

DataLab. Internship Program (2023 Summer)

Week 2

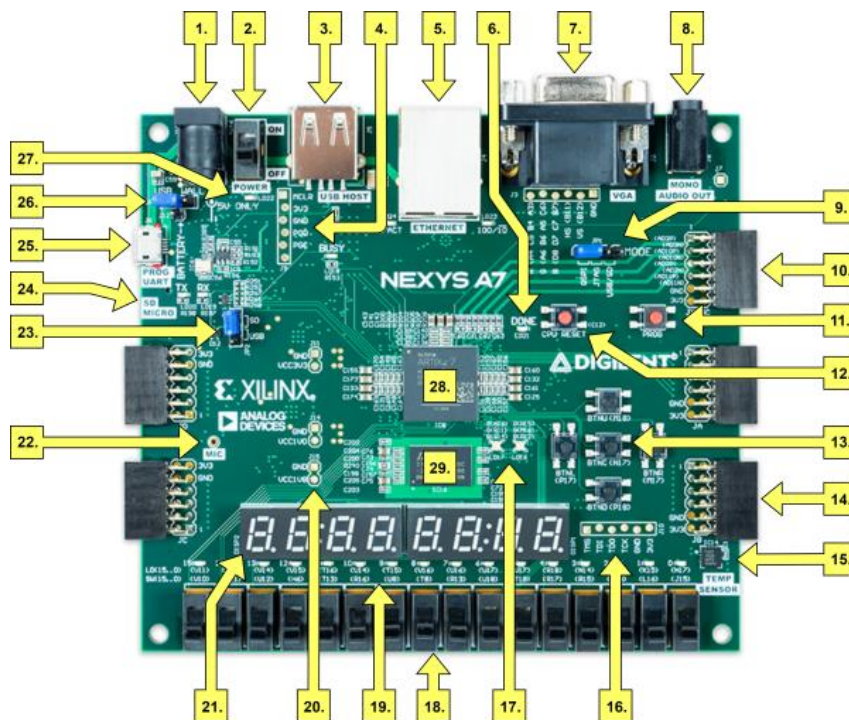
Tutor: 배은태

실습

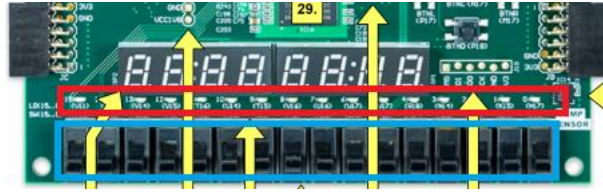
Nexys A7 보드와 Vivado tool을 이용하여 실습을 진행합니다.

실습1: LED Blink

이번 실습에서 사용할 보드는 Digilent사의 Nexys A7-100T입니다. Nexys A7은 Xilinx Artix-7 FPGA를 탑재하고 있습니다. 50T와 100T는 FPGA의 하드웨어 리소스 차이입니다. Nexys A7 보드는 다음 그림과 같이 구성되어 있습니다.



가운데 있는 정사각형의 칩이 FPGA입니다. 전원 공급은 직류전원(1번)이나 USB(25번)를 통해 가능합니다(USB 케이블은 보드랑 동봉되어 있습니다). 또한 보드를 PC와 연결하여 FPGA에 회로(bitstream)를 올리는 작업(configuration 또는 program)도 이 USB(25번)를 통해 할 수 있습니다. 전원 스위치(2번)를 이용하여 전원을 공급하거나 차단할 수 있습니다.



하단에는 16개의 LED와(19번, 빨간색으로 강조), 16개의 슬라이드 스위치(18번, 하늘색으로 강조)가 장착되어 있습니다. LED는 오른쪽에서 왼쪽 순서대로 LED[0], LED[1], LED[2], ..., LED[15]로 배치되어 있습니다. 슬라이드 스위치 역시 마찬가지로 오른쪽에서 왼쪽 순서대로 올라갑니다(비트열을 생각하시면 좋을 것 같습니다).

예제 1번에서는 카운터를 이용하여 일정한 주기로 LED의 on/off 신호를 생성하는 blink라는 모듈을 직접 만들어 가장 오른쪽에 있는 LED[0]와 연결시키도록 하겠습니다.

