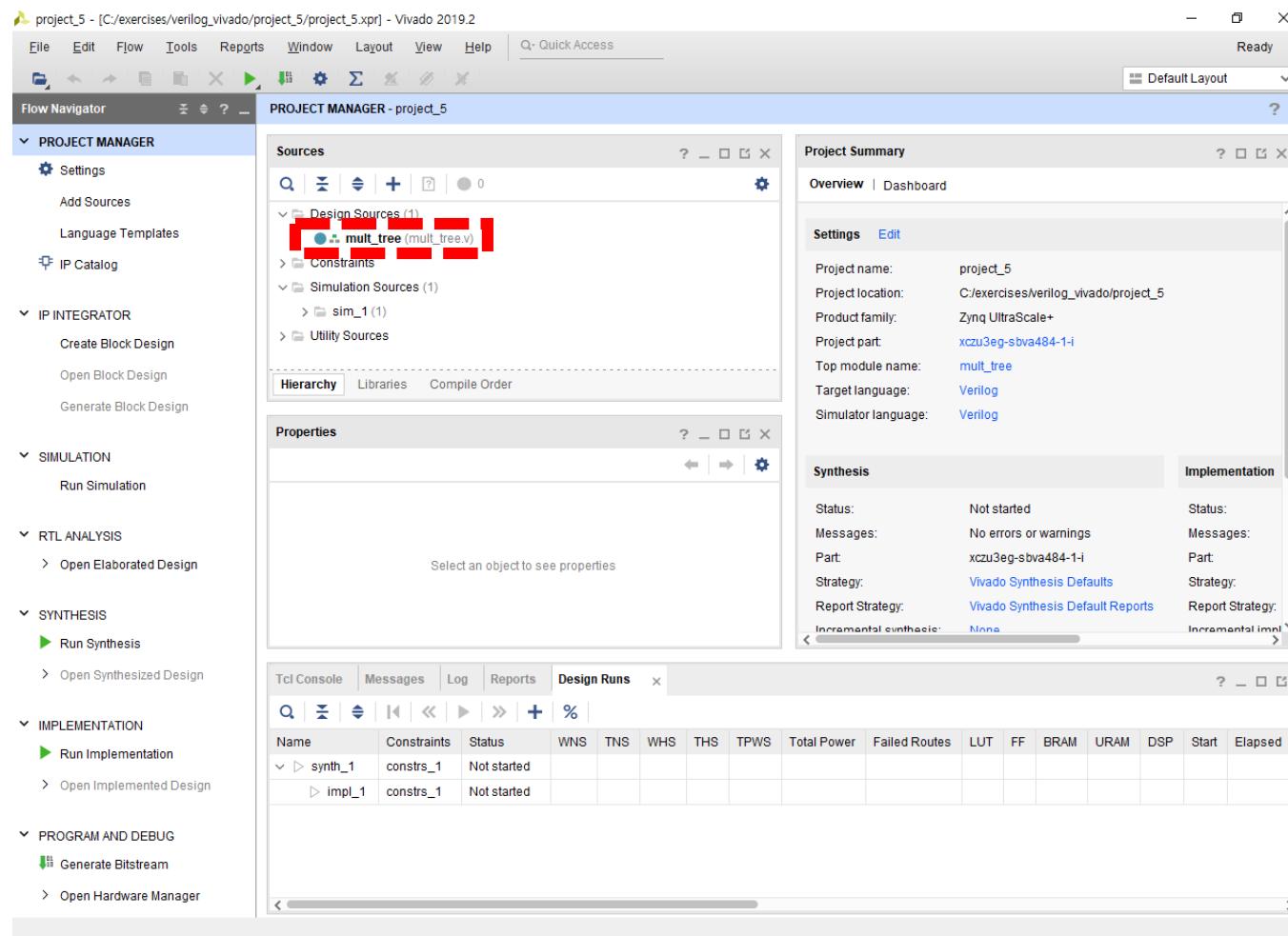
- Timing Constraints
- Static Timing Analysis
- How to Debug Hardware Directly
- Debugging using ILA
- Ultra96 Training Kit Exercises 5

# 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.vhd 파일을 프로젝트에 추가한다.

# Step 3 Write Design Source Code

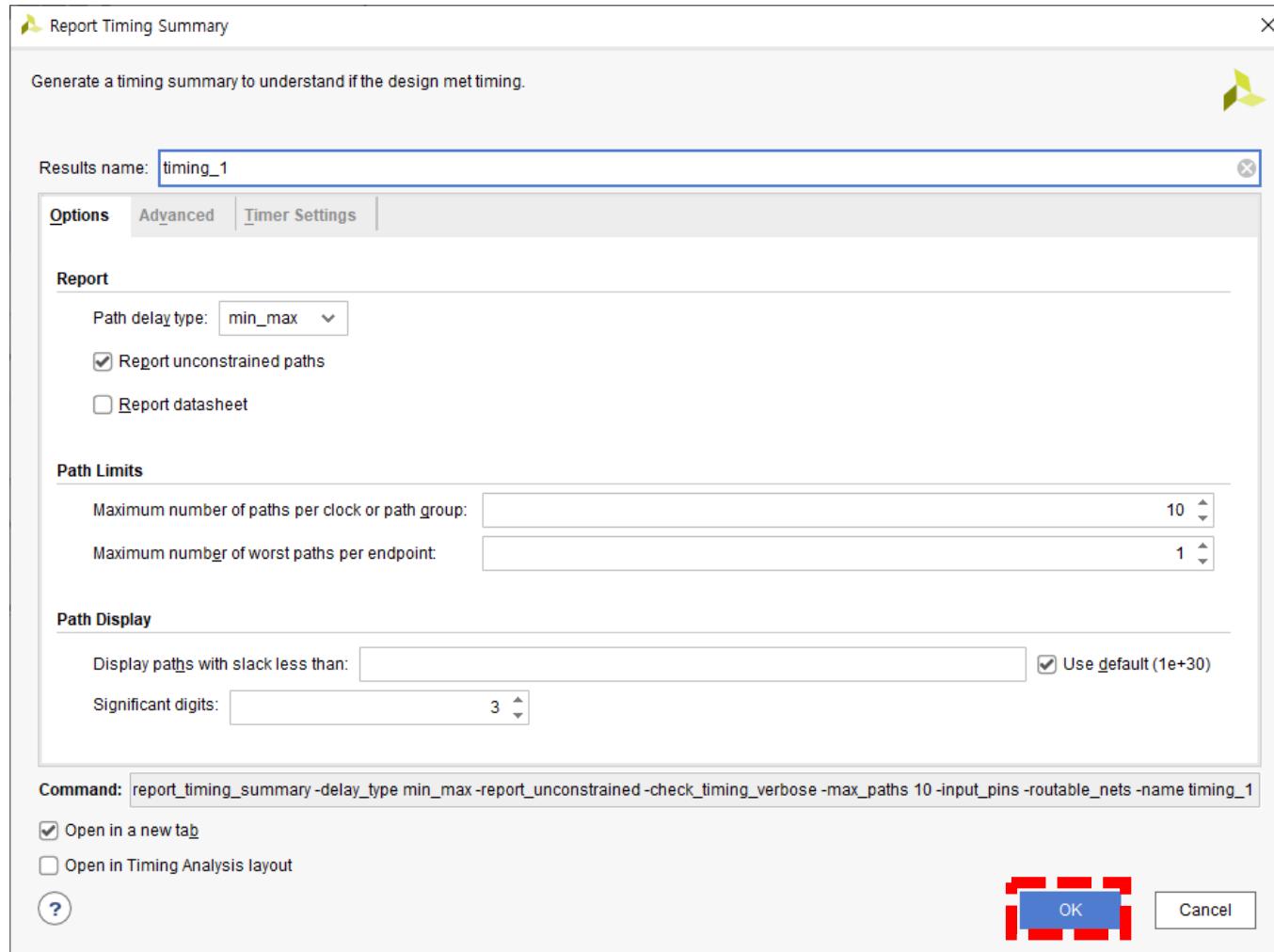
- ❖ Source Windows 안의 mult\_tree.vhd 파일을 더블 클릭하여 Editor Window에 연 후 mult\_tree 모듈 소스 코드를 입력한다.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mult_tree is
    Port(
        clk : in std_logic;
        i_a, i_b, i_c, i_d : in std_logic_vector(3 downto 0);
        o_result : out std_logic_vector(7 downto 0)
    );
end mult_tree;
```

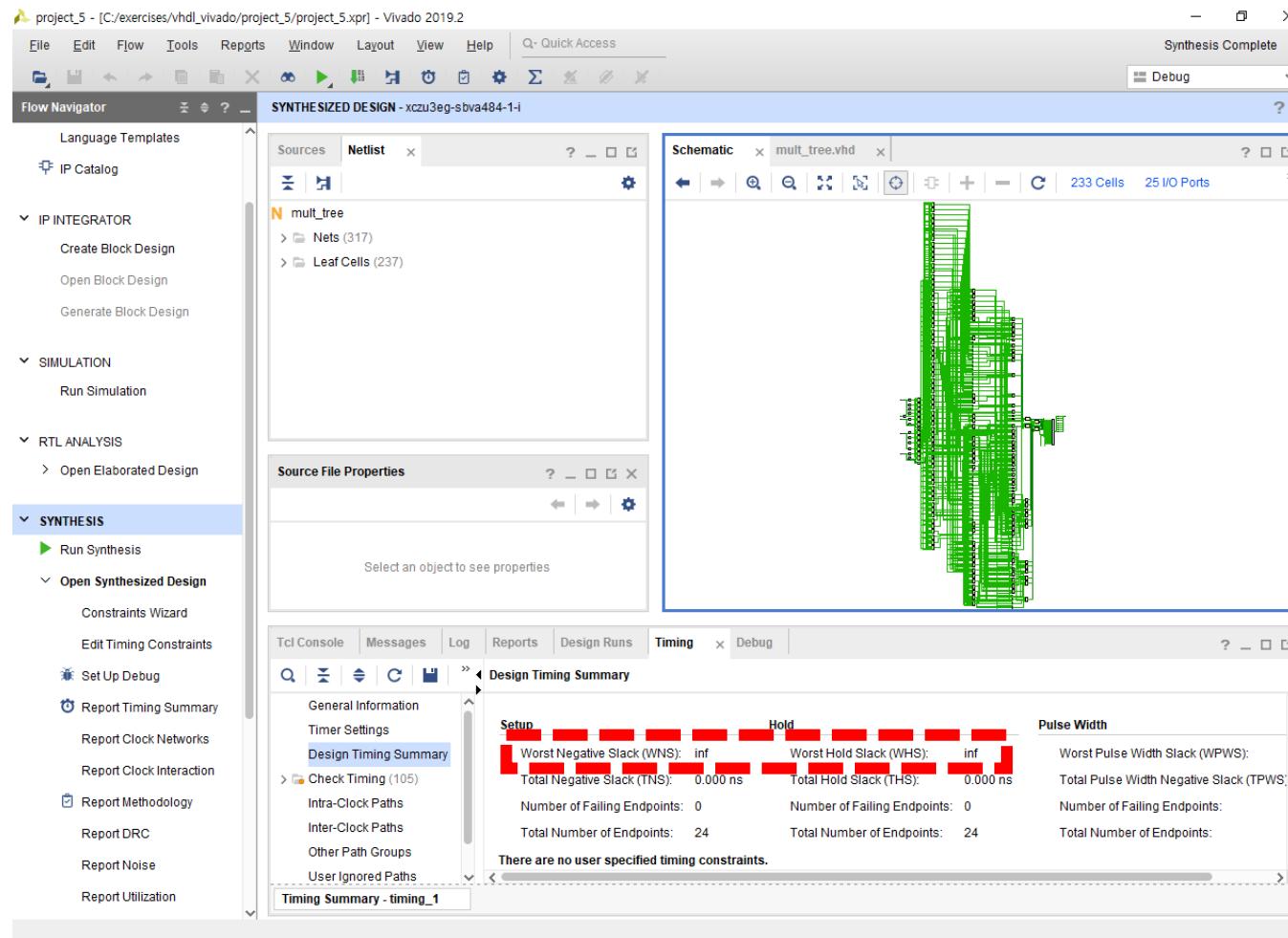
```
architecture Behavioral of mult_tree is
    signal a_buf, b_buf, c_buf, d_buf : std_logic_vector(3 downto 0);
    signal result_buf : std_logic_vector(23 downto 0);
begin
    process(clk)
    begin
        if rising_edge(clk) then
            a_buf <= i_a;
            b_buf <= i_b;
            c_buf <= i_c;
            d_buf <= i_d;
            result_buf <= (
                ((a_buf * b_buf) * b_buf) *
                ((c_buf * d_buf) * d_buf)
            );
        end if;
    end process;
    o_result <= result_buf(23 downto 16);
end Behavioral;
```

# Step 4 Add Timing Constraints 1



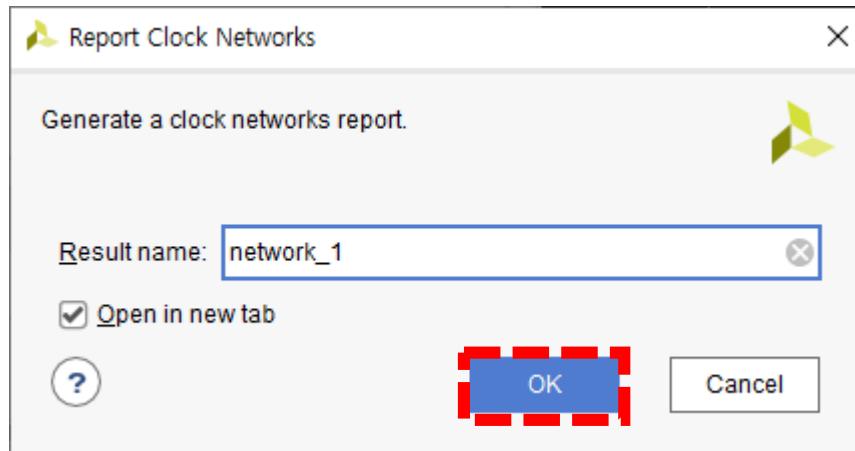
- ❖ Flow Navigator ⇒ Synthesis  
⇒ Run Synthesis를 클릭하여 디자인을 합성한다.
- ❖ Flow Navigator ⇒ Synthesis  
⇒ Open Synthesized Design  
⇒ Report Timing Summary를 클릭하면 Report Timing Summary Window가 열린다.
- ❖ OK버튼을 클릭하여 Timing Summary를 생성한다.

# Step 4 Add Timing Constraints 2



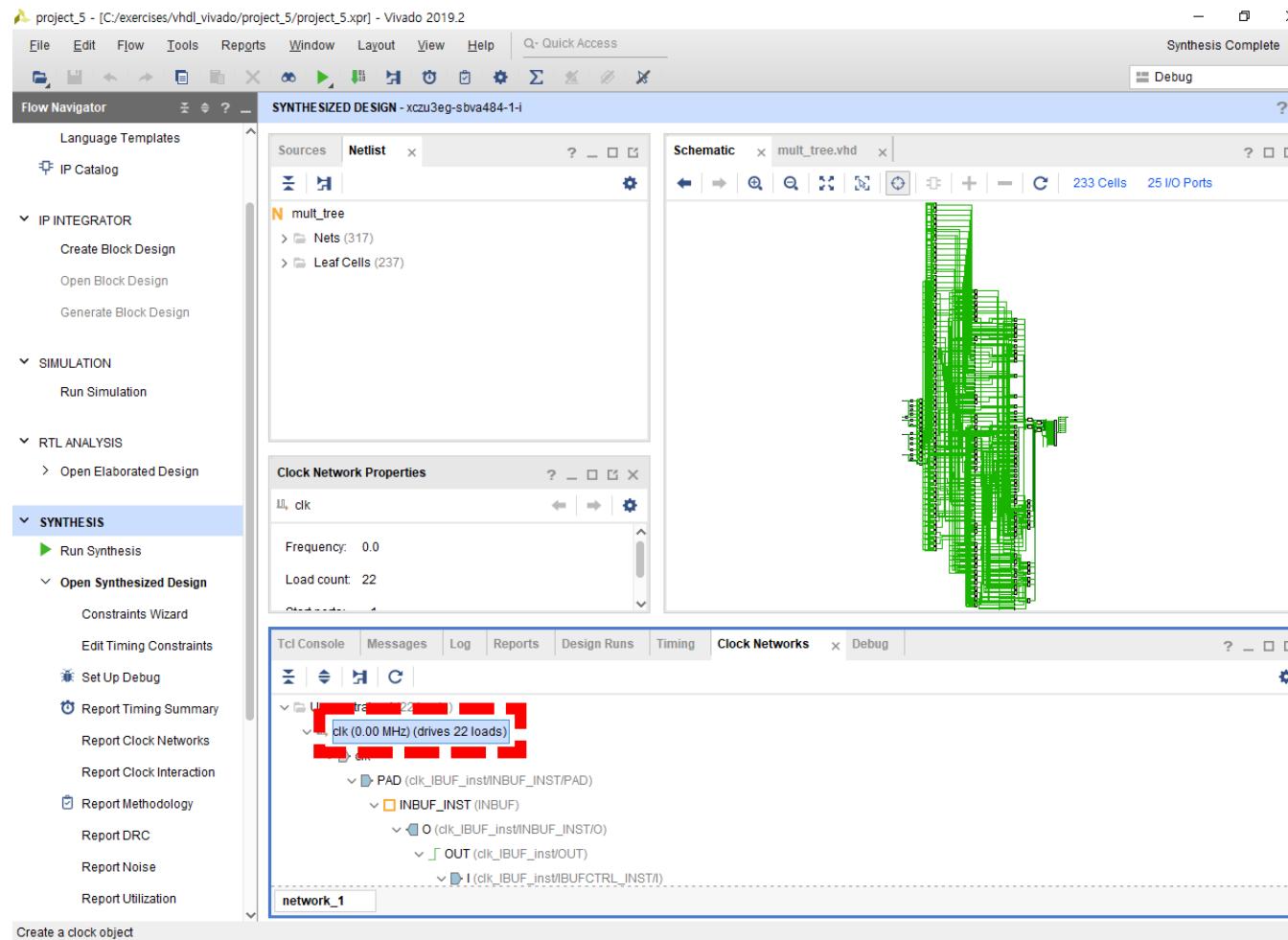
- ❖ Vivado 하단 Window의 Timing 탭에서 Design Timing Summary 메뉴를 선택하면 Setup과 Hold의 Worst Negative Slack이 inf로 뜨는 것을 확인할 수 있다.
- ❖ Inf로 나오는 이유는 Timing Constraints가 적용되지 않았기 때문이다.

## Step 4 Add Timing Constraints 3



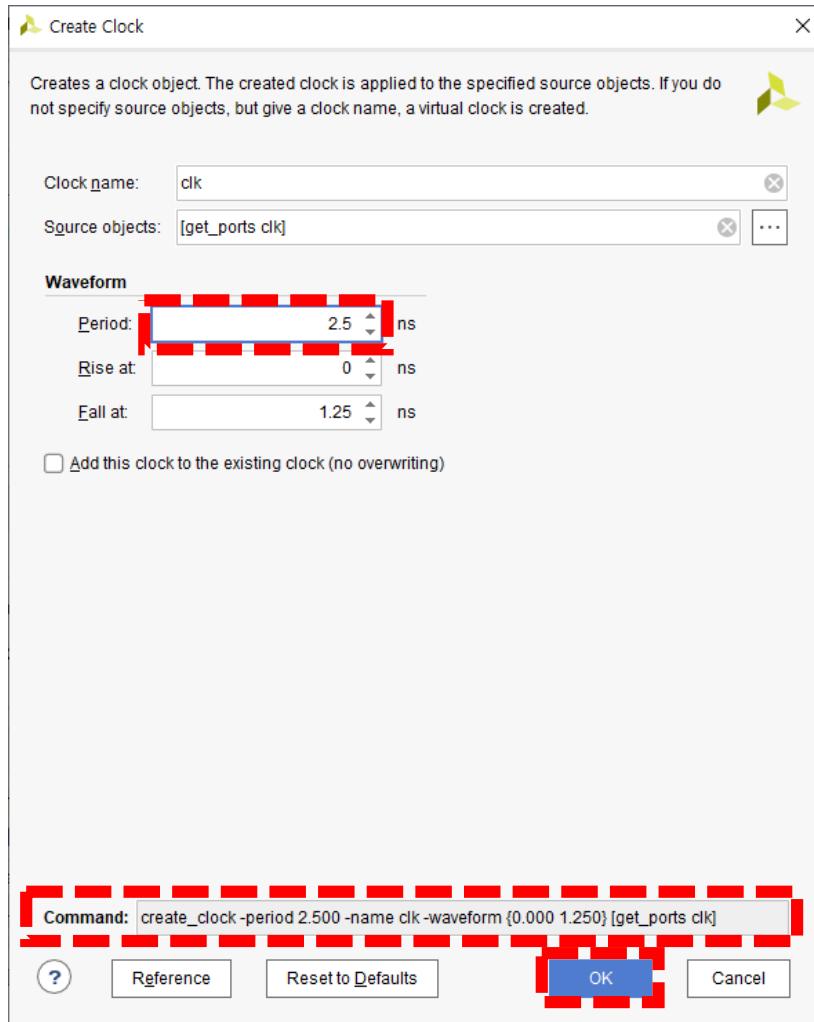
❖ Flow Navigator ⇒ Synthesis  
⇒ Open Synthesized Design  
⇒ Report Clock Network를  
클릭하여 Report Clock  
Network Window가 열리면  
OK버튼을 클릭한다.

# Step 4 Add Timing Constraints 4



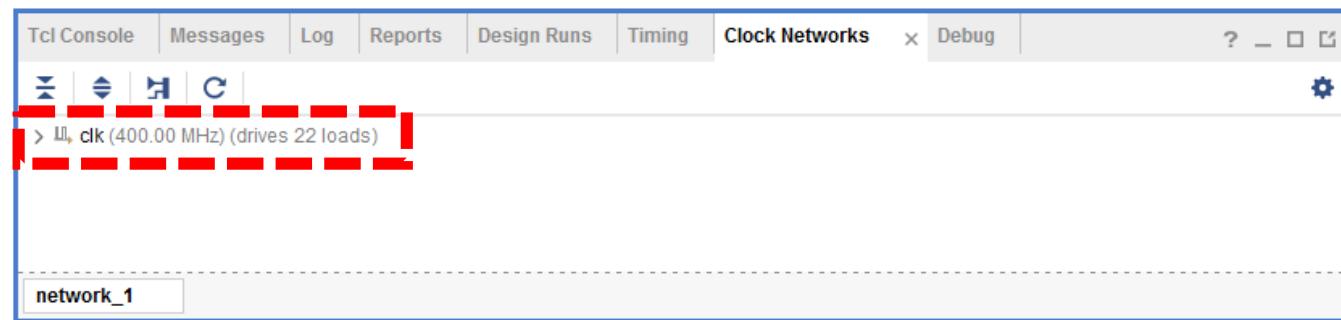
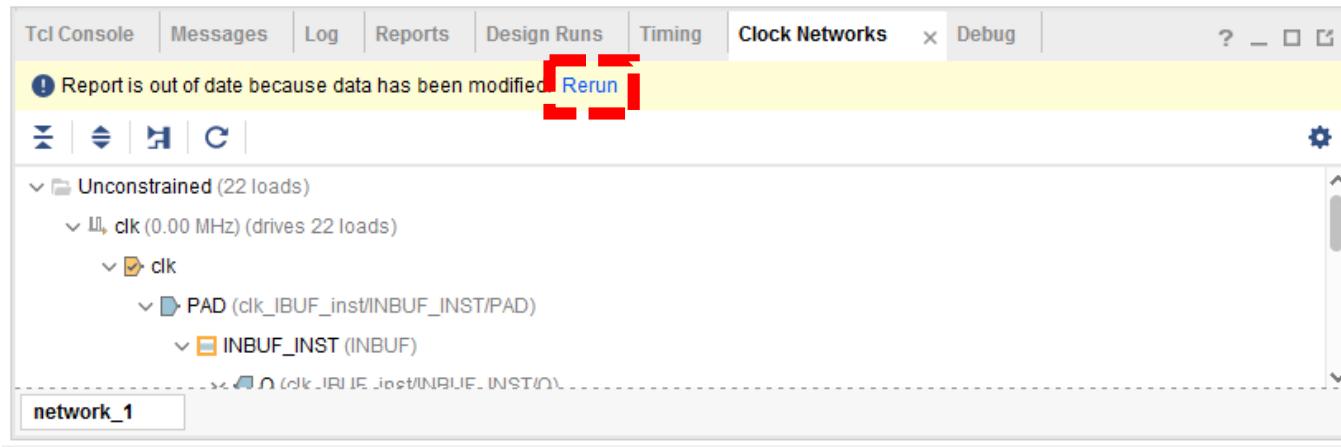
❖ Vivado 하단 Window안에  
Clock Networks탭에서  
Unconstrained 항목 바로 아래  
에 뜨는 clk위에서 마우스 오  
른쪽 버튼을 클릭하고 Create  
Clock메뉴를 클릭한다.

# Step 4 Add Timing Constraints 5



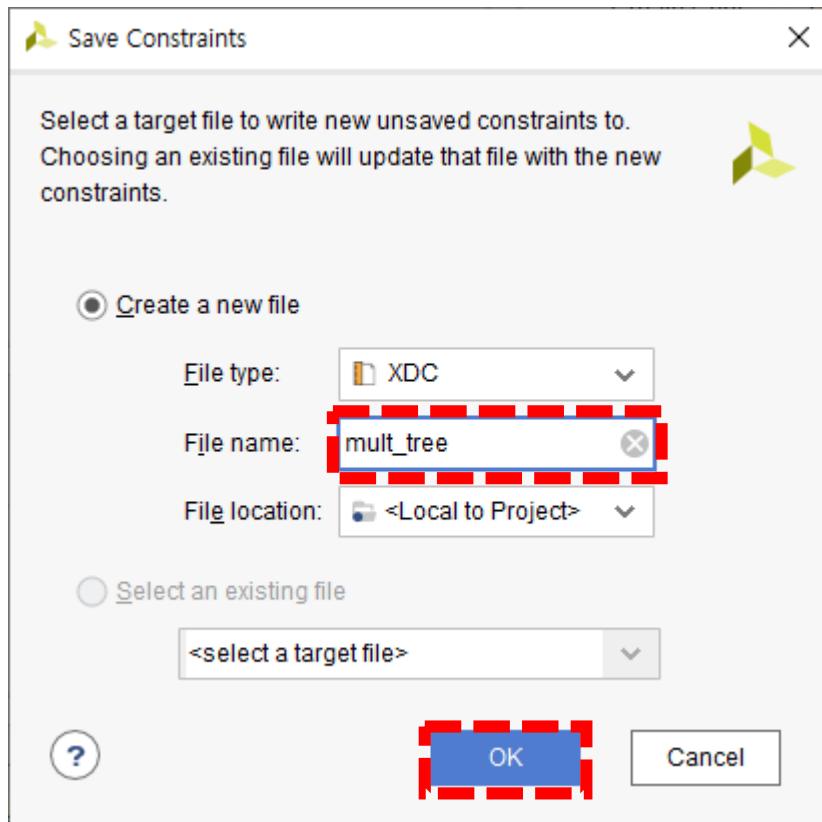
- ❖ Create Clock Windows가 나오면 Period에 2.5를 입력하여 400MHz타겟으로 Clock Constraints를 주고 OK버튼을 클릭한다.
- ❖ Command 폼에 뜨는 내용은 GUI를 통해 설정한 Constraints에 대한 스크립트로 xdc파일에 저장될 코드이다.

# Step 4 Add Timing Constraints 6



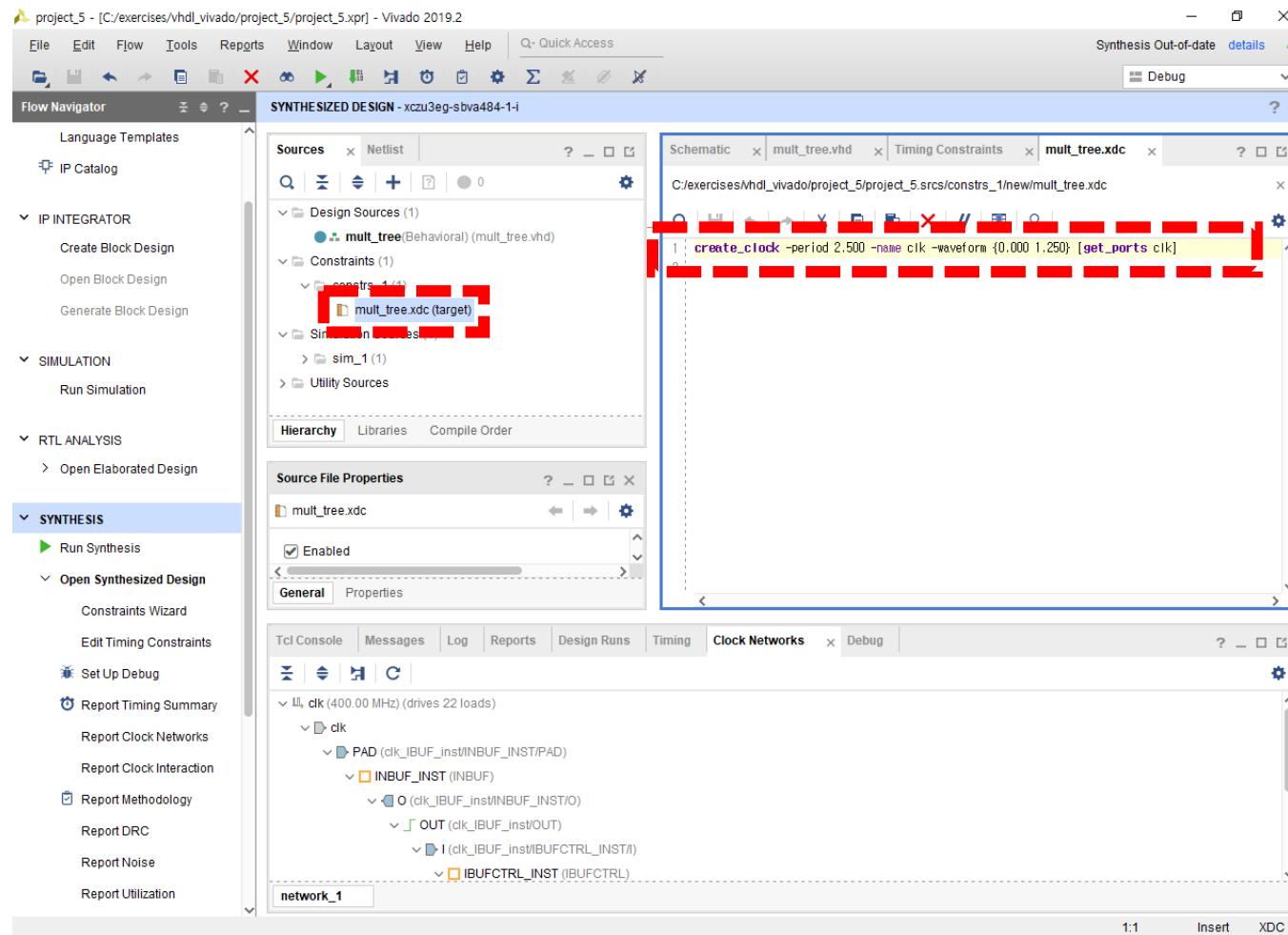
❖ Clock Networks 탭에 생긴 Rerun 메뉴를 클릭하면 clk에 대한 Timing Constraints가 추가되는 것을 확인할 수 있다.

# Step 4 Add Timing Constraints 7



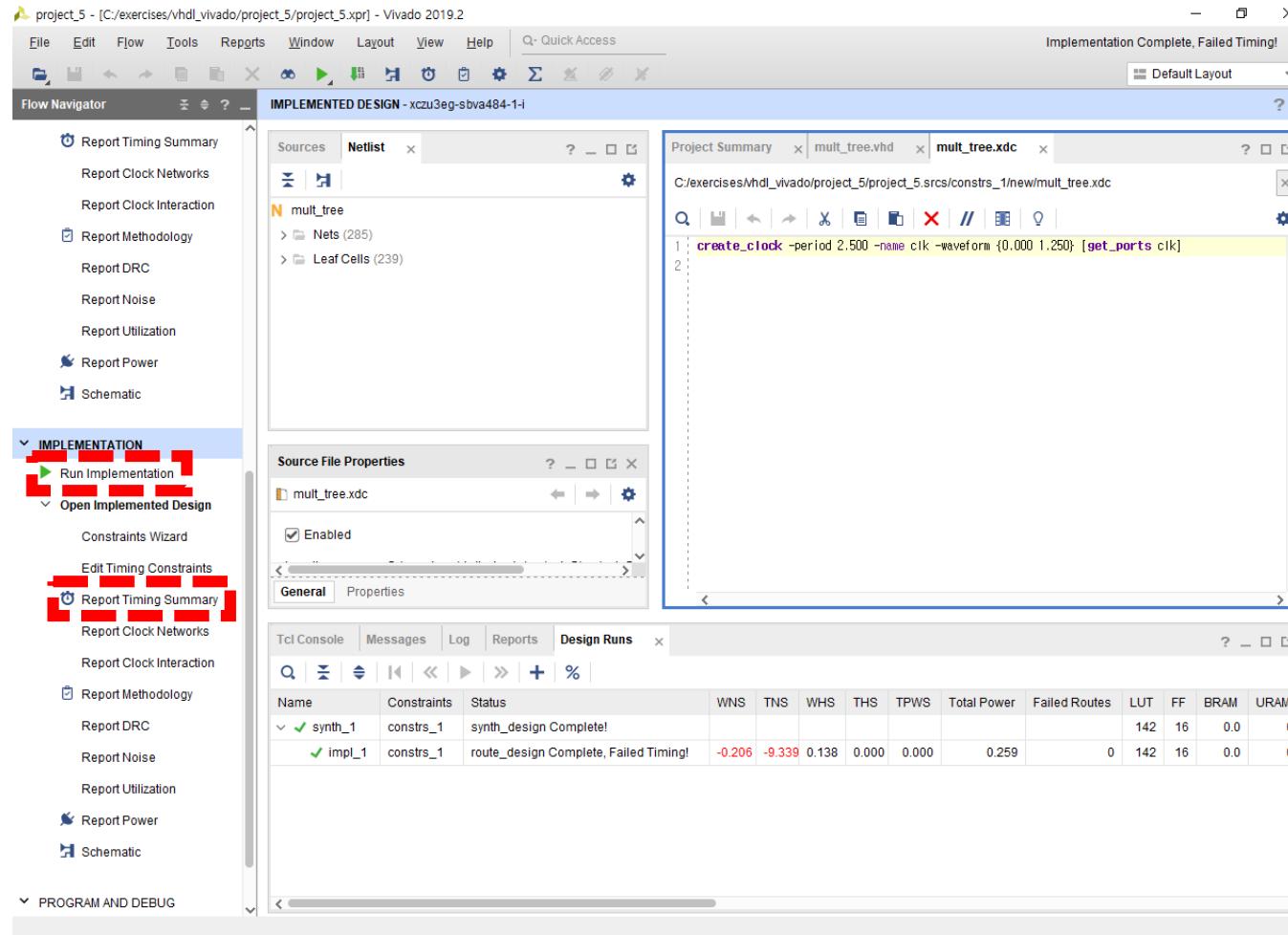
- ❖ File ⇒ Constraints ⇒ Save 메뉴를 클릭한다.
- ❖ Save Constraints Window가 나오면 File name에 mult\_tree를 입력하고 OK버튼을 클릭한다.

# Step 4 Add Timing Constraints 8



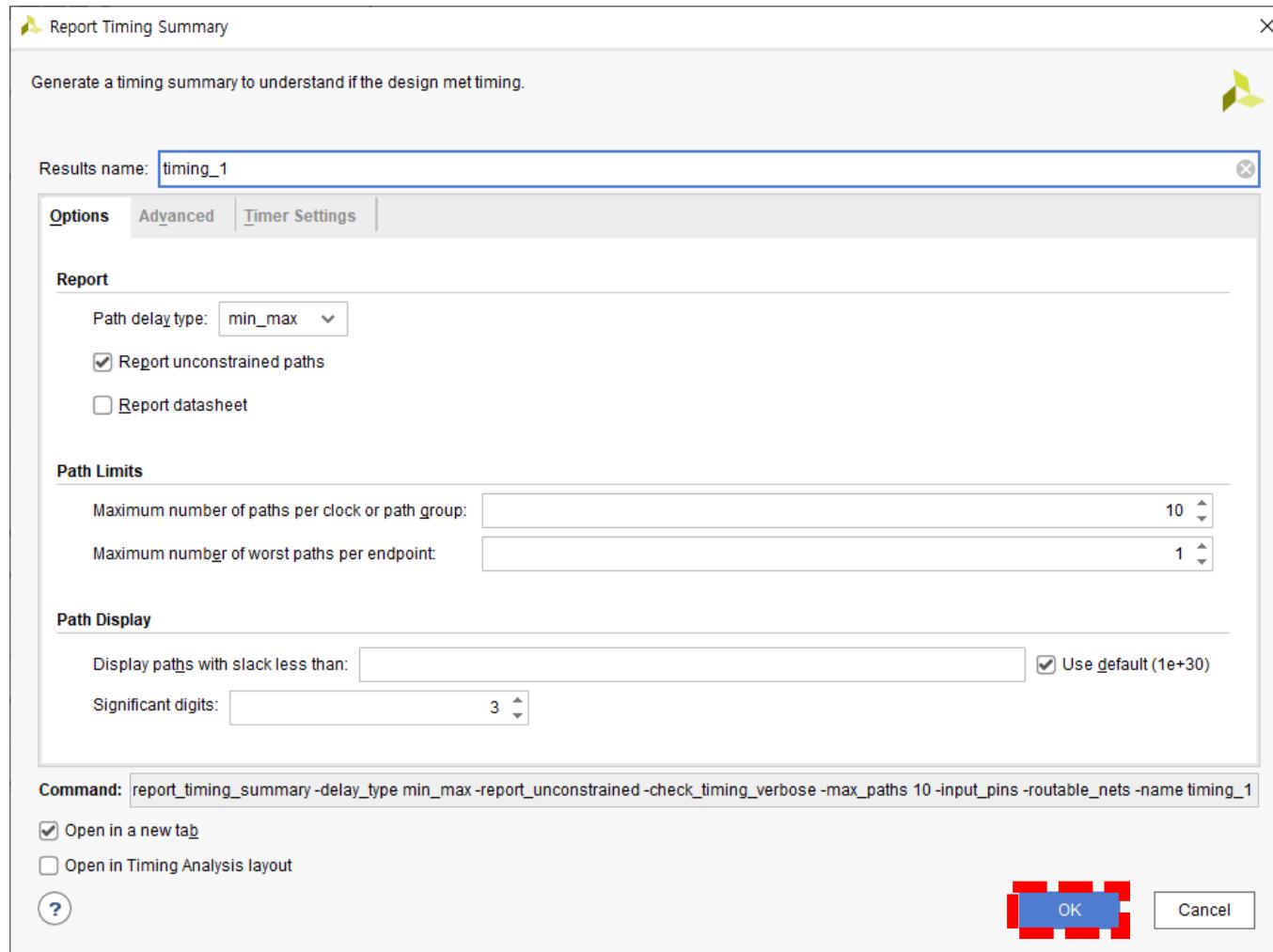
❖ Sources Windows 안에  
Constraints 아래 mult\_tree.xdc  
파일을 더블 클릭하면 Timing  
Constraints 내용이 추가된 것  
을 확인할 수 있다.

# Step 5 Run Implementation



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

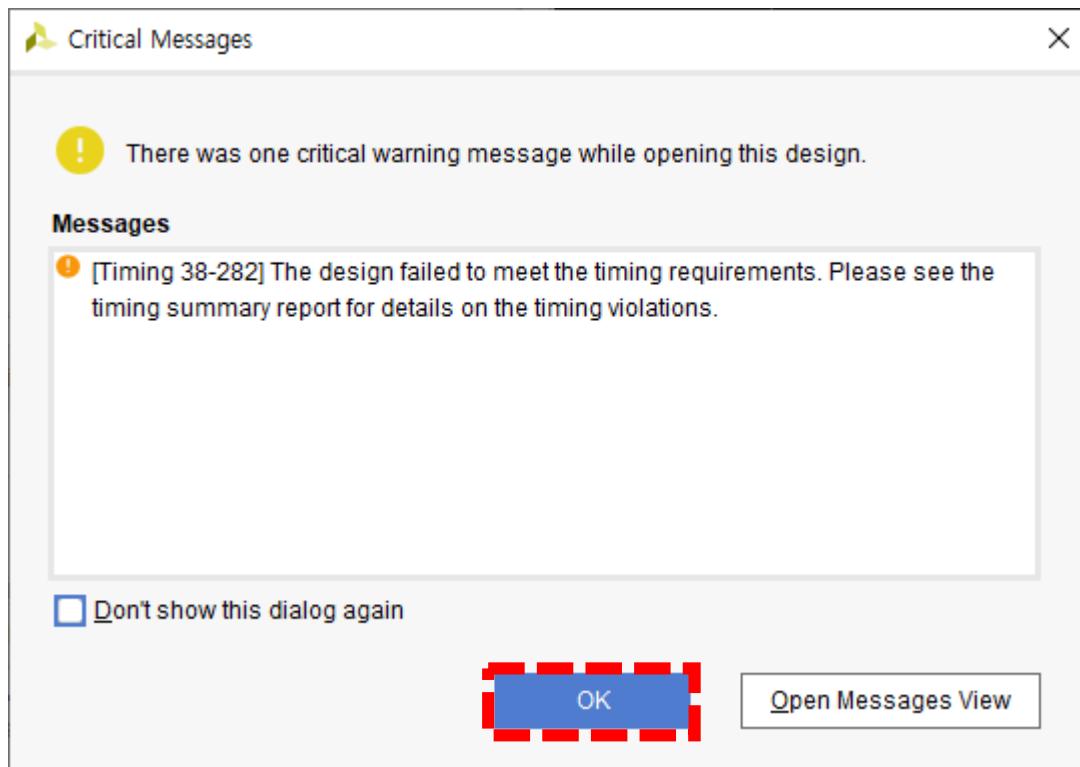
# Step 6 Analysis Timing 1



## ❖ Report Timing Summary

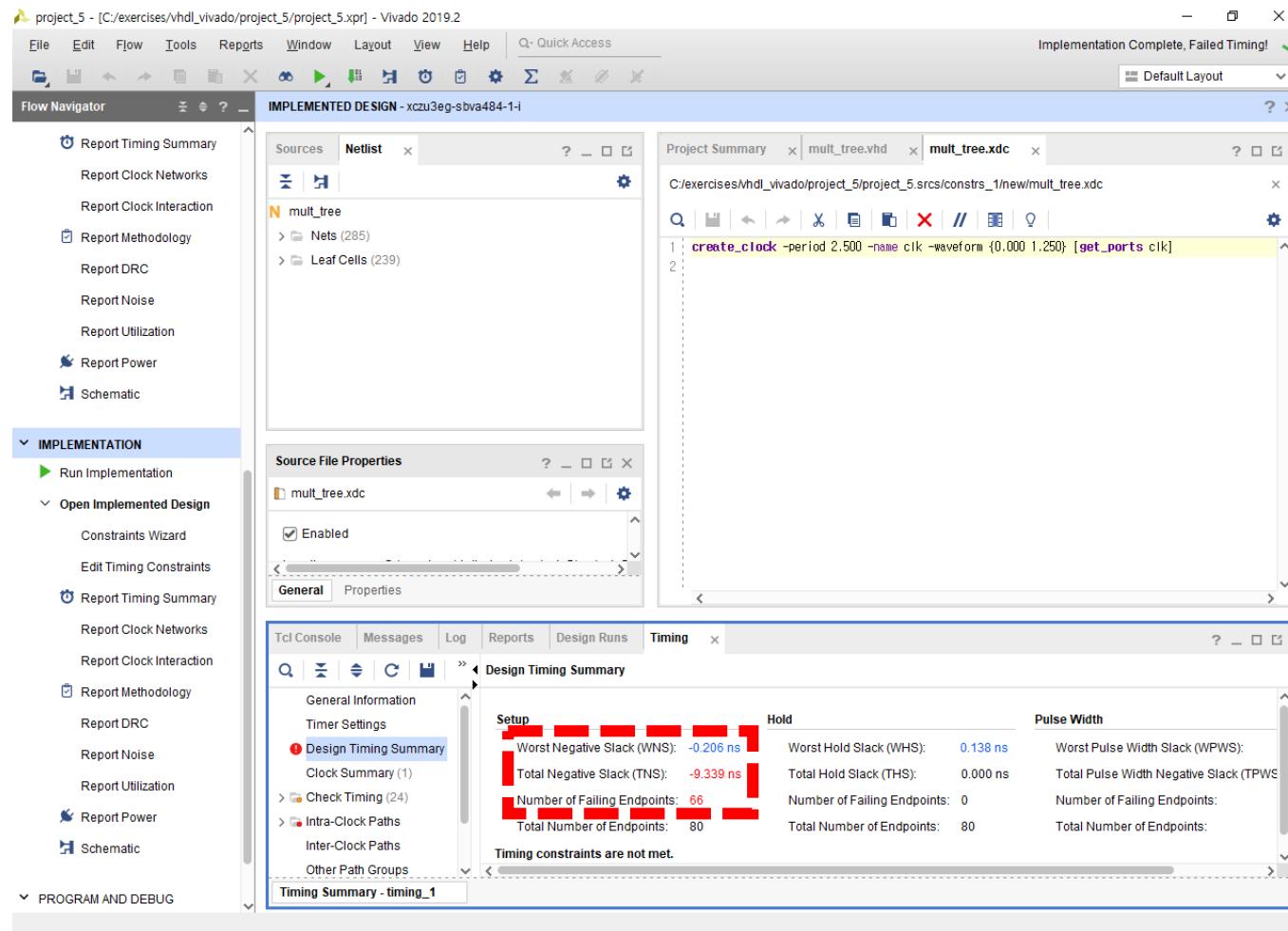
Window가 나오면 OK 버튼을 클릭한다.

# Step 6 Analysis Timing 2



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

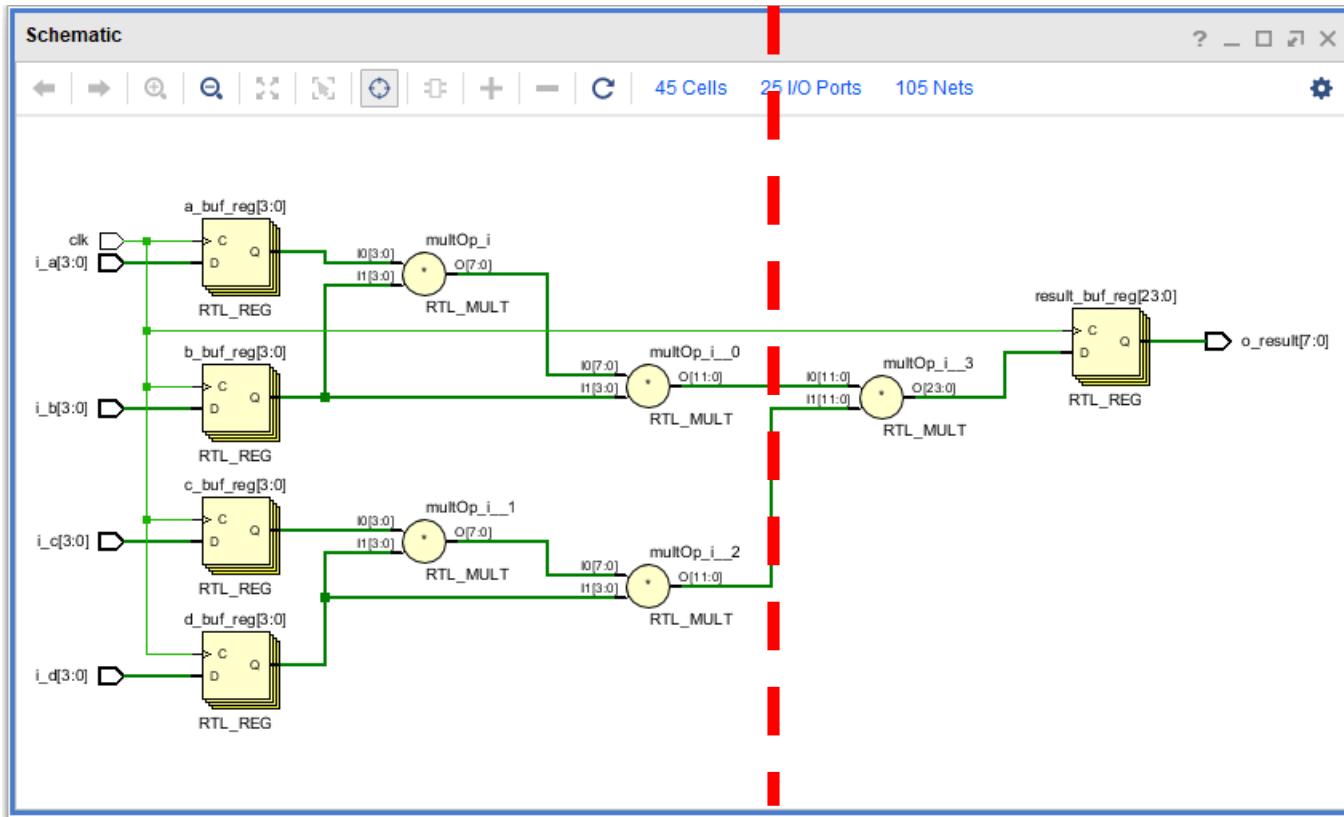
# Step 6 Analysis Timing 3



❖ 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

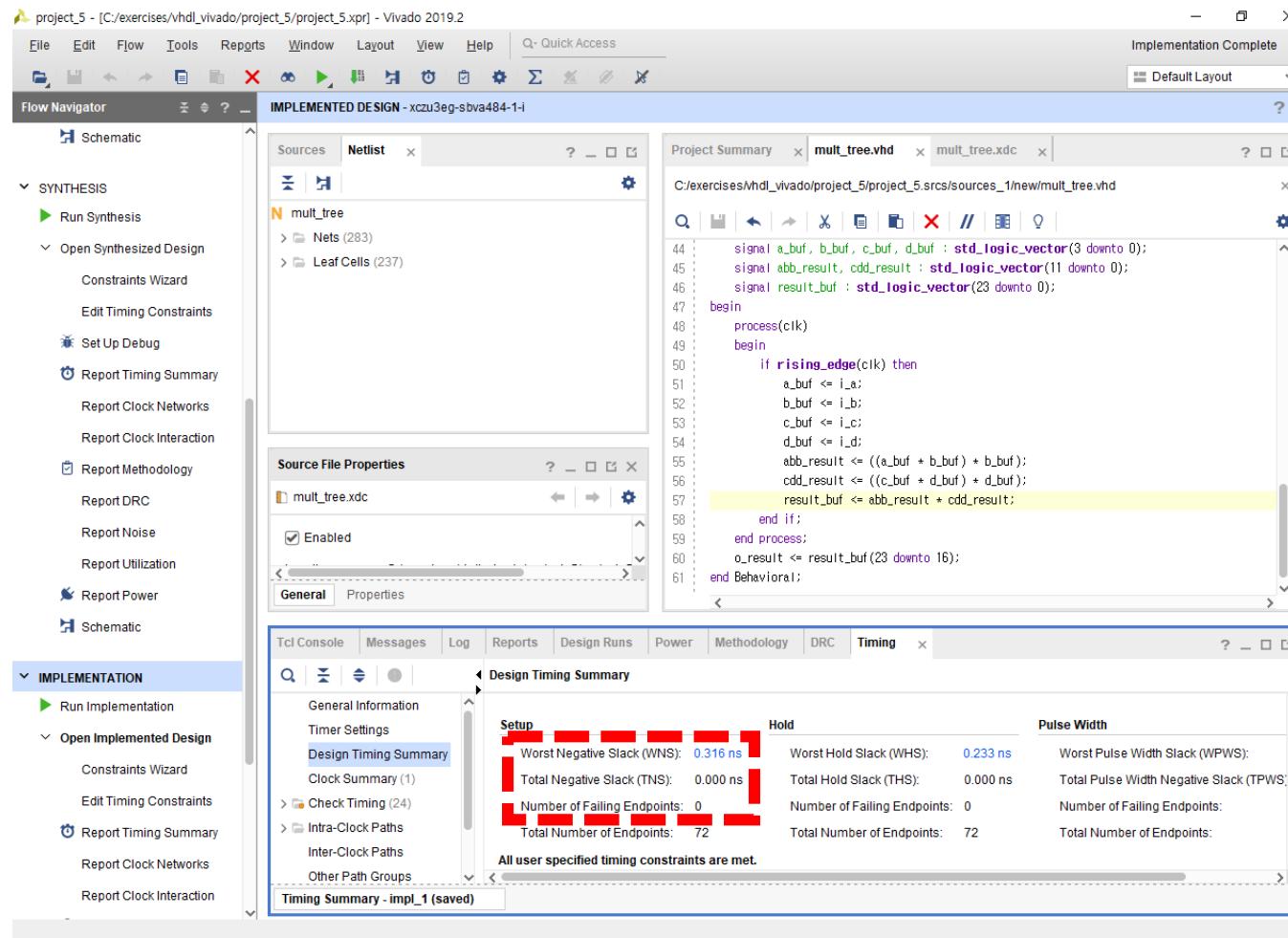
- ❖ Sources Window의 mult\_tree.vhd파일을 더블 클릭하여 Editor Window에 연다.
- ❖ mult\_tree 모듈을 우측의 코드와 같이 수정한다.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mult_tree is
    Port(
        clk : in std_logic;
        i_a, i_b, i_c, i_d : in std_logic_vector(3 downto 0);
        o_result : out std_logic_vector(7 downto 0)
    );
end mult_tree;
```

```
architecture Behavioral of mult_tree is
    signal a_buf, b_buf, c_buf, d_buf : std_logic_vector(3 downto 0);
    signal abb_result, cdd_result : std_logic_vector(11 downto 0);
    signal result_buf : std_logic_vector(23 downto 0);
begin
    process(clk)
    begin
        if rising_edge(clk) then
            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 if;
    end process;
    o_result <= result_buf(23 downto 16);
end Behavioral;
```

# 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 TNS값이 0으로 Slack이 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

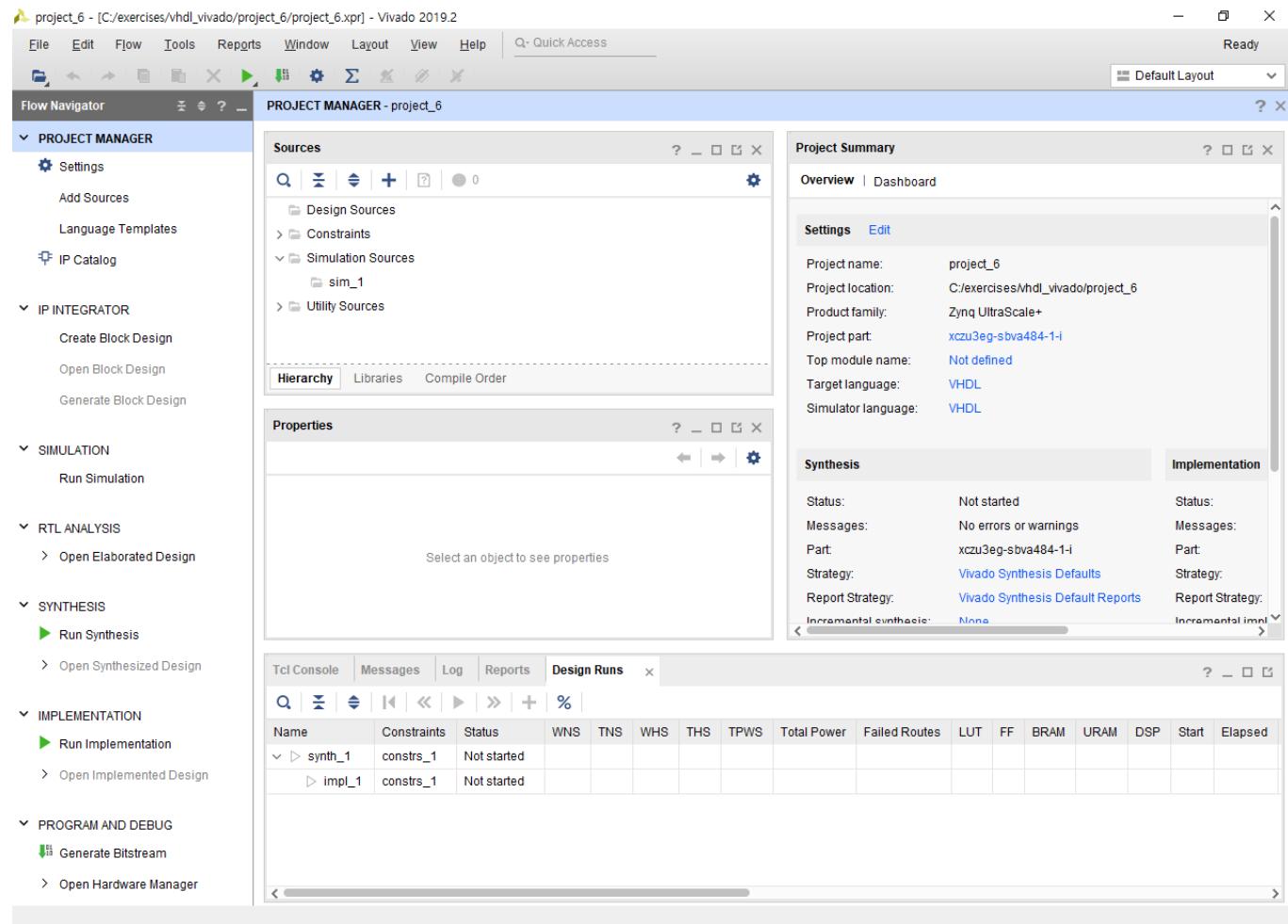


- ❖ Digilent의 휴대용 오실로스코프인 Analog Discovery 2 이다.
- ❖ Analog Discovery 2를 사용하기 위해서는 WaveForms 라는 PC용 무료 소프트웨어를 웹에서 다운로드 받아서 설치하면 사용 가능하다.
- ❖ Analog Discovery 2와 WaveForms는 USB 2.0 인터페이스를 통해 연동된다.
- ❖ [https://www.inipro.net/goods/goods\\_view.php?goodsNo=1000617163](https://www.inipro.net/goods/goods_view.php?goodsNo=1000617163)에서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

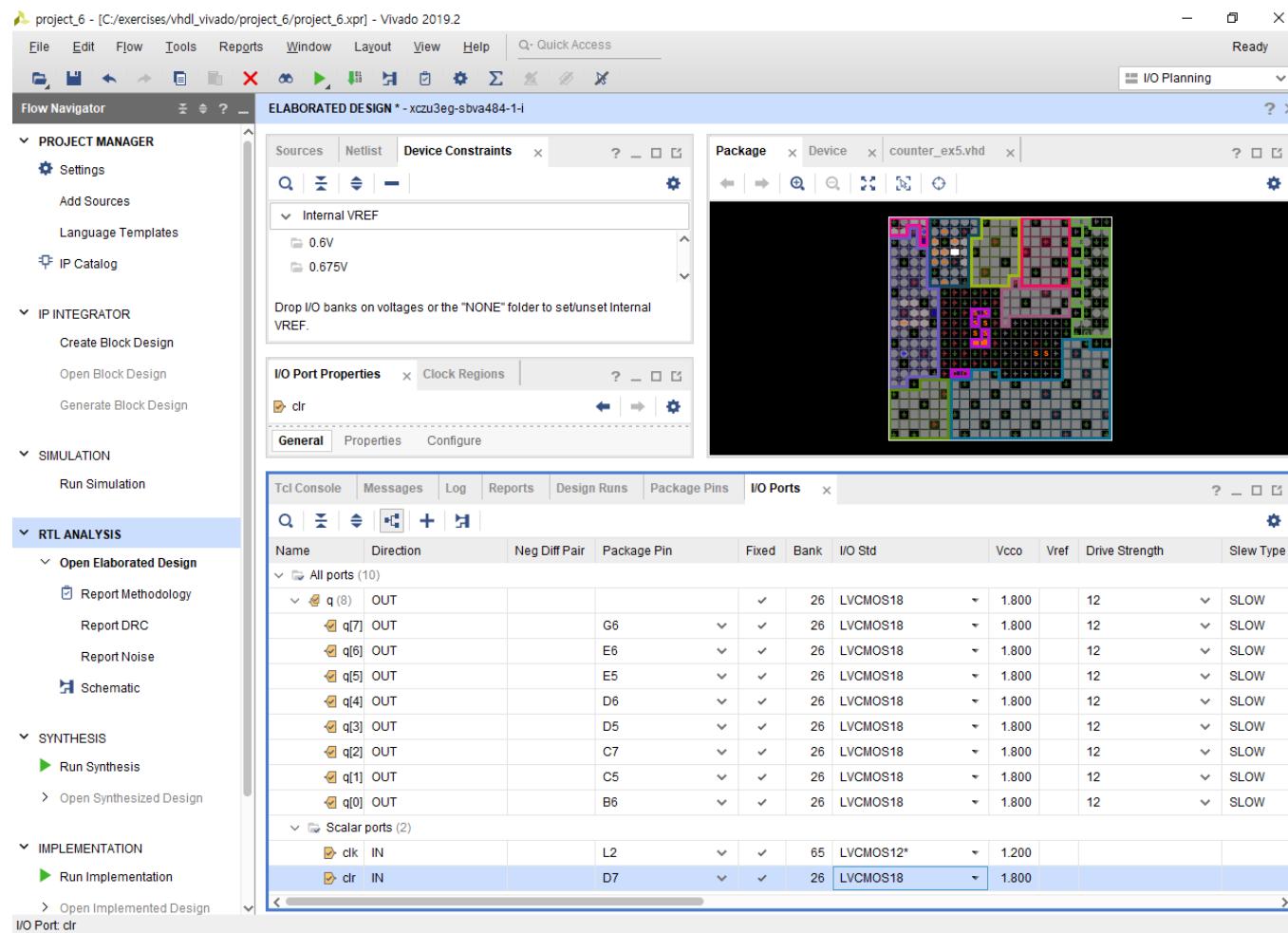
# Step 2 Add Design Source in Vivado Project

- ❖ Chapter 2 Vivado Design Flow의 Step 2를 참고하여 counter\_ex5.vhd 파일을 프로젝트에 추가한다.
- ❖ Sources Window의 counter\_ex5.vhd 파일을 더블 클릭하여 Editor Window에 연 후 우측 예제 코드를 입력한다.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity counter_ex5 is
    Port (
        clk, clr : in std_logic;
        q : out std_logic_vector(7 downto 0)
    );
end counter_ex5;
```

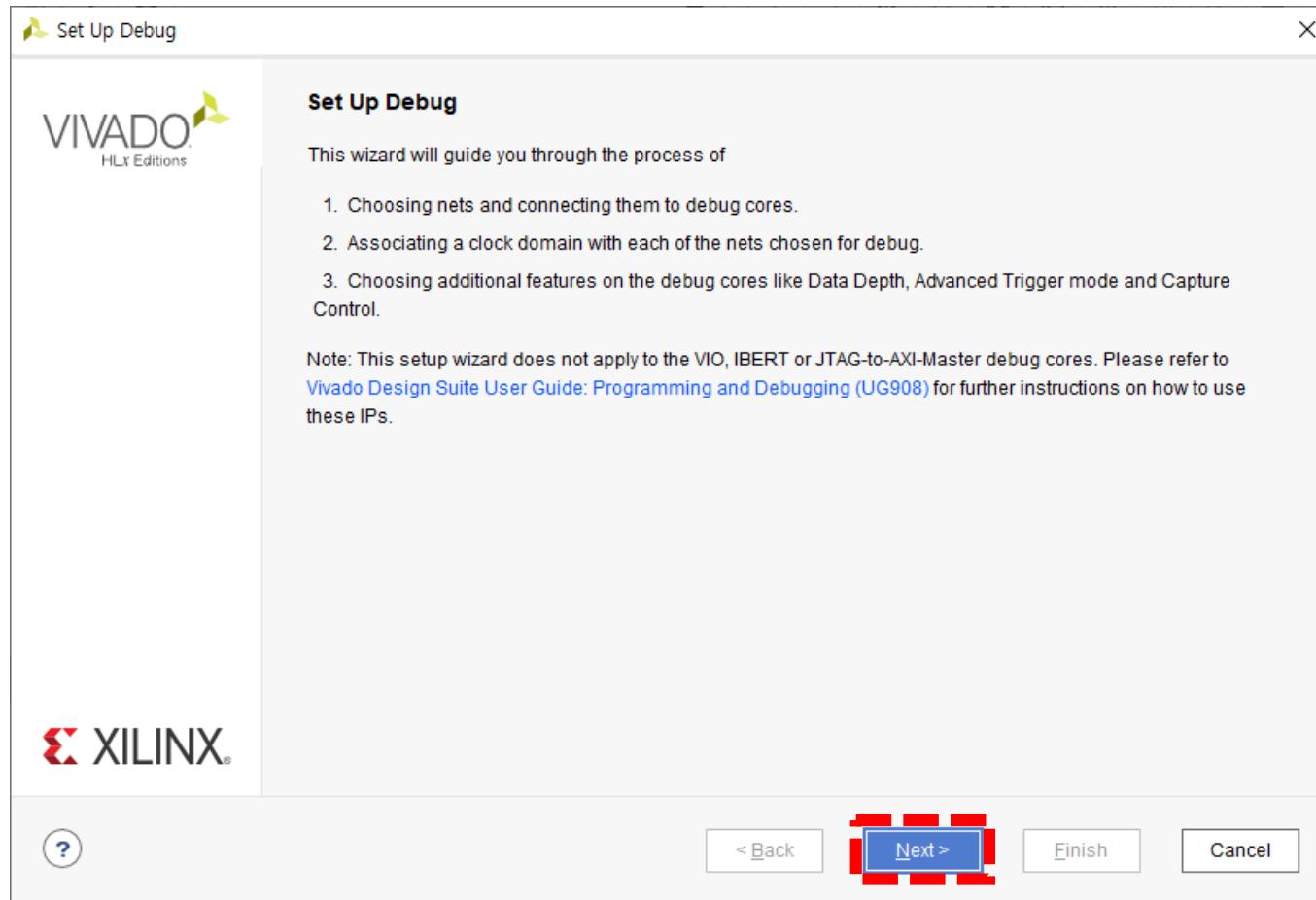
```
architecture Behavioral of counter_ex5 is
    signal cnt : std_logic_vector(31 downto 0);
    attribute mark_debug : string;
    attribute mark_debug of clr : signal is "true";
    attribute mark_debug of q : signal is "true";
begin
    process(clk,clr)
    begin
        if clr = '1' then
            cnt <= x"00000000";
        elsif rising_edge(clk) then
            cnt <= cnt + 1;
        end if;
    end process;
    q <= cnt;
end Behavioral;
```

# Step 3 Add Constraints



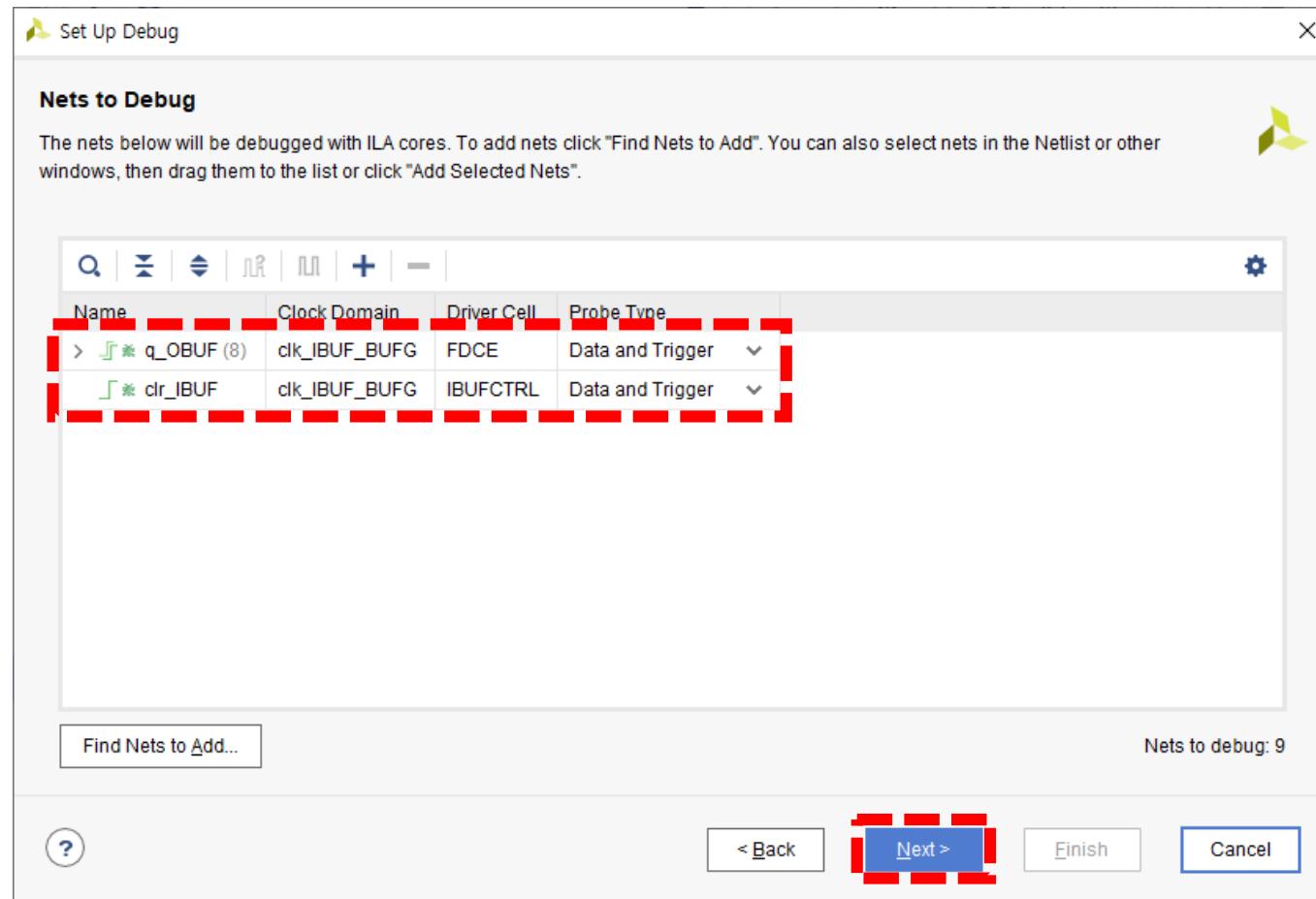
- ❖ Chapter 2 Vivado Design Flow의 Step 5를 참고하여 counter\_ex5.xdc 파일을 프로젝트에 추가한다.
- ❖ Chapter 2 Vivado Design Flow의 Step 5를 참고하여 clk 포트는 L2핀에 연결하고 clr 포트는 PMOD\_A커넥터 중 하나에 연결하고 q는 PMOD\_B커넥터에 연결한다.
- ❖ clk의 I/O Std는 LVCMOS12로 Constraints하고 clr, led포트는 LVCMOS18로 Constraints해준다.

# Step 4 Add ILA on Netlist 1



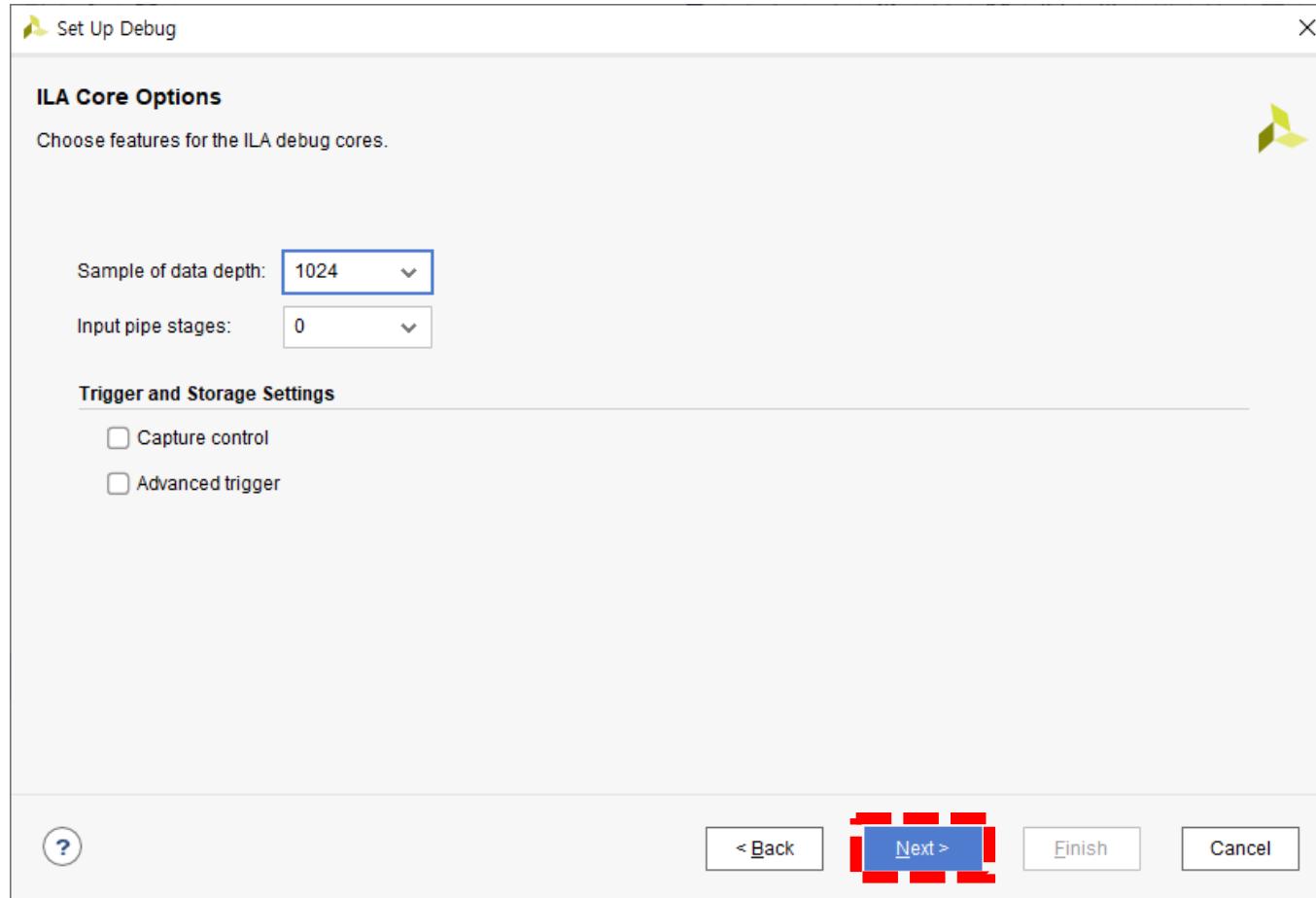
- ❖ Flow Navigator ⇒ Synthesis  
⇒ Run Synthesis를 클릭하여 디자인을 합성한다.
- ❖ Flow Navigator ⇒ Synthesis  
⇒ Open Synthesized Design  
⇒ Set Up Debug를 클릭한다.
- ❖ Set Up Debug Window가 나오면 Next 버튼을 클릭한다.

# Step 4 Add ILA on Netlist 2



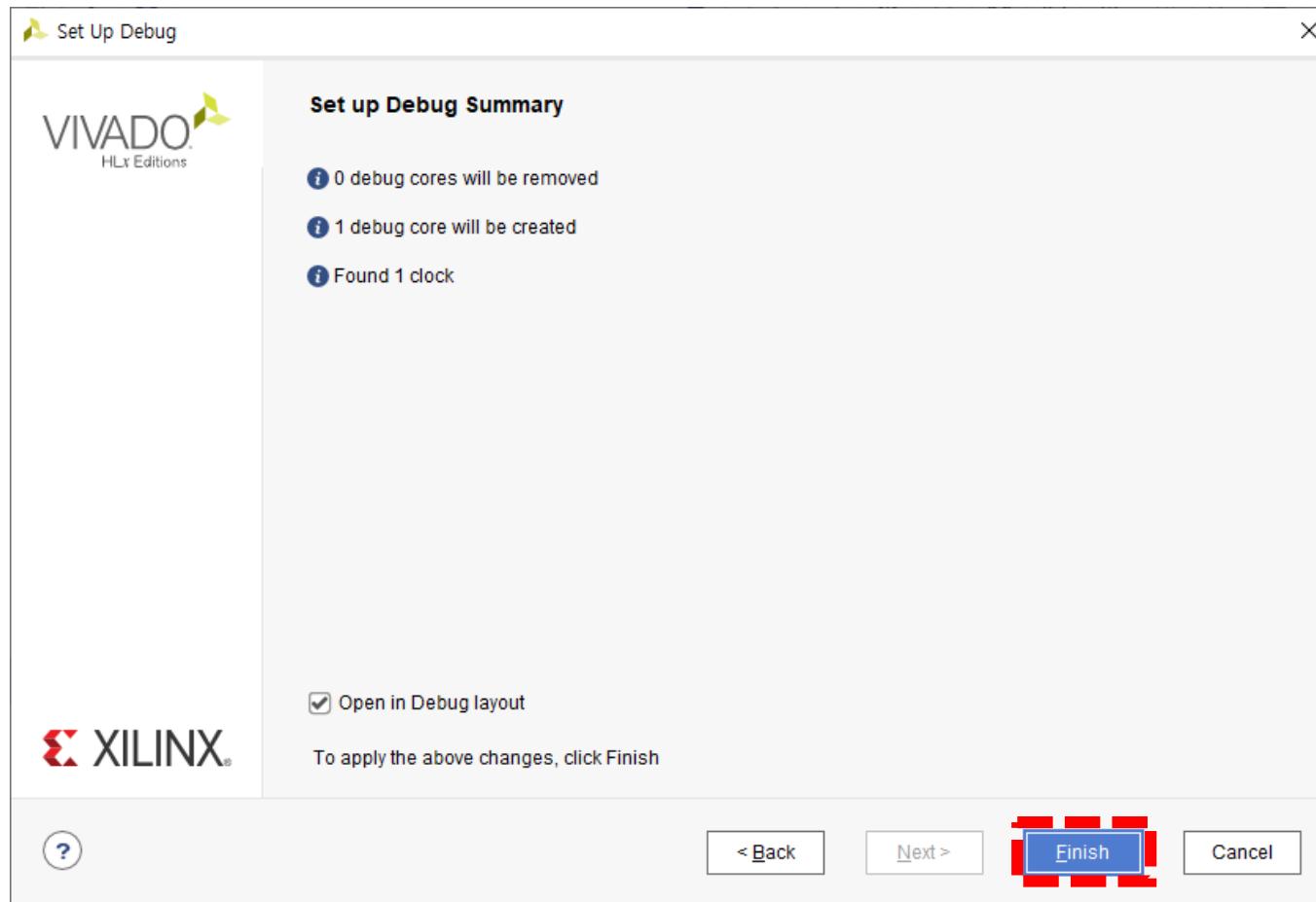
❖ 디버거로 계측 할 신호를 추가하는 창이 나오면 clr, q 신호를 추가한 후 Next 버튼을 클릭한다.

# Step 4 Add ILA on Netlist 3



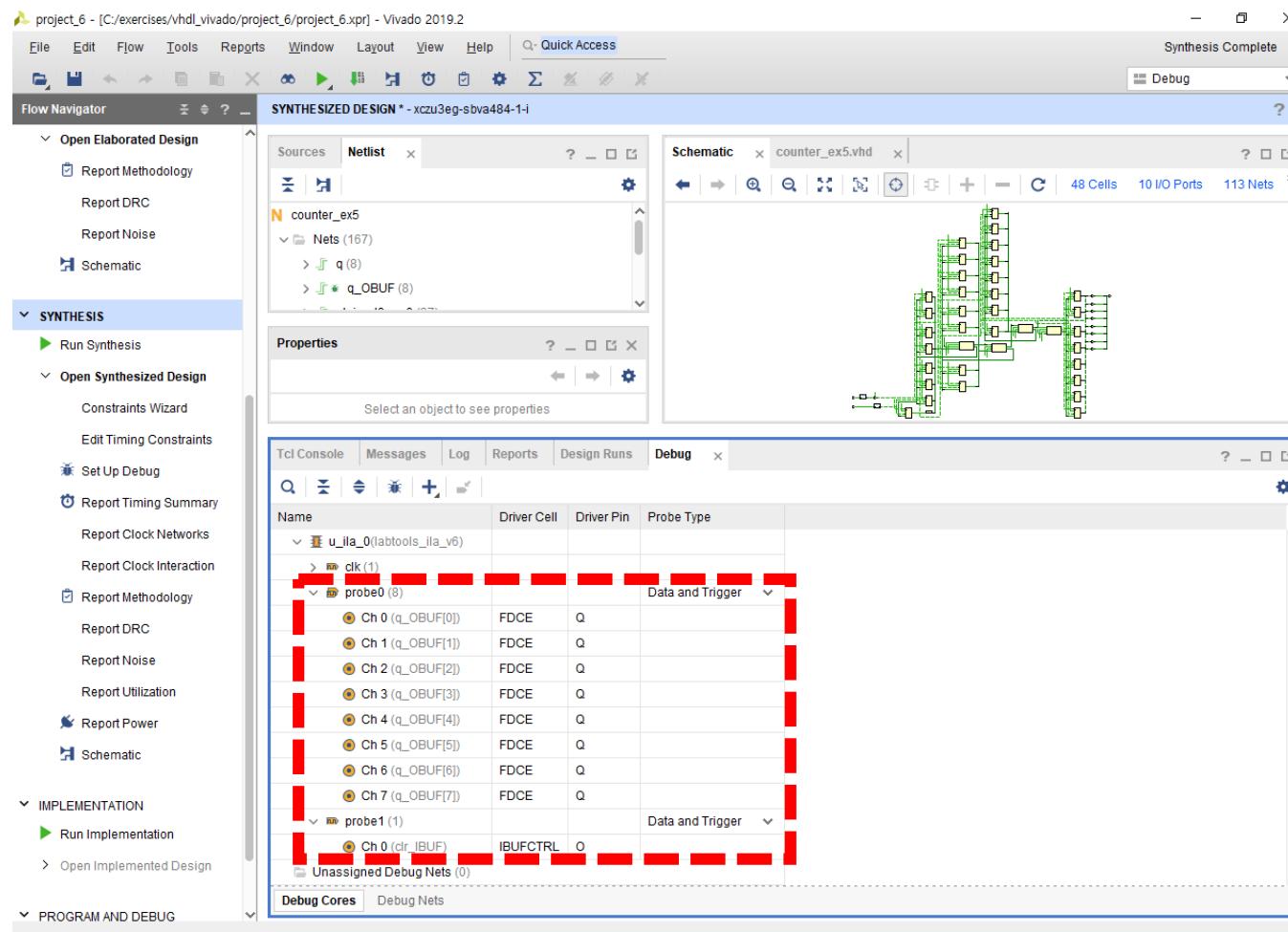
❖ Sample of data depth, Input pipe stages 등을 설정하는 창이 나오면 디폴트 상태 그대로 두고 Next 버튼을 클릭한다.

# Step 4 Add ILA on Netlist 4



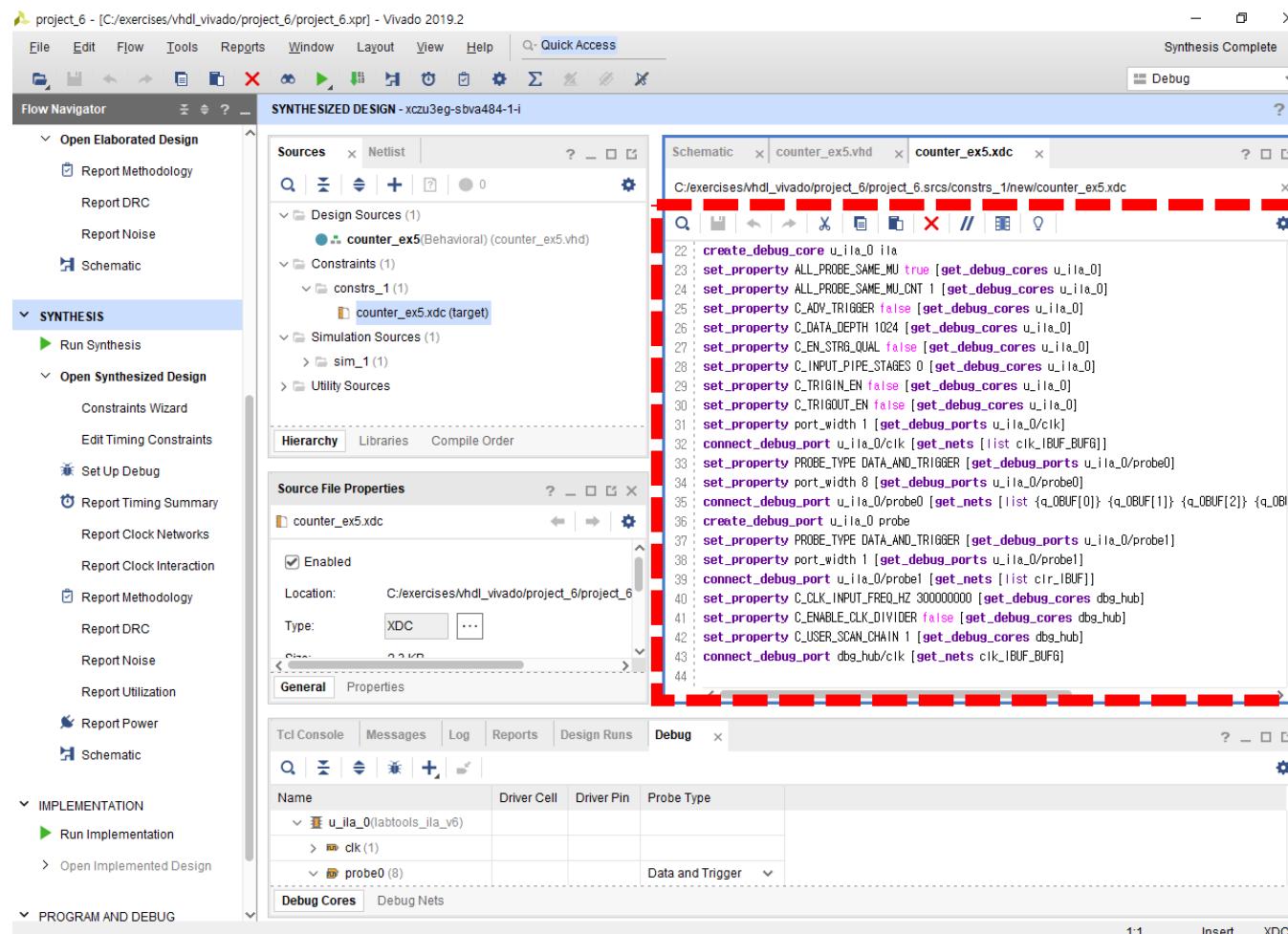
❖ Summary Window가 나오면  
설정 내용을 확인한 후 Finish  
버튼을 클릭한다.

# Step 4 Add ILA on Netlist 5



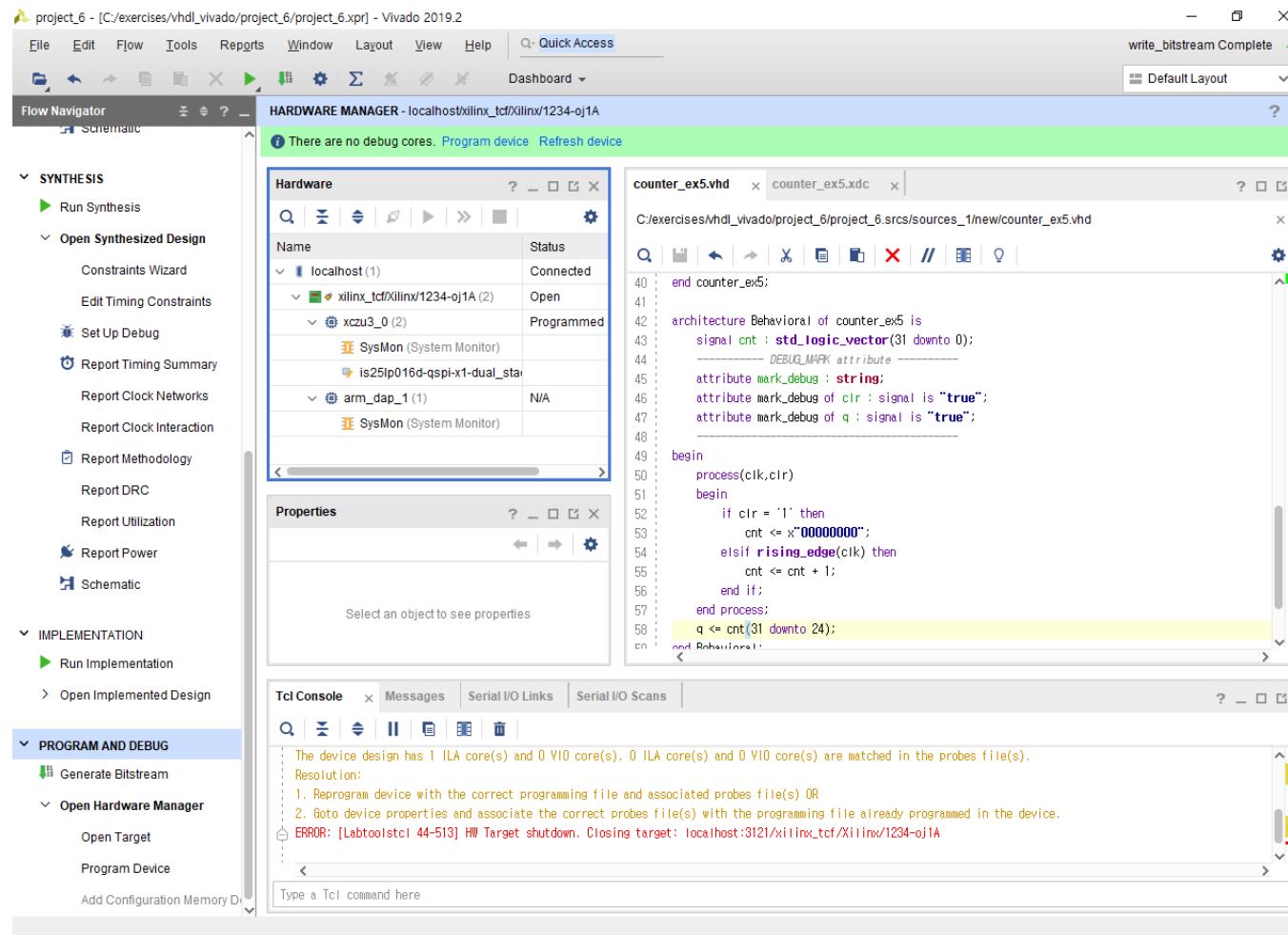
❖ Vivado 하단 Window의 Debug 탭을 보면 probe0와 probe1에 q와 clr이 Data and Trigger Probe Type으로 설정되어 있는 것을 확인할 수 있다.

# 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

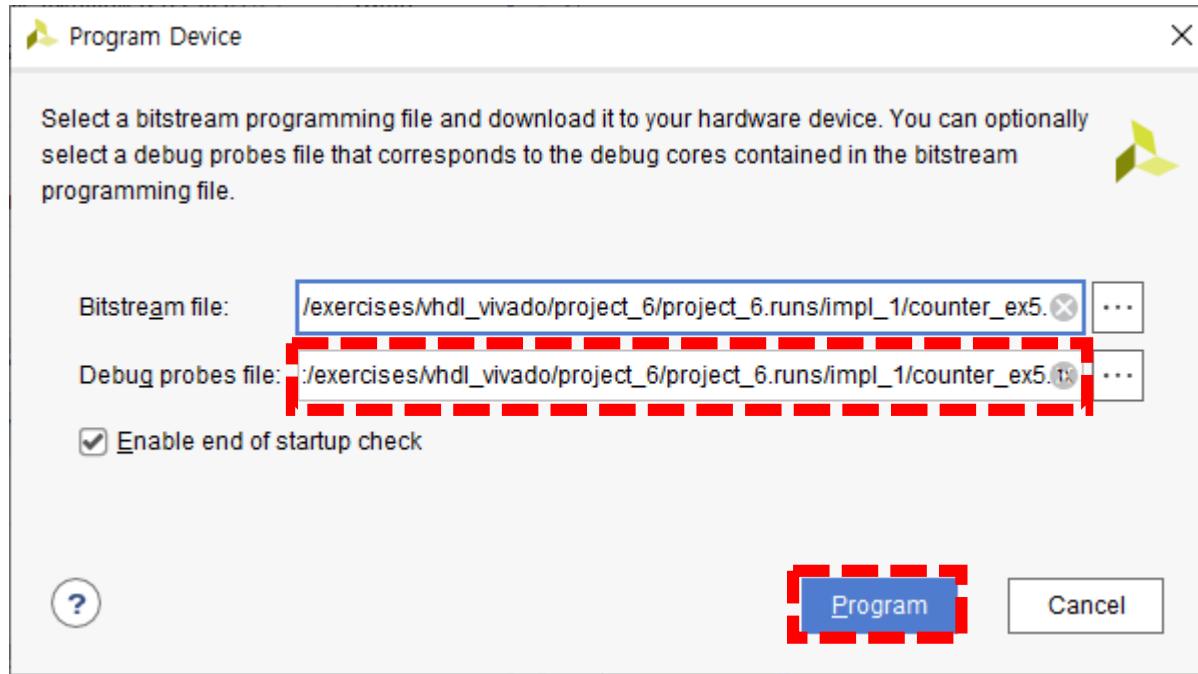


## ❖ Chapter 2 Vivado Design

Flow의 Step 6를 참고하여 Bitstream을 생성한다.

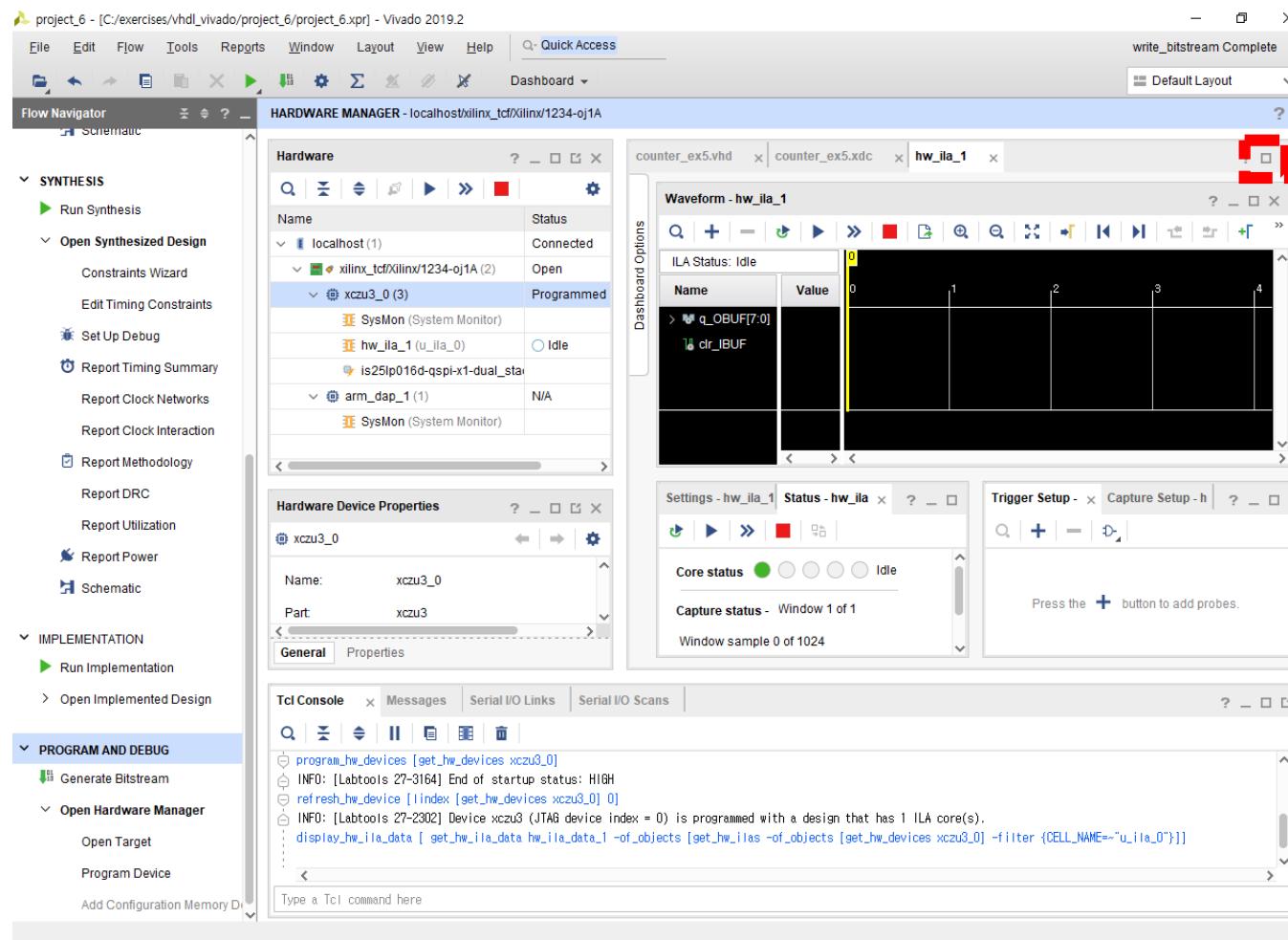
❖ Device Programming을 위해 Flow Navigator ⇒ PROGRAM AND DEBUG ⇒ Open Hardware Manager를 클릭한다.

# Step 5 Generating Bitstream and Programming Device 2



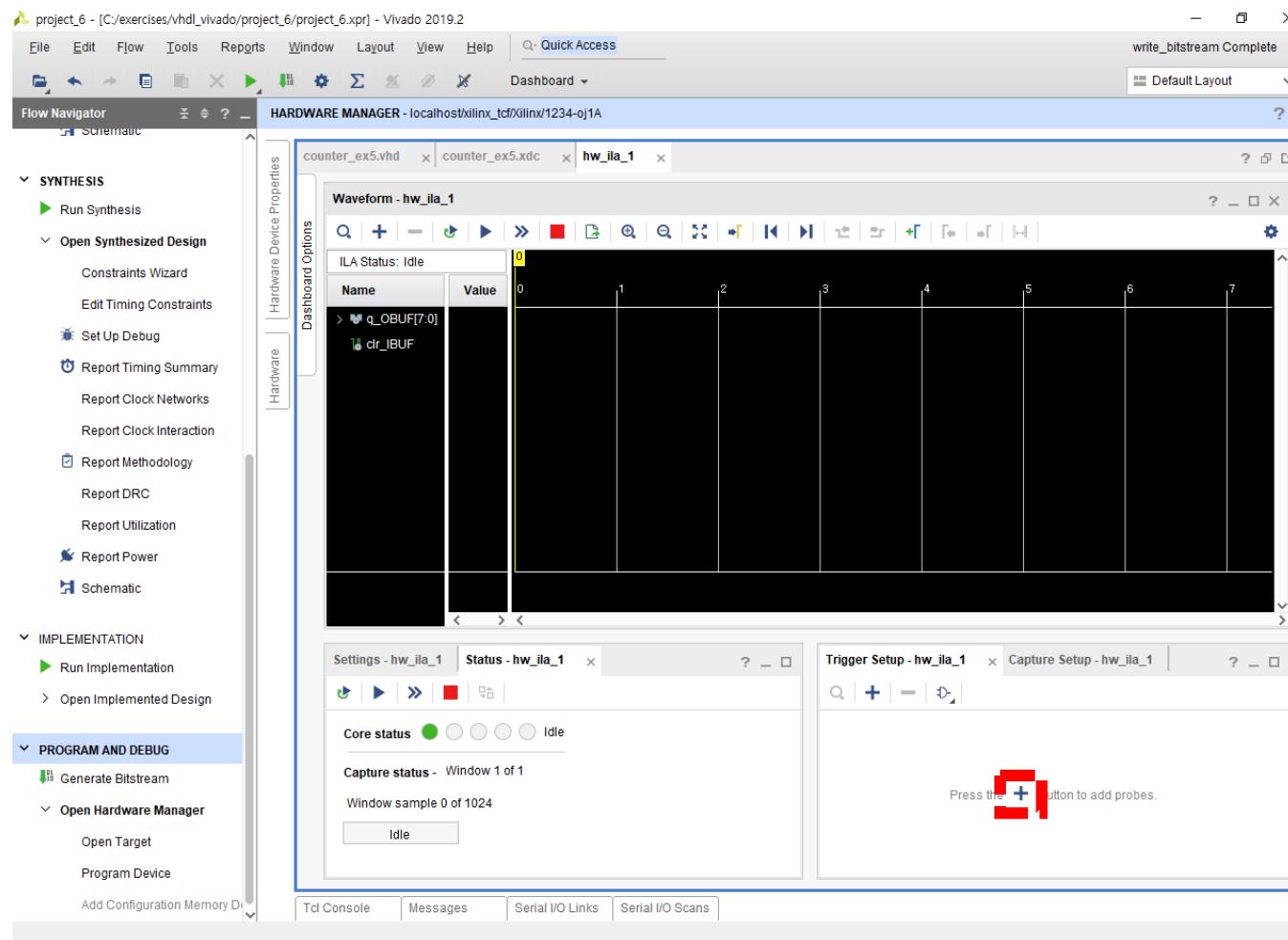
- ❖ Program device 버튼을 클릭하여 Program Device Window가 나오면 Bitstream file과 함께 디폴트로 Debug probes file이 입력되어 있는 것을 확인할 수 있다.
- ❖ Program 버튼을 클릭하여 Bitstream과 Debug probes를 프로그래밍한다.

# Step 7 Debug Signal using ILA 1



❖ 프로그래밍이 완료되어  
hw\_ila\_1 Window가 나오면 를 클릭하여 화면을 확장한다.

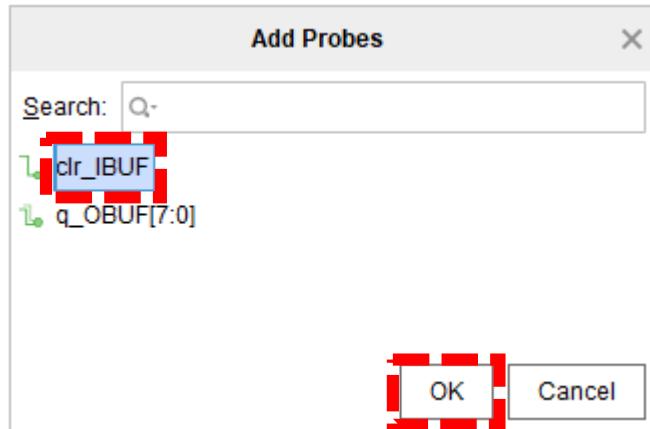
# Step 7 Debug Signal using ILA 2



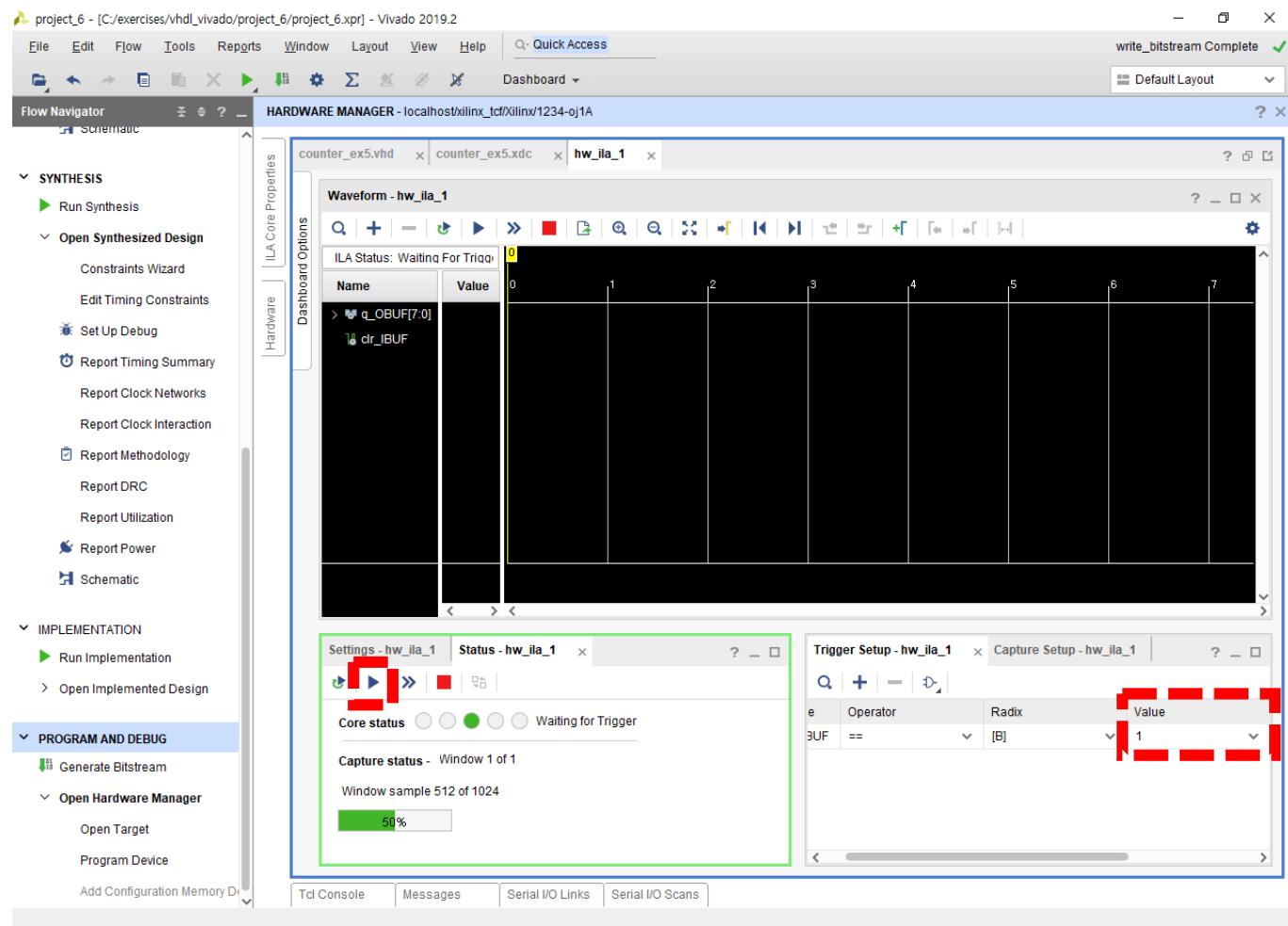
❖ 우측 하단 Window의 Trigger  
Setup – hw\_il\_1 탭 안에 +  
버튼을 클릭한다.

# Step 7 Debug Signal using ILA 3

❖ Add Probes Window가 나오면 clr\_IBUF 선택한 후 OK버튼을 클릭한다.

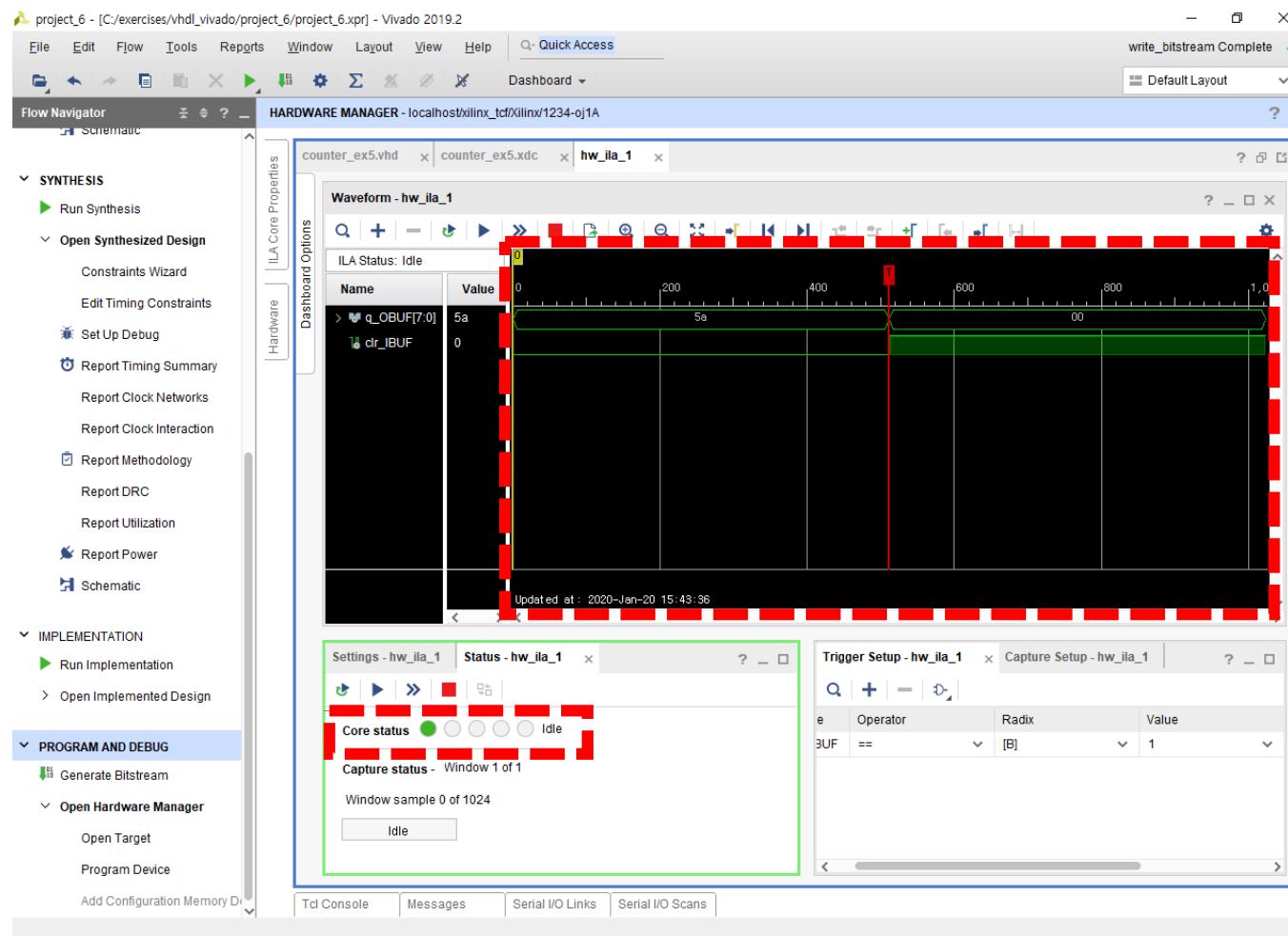


# Step 7 Debug Signal using ILA 4



- ❖ 우측 하단 Window의 Trigger Setup
  - hw\_ila\_1 탭 안에 clr\_IBUF가 추가된 것을 확인할 수 있다.
- ❖ clr\_IBUF의 Value를 1로 설정한다.  
(※ clr\_IBUF의 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\_IBUF의 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 Pmod 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 VHDL로 어떻게 표현하여 사용하는지 알아본다.

# Memory Example 1

- ❖ 메모리 예를 보면 mem\_array 로 width가 16이고 depth가 8인 메모리 타입을 선언 후에 mem을 mem\_array로 선언하여 메모리를 표현하였다. (※ mem은 16-bit 크기의 데이터를 8개 저장할 수 있는 메모리로 선언된 것이다.)
- ❖ 프로세스는 선언된 메모리에 어떻게 읽고 쓰는지에 대한 내용을 표현하였다.
- ❖ clk의 상승 에지에 동기되어 읽고 쓰기가 동작되며 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.
- ❖ 메모리를 읽고 쓰는 주소를 표현할 때 conv\_integer(address)로 표현하였다. (※ conv\_integer은 std\_logic\_vector 데이터 타입을 integer 데이터 타입으로 바꾸어 주는 함수로 STD\_LOGIC\_UNSIGNED 패키지 안에 정의되어 있어서 맨 위에 STD\_LOGIC\_UNSIGNED 패키지가 선언되어 있는 것을 확인할 수 있다.)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mem_ex1 is
    port(clk,wr_en : in std_logic;
          data : in std_logic_vector(15 downto 0);
          address : in std_logic_vector(2 downto 0);
          mem_out : out std_logic_vector(15 downto 0));
end mem_ex1;

architecture Behavioral of mem_ex1 is
    type mem_array is array (0 to 7) of std_logic_vector (15 downto 0);
    signal mem : mem_array;
begin
    process(clk,wr_en)
    begin
        if rising_edge(clk) then
            if wr_en='1' then
                mem(conv_integer(address)) <= data;
            end if;
            mem_out <= mem(conv_integer(address));
        end if;
    end process;
end Behavioral;
```

# Memory Example 2

- ❖ 우측 메모리 예를 보면 data 포트로 입력된 데이터는 wr\_addr 포트로 입력되는 주소에 쓰여 지고 rd\_addr 포트로 입력되는 주소의 메모리 값은 mem\_out 포트로 출력된다.
- ❖ Address Port만 둘로 나누어져 있지만 Data Port 또는 Control Port들도 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 있으며 Dual-Port Memory라고 부른다.)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mem_ex2 is
    port(clk,wr_en : in std_logic;
          data : in std_logic_vector(15 downto 0);
          rd_addr, wr_addr : in std_logic_vector(2 downto 0);
          mem_out : out std_logic_vector(15 downto 0));
end mem_ex2;

architecture Behavioral of mem_ex2 is
    type mem_array is array (0 to 7) of std_logic_vector (15 downto 0);
    signal mem : mem_array;
begin
    process(clk,wr_en)
    begin
        if rising_edge(clk) then
            if wr_en='1' then
                mem(conv_integer(wr_addr)) <= data;
            end if;
            mem_out <= mem(conv_integer(rd_addr));
        end if;
    end process;
end Behavioral;
```

# Memory Example 3

- ❖ 우측 메모리 예는 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 사용할 수 있다.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity mem_ex3 is
    port( wr_clk, rd_clk, wr_en : in std_logic;
          data : in std_logic_vector(15 downto 0);
          rd_addr, wr_addr : in std_logic_vector(2 downto 0);
          mem_out : out std_logic_vector(15 downto 0));
end mem_ex3;
```

```
architecture Behavioral of mem_ex3 is
begin
    type mem_array is array (0 to 7) of std_logic_vector (15 downto 0);
    signal mem : mem_array;
    begin
        process(wr_clk,wr_en)
        begin
            if rising_edge(wr_clk) then
                if wr_en='1' then
                    mem(conv_integer(wr_addr)) <= data;
                end if;
            end if;
        end process;

        process(rd_clk)
        begin
            if rising_edge(rd_clk) then
                mem_out <= mem(conv_integer(rd_addr));
            end if;
        end process;
    end Behavioral;
```

# Memory Example 4

- ❖ 메모리를 생성하였을 때 메모리의 초기값을 설정하고 싶다면 우측 메모리 예와 같이 초기값을 설정할 수 있다.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity mem_ex4 is
    port( wr_clk, rd_clk, wr_en : in std_logic;
          data : in std_logic_vector(15 downto 0);
          rd_addr, wr_addr : in std_logic_vector(2 downto 0);
          mem_out : out std_logic_vector(15 downto 0));
end mem_ex4;
```

```
architecture Behavioral of mem_ex4 is
begin
    type mem_array is array (0 to 7) of std_logic_vector (15 downto 0);
    signal mem : mem_array := ( 0 => x"abcd", 1 => x"3abf",
                                2 => x"1234", 3 => x"4567", 4 => x"7347",
                                5 => x"a4fb", 6 => x"0993", 7 => x"af34");
begin
    process(wr_clk,wr_en)
    begin
        if rising_edge(wr_clk) then
            if wr_en='1' then
                mem(conv_integer(wr_addr)) <= data;
            end if;
        end if;
    end process;

    process(rd_clk)
    begin
        if rising_edge(rd_clk) then
            mem_out <= mem(conv_integer(rd_addr));
        end if;
    end process;
end Behavioral;
```

# Memory Example 5

- ❖ 우측 코드 예제에서는 2차원 배열을 사용하여 메모리를 선언하였다.
- ❖ 기존의 1차원 배열로 선언한 것과 비교해보면 생성되는 메모리는 동일하지만 메모리를 Addressing하는 주소가 2개로 달라짐을 알 수 있다.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mem_ex5 is
    port( wr_clk, rd_clk, wr_en : in std_logic;
          data : in std_logic_vector(15 downto 0);
          rd_addr0, wr_addr0 : in std_logic_vector(0 downto 0);
          rd_addr1, wr_addr1 : in std_logic_vector(1 downto 0);
          mem_out : out std_logic_vector(15 downto 0));
end mem_ex5;
```

```
architecture Behavioral of mem_ex5 is
begin
    type mem_array is array (0 to 1,0 to 3) of std_logic_vector (15 downto 0);
    signal mem : mem_array :=
        ( 0 => (x"abcd",x"3abf",x"1234",x"4567"),
          1 => (x"7347",x"a4fb",x"0993",x"af34") );
begin
    process(wr_clk,wr_en)
    begin
        if rising_edge(wr_clk) then
            if wr_en='1' then
                mem(conv_integer(wr_addr0), conv_integer(wr_addr1)) <= data;
            end if;
        end if;
    end process;

    process(rd_clk)
    begin
        if rising_edge(rd_clk) then
            mem_out <= mem(conv_integer(rd_addr0), conv_integer(rd_addr1));
        end if;
    end process;
end Behavioral;
```

# 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.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity rom_ex1 is
    port( clk : in std_logic;
          addr : in std_logic_vector(2 downto 0);
          rom_out : out std_logic_vector(15 downto 0));
end rom_ex1;
architecture Behavioral of rom_ex1 is
    type mem_array is array (0 to 7) of std_logic_vector (15 downto 0);
    signal mem : mem_array := ( 0 => x"abcd", 1 => x"3abf",
                                2 => x"1234", 3 => x"4567", 4 => x"7347",
                                5 => x"a4fb", 6 => x"0993", 7 => x"af34");
begin
    process(clk)
        begin
            if rising_edge(clk) then
                rom_out <= mem(conv_integer(addr));
            end if;
        end process;
    end Behavioral;
```

# ROM Example 2

- ❖ case Statement를 사용하여 ROM을 표현한 모듈이다.
- ❖ rom\_ex1 모듈과 rom\_ex2 모듈은 서로 표현한 코드는 다르지만 합성한 결과물은 동일한 회로가 된다.
- ❖ Vivado에서 Flow Navigator ⇒ Synthesis ⇒ Schematic을 클릭하여 합성된 회로를 비교해보면 확인할 수 있다.

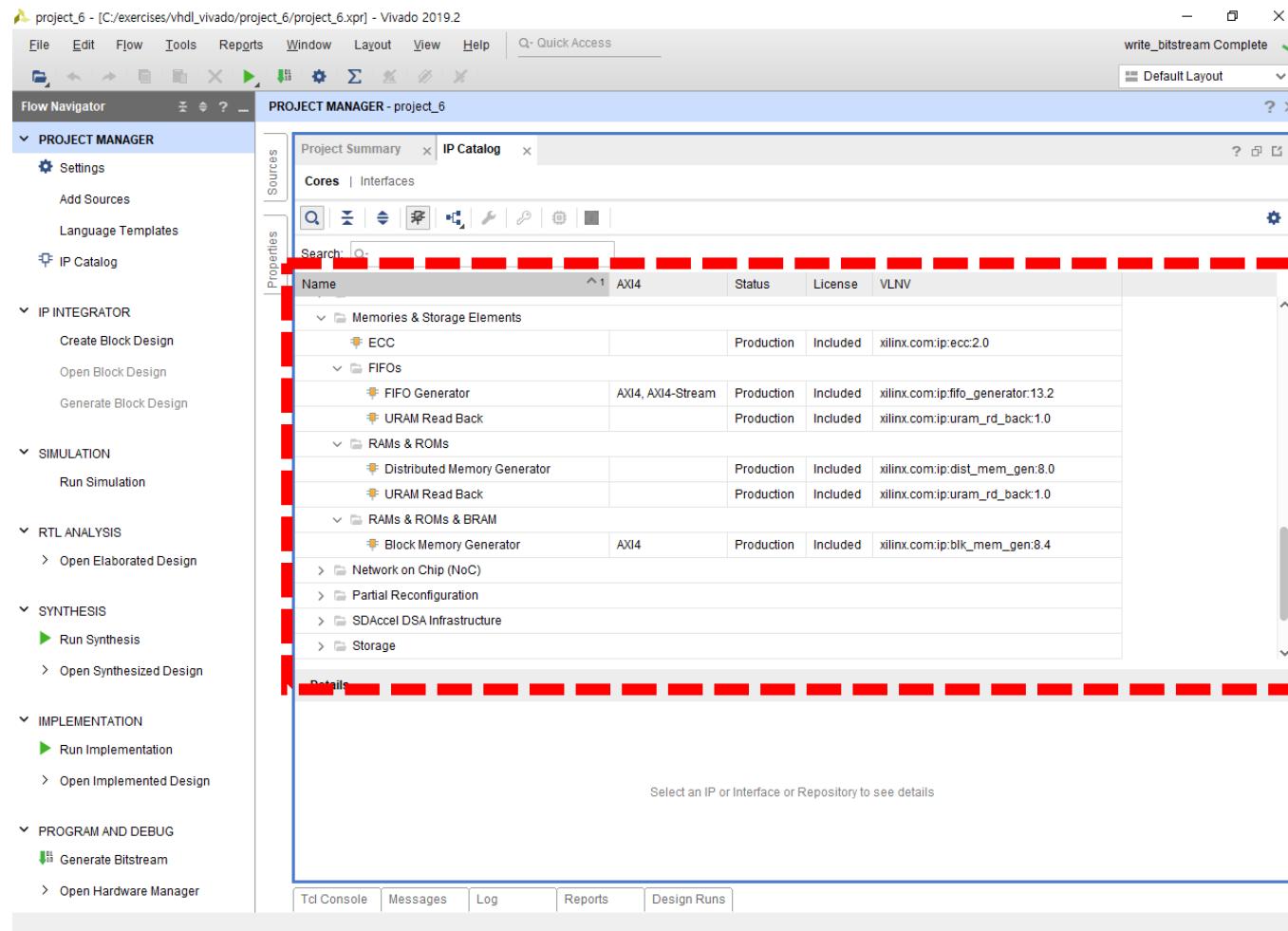
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity rom_ex2 is
    port( clk : in std_logic;
          addr : in std_logic_vector(2 downto 0);
          rom_out : out std_logic_vector(15 downto 0));
end rom_ex2;
```

```
architecture Behavioral of rom_ex2 is
begin
    process(clk)
    begin
        if rising_edge(clk) then
            case addr is
                when "000" => rom_out <= x"abcd";
                when "001" => rom_out <= x"3abf";
                when "010" => rom_out <= x"1234";
                when "011" => rom_out <= x"4567";
                when "100" => rom_out <= x"7347";
                when "101" => rom_out <= x"a4fb";
                when "110" => rom_out <= x"0993";
                when "111" => rom_out <= x"af34";
                when others => rom_out <= x"abcd";
            end case;
        end if;
    end process;
end Behavioral;
```

# 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.

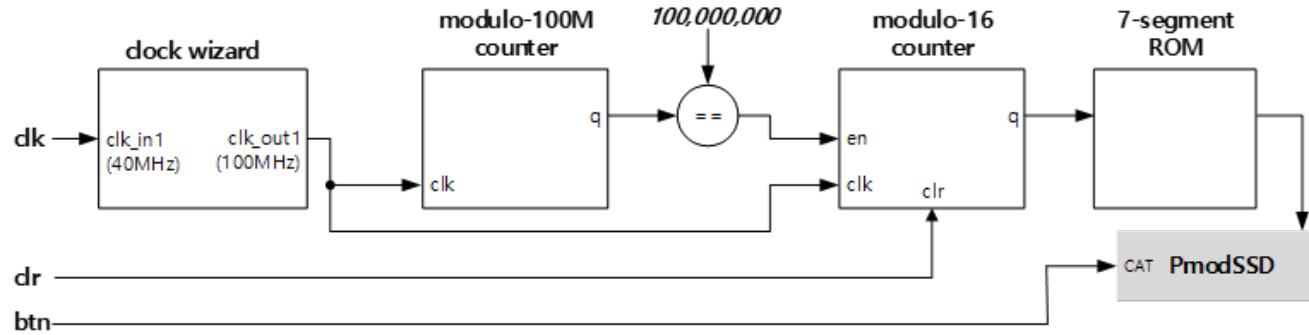
## Memory IP Cores 2

- ❖ 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 VHDL로 표현하여 FSM 설계를 완성할 수 있다.

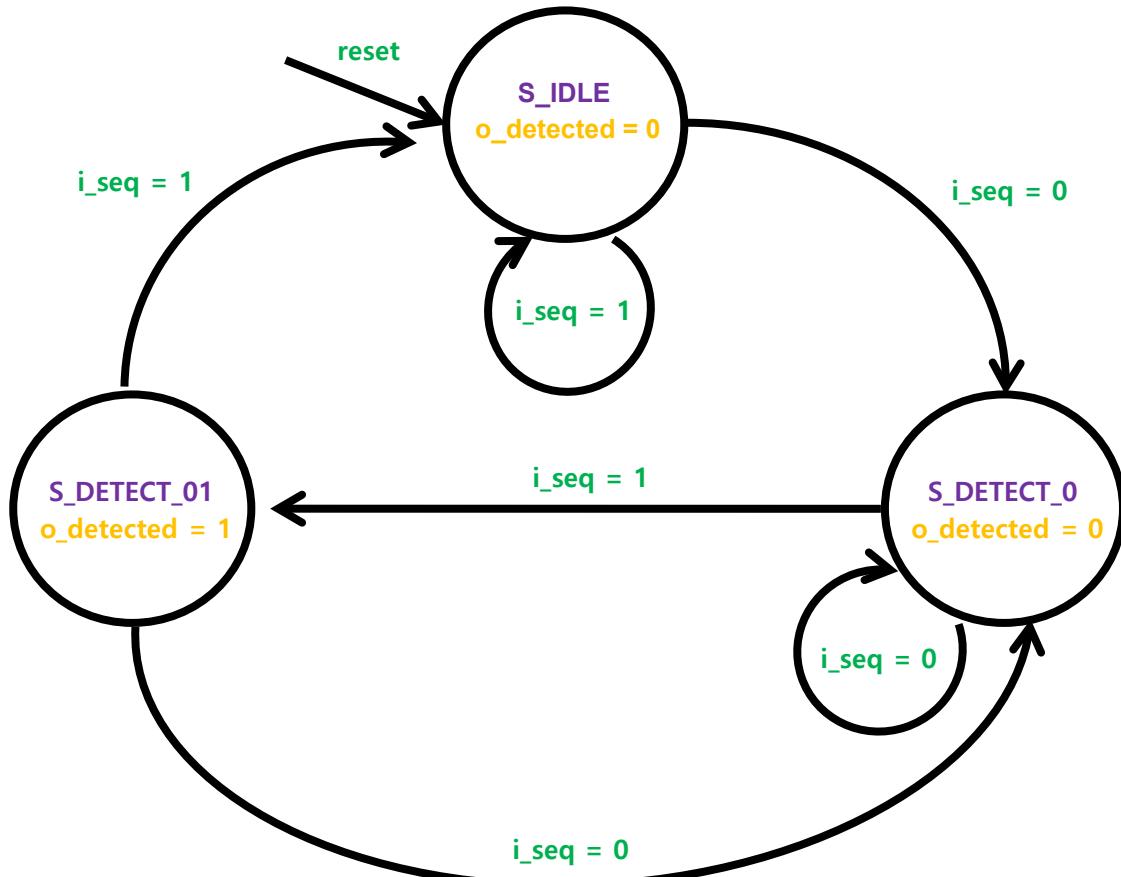
# FSM Example

- ❖ 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

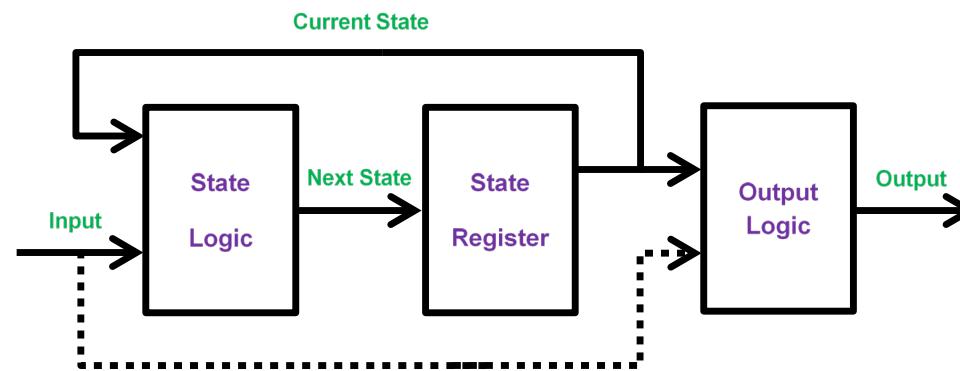
- ❖ 스테이트 디어그램을 작성하려면 제일 먼저 스테이트를 몇 개로 할 것인지 고려해봐야 한다.
- ❖ 회로가 리셋되거나 시퀀스를 처음부터 다시 기다리는 S\_IDLE상태, 시퀀스 "0"이 검출된 S\_DETECT\_0상태, 시퀀스 "01"이 검출된 S\_DETECT\_01상태로 표현할 수 있다.
- ❖ 이것을 스테이트 디어그램으로 표현하면 왼쪽 그림과 같다.

# 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\_IDLE**상태, 시퀀스 "0"이 검출된 **S\_DETECT\_0**상태, 시퀀스 "01"이 검출된 **S\_DETECT\_01**상태로 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.
- ❖ Readability를 위해 Enumerated Data Type으로 Current State와 Next State를 선언한다.
- ❖ 3개의 Block으로 표현된 Block Diagram 과 같이 3개의 Block을 각각 3개의 process로 표현해준다.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity detect_01_fsm is
    port(
        clk : in std_logic;
        rst : in std_logic;
        i_seq : in std_logic;
        o_detected : out std_logic
    );
end detect_01_fsm;

architecture Behavioral of detect_01_fsm is
    type state_type is (S_IDLE, S_DETECT_0, S_DETECT_01);
    signal current_state, next_state : state_type;
begin
```

# FSM Implementation 2

- ❖ State Logic Block을 하나의 Process로 표현한 코드이다.
- ❖ Sensitivity List에 State Logic 블록의 입력인 current\_state와 입력신호를 넣었다.
- ❖ 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.

```
process(current_state, i_seq)
begin
    next_state <= current_state;
    case current_state is
        when S_IDLE =>
            if i_seq = '0' then
                next_state <= S_DETECT_0;
            end if;
        when S_DETECT_0 =>
            if i_seq = '1' then
                next_state <= S_DETECT_01;
            end if;
        when S_DETECT_01 =>
            if i_seq = '0' then
                next_state <= S_DETECT_0;
            else
                next_state <= S_IDLE;
            end if;
        when others =>
            next_state <= S_IDLE;
    end case;
end process;
```

# FSM Implementation 3

- ❖ State Register Block을 하나의 Process로 표현한 코드이다.
- ❖ rst 신호가 '1'(High)이면 current\_state는 S\_IDLE상태가 되도록 설계한다.
- ❖ Register를 만들기 위해 if rising\_edge(clk) then 구문 안에서 Next State를 Current State에 Assign하였다.
- ❖ 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.

```
process(clk, rst)
begin
    if rst = '1' then
        current_state <= S_IDLE;
    elsif rising_edge(clk) then
        current_state <= next_state;
    end if;
end process;
```

# FSM Implementation 4

- ❖ Output Block을 하나의 Process로 표현한 코드이다.
- ❖ Sensitivity List에 Output Logic Block의 입력인 current\_state를 입력하였다.
- ❖ case Statement는 current\_state를 판별한 값에 따라서 Output Port인 o\_detected로 출력한다.

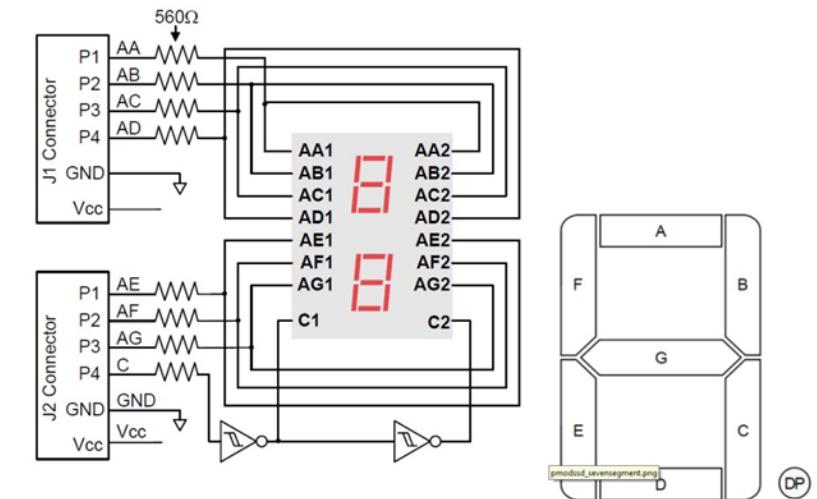
```
process(current_state)
begin
    case current_state is
        when S_IDLE => o_detected <= '0';
        when S_DETECT_0 => o_detected <= '0';
        when S_DETECT_01 => o_detected <= '1';
        when others => o_detected <= '0';
    end case;
end process;
```

# 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 Pmod BTN을 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 VHDL Advanced Syntax

- Generic
- Generate
- Subprogram

# Generic 1

- ❖ Generic은 Entity의 Port위에 작성하는 것으로 앞에서 작성한 코드와 같이 생략 가능하다.
- ❖ Generic은 모듈이나 IP의 재사용을 위해 사용되는 것으로 인스턴스 생성(Instantiation)을 할 때 파라미터 값을 전달하기 위한 용도로 사용한다.
- ❖ 우측 예제 코드에서는 엔터티 안에 generic안에 width가 16으로 선언되어 있는데 포트와 시그널 선언부에서 사용한 width는 16이 되어서 포트 q와 시그널 cnt는 16-bit 크기의 std\_logic\_vector로 선언되고 이 카운터 모듈을 다른 모듈에서 Instantiation할 때 generic안에 있는 선언되어 있는 파라미터들을 재설정하여 Instantiation 할 수 있다.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity counter is
    generic( width : integer := 16 );
    port( clk, clr : in std_logic;
          q : out std_logic_vector(width-1 downto 0));
end counter;

architecture Behavioral of counter is
    signal cnt : std_logic_vector(width-1 downto 0);
begin
    process(clk,clr)
    begin
        if clr='1' then
            cnt <= (others => '0');
        elsif rising_edge(clk) then
            cnt <= cnt + 1;
        end if;
    end process;
    q <= cnt;
end Behavioral;
```

# Generic 2

- ❖ counter\_top 모듈에서 counter 모듈을 2번 Instantiation하였다.
- ❖ Instantiation할 때 generic map 으로 설정한 값에 의해서 Instantiation된다.
- ❖ 첫 번째 Instantiation은 width가 16으로 설정된 Counter 모듈을 Instantiation하여 16-bit 카운터가 되고 두 번째 Instantiation 은 width가 32로 설정 된 Counter 모듈을 Instantiation하여 32-bit 카운터가 된다.
- ❖ IP를 Instantiation할 때에도 Generic을 사용하여 여러가지 설정을 통해 재사용한다.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity counter_top is
    port( clk, clr : in std_logic;
          q1 : out std_logic_vector(15 downto 0);
          q2 : out std_logic_vector(31 downto 0));
end counter_top;

architecture Behavioral of counter_top is
    component counter is
        generic( width : integer );
        port( clk, clr : in std_logic;
              q : out std_logic_vector(width-1 downto 0));
    end component;
begin
    u1 : counter
        generic map (width => 16)
        port map (clk => clk, clr => clr, q => q1);
    u2 : counter
        generic map (width => 32)
        port map (clk => clk, clr => clr, q => q2);
end Behavioral;
```

# Chapter 11 VHDL Advanced Syntax

- Generic
- Generate
- Subprogram

# Generate

- ❖ Generate Statement는 Process 밖에 표현한 Concurrent Statement들을 하드웨어로 생성할지를 결정하는 Statement이다.
- ❖ Generate Statement에는 If Generate If 와 For Generate Statement가 있다.
- ❖ If Generate는 If 뒤에 따라오는 식에 따라 생성할지 안 할지를 결정할 수 있는 Statement이고 For Generate는 뒤에 따라오는 식의 범위만큼 반복하여 여러 개를 생성할 수 있는 Statement이다.
- ❖ 오른쪽 코드 예제는 If generate와 For generate를 사용하여 shift register를 표현한 예제이다.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity shift_reg is
    port( clk, input : in std_logic;
          output : out std_logic);
end shift_reg;
architecture GEN of shift_reg is
    component flipflop is
        port( clk, d : in std_logic;
              q : out std_logic );
    end component;
    signal inter : std_logic_vector(14 downto 0);
begin
    for_gen : for i in 0 to 15 generate
        if_gen1: if i=0 generate
            U0 : flipflop port map (clk=>clk, d=>input, q=>inter(i));
        end generate;
        if_gen2: if i>0 and i<15 generate
            UX : flipflop port map (clk=>clk, d=>inter(i-1), q=>inter(i));
        end generate;
        if_gen3: if i=15 generate
            U15 : flipflop port map (clk=>clk, d=>inter(i-1), q=>output);
        end generate;
    end generate;
end GEN;
```

# Chapter 11 VHDL Advanced Syntax

- Generic
- Generate
- Subprogram

# Function

- ❖ function이라는 예약어를 쓰고 function 이름을 쓰고 function에 입력되는 파라미터들을 쓰고 return 예약어를 쓰고 return\_type을 쓴다.
- ❖ function과 begin사는 function의 선언부로 구현부에서 사용할 Variable들을 선언해 준다.
- ❖ begin과 end function사는 구현부로 function에서 구현할 기능을 순차 구문들을 넣어서 표현하고 마지막으로 function에서 리턴할 값을 return 예약어를 사용하여 return 한다.

```
function multiply_func (a,b : in std_logic_vector) return  
std_logic_vector is  
variable c : std_logic_vector(15 downto 0);  
begin  
c := a * b;  
return c;  
end function multiply_func;
```

# Procedure

- ❖ Procedure 예약어를 쓴 후 Procedure 이름을 쓰고 Procedure에서 사용할 입출력 파라미터를 쓴다.
- ❖ procedure와 begin 사이는 procedure의 선언부로 구현부에서 사용할 Variable들을 선언해 준다.
- ❖ begin과 end Procedure 사이는 구현부로 Procedure에서 구현할 기능을 순차 구문들을 넣어서 표현해 준다.

```
procedure multiply_proc ( a,b : in std_logic_vector;
                           c : out std_logic_vector ) is
begin
  c := a * b;
end procedure multiply_proc;
```

# Package 1

- ❖ std\_logic\_vector 데이터 타입은 기본적으로 사용되어지므로 std\_logic\_1164 패키지를 먼저 선언해 준다.
- ❖ 패키지는 Package와 Package Body로 나누어진다.
- ❖ Package는 선언부이고 Package Body는 구현부이다.
- ❖ 선언부에는 Data Type 또는 Subprogram을 선언하는 부분이고 구현부는 Subprogram들을 구현한 코드를 넣는 곳이다.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

package mypkg is
    function multiply_func (a,b : in std_logic_vector) return std_logic_vector;
    procedure multiply_proc (a,b : in std_logic_vector; c : out std_logic_vector);
end package mypkg;

package body mypkg is
    function multiply_func (a,b : in std_logic_vector) return std_logic_vector is
        variable c : std_logic_vector(15 downto 0);
    begin
        c := a * b;
        return c;
    end function multiply_func;
    procedure multiply_proc (a,b : in std_logic_vector; c : out std_logic_vector) is
    begin
        c := a * b;
    end procedure multiply_proc;
end package body mypkg;
```

# Package 2

- ❖ `use work.mypkg.all;` 라는 Statement를 통해 Package를 불러올 수 있다.
- ❖ `work`는 현재 프로젝트에 포함된 라이브러리를 의미하며 프로젝트에 포함되어 있는 패키지는 위와 같은 Statement를 사용하여 Package를 불러올 수 있다.
- ❖ 우측 코드 예제와 같이 Package를 불러온 후 Package에 있는 Function과 Procedure를 사용할 수 있다.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.mypkg.all;

entity subprogram_ex is
    port( a,b,c : in std_logic_vector(7 downto 0);
          result_func : out std_logic_vector(15 downto 0);
          result_proc : out std_logic_vector(15 downto 0) );
end subprogram_ex;

architecture Behavioral of subprogram_ex is
begin
    result_func <= multiply_func(a,b);
    process(b,c)
        variable temp : std_logic_vector(15 downto 0);
    begin
        multiply_proc(b,c,temp);
        result_proc <= temp;
    end process;
end Behavioral;
```

# 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 600x480 60Hz의 Pixel Clock(pclk)인 25MHz 클럭 신호를 만들고 오른쪽 코드 예제와 같이 코딩을 한다. (※ VGA 출력 신호에 대한 타이밍은 VESA Monitor Timing Standard를 참조한다.)
- ❖ Horizontal Sync Timing과 Vertical Sync Timing 맞추어 HS와 VS신호를 출력하고 있다.
- ❖ Porch를 계산하여 hcnt와 vcnt가 원하는 카운트 값이 되었을 때 R, G, B 출력이 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.

```
process(pclk)
begin
    if rising_edge(pclk) then
        hcnt <= hcnt + 1;
        if hcnt = 799 then
            hcnt <= 0;
            vcnt <= vcnt + 1;
            if vcnt = 524 then vcnt <= 0;
            end if;
        end if;
        if (hcnt>=656 and hcnt<752) then vga_hs <= '0';
        else vga_hs <= '1';
        end if;
        if(vcnt>=490 and vcnt<492) then vga_vs <= '0';
        else vga_vs <= '1';
        end if;
        if(hcnt>=0 and hcnt<640 and vcnt>=0 and vcnt<480) then
            if (hcnt = 300) then vga_o <= x"f00";
            else vga_o <= X"000";
            end if;
            else vga_o <= X"000";
            end if;
        end if;
    end process;
```