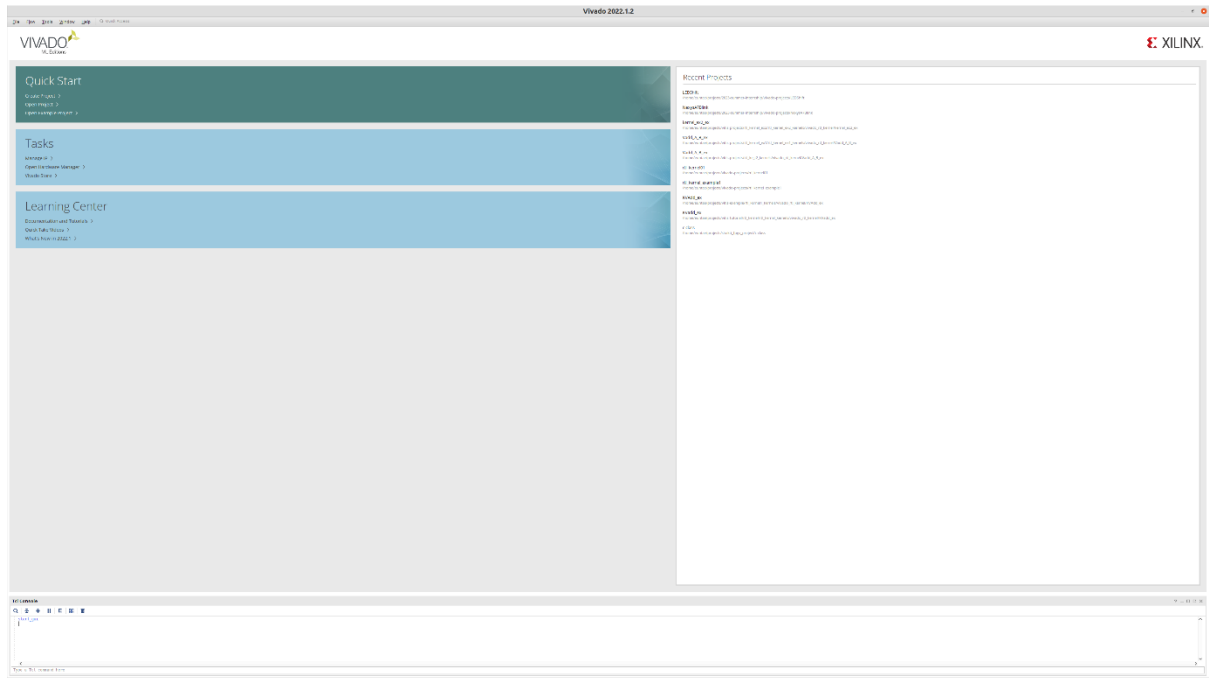
프로젝트 생성

이제 Vivado를 키고 프로젝트를 생성하겠습니다. 터미널에 vivado라고만 입력해도 Vivado가 실행됩니다. 프로젝트가 생성되기를 원하는 임의의 위치에서 vivado를 실행합니다. 참고로 vivado를 실행한 디렉토리는 vivado.log, vivado.jou라는 파일이 자동으로 생성됩니다. 이 둘은 각각 로그와 저널(journal) 파일로, 여기에는 작업 중 발생한 로그, 그리고 작업 중 실행한 명령이 Tcl 명령어 형태로 기록됩니다.

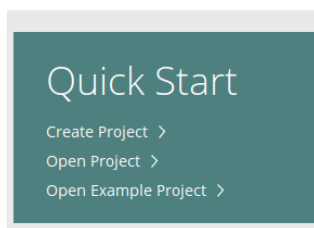
```
euntae@datalab: ~/projects/2023-summer-internship/vivado-projects
euntae@datalab: ~/projects/2023-summer-internship/vivado-projects 111x24
(base) euntae@datalab:~/projects/2023-summer-internship/vivado-projects$ vivado &
[2] 227366
(base) euntae@datalab:~/projects/2023-summer-internship/vivado-projects$
***** Vivado v2022.1.2 (64-bit)
***** SW Build 3605665 on Fri Aug 5 22:52:02 MDT 2022
***** IP Build 3603185 on Sat Aug 6 04:07:44 MDT 2022
** Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.

start_gui
█
```

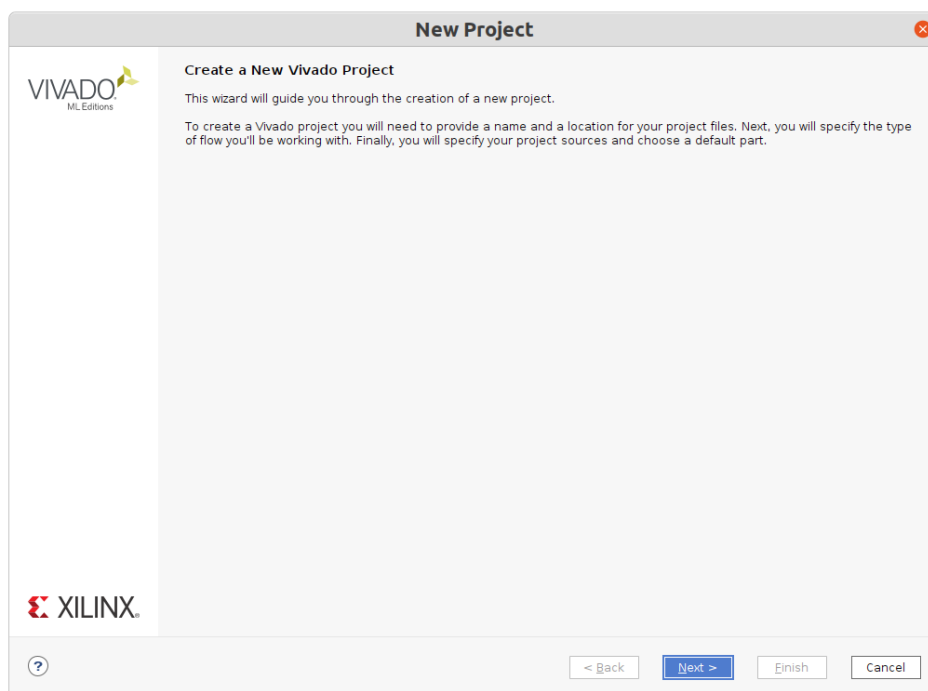
Vivado의 초기 화면은 다음과 같습니다.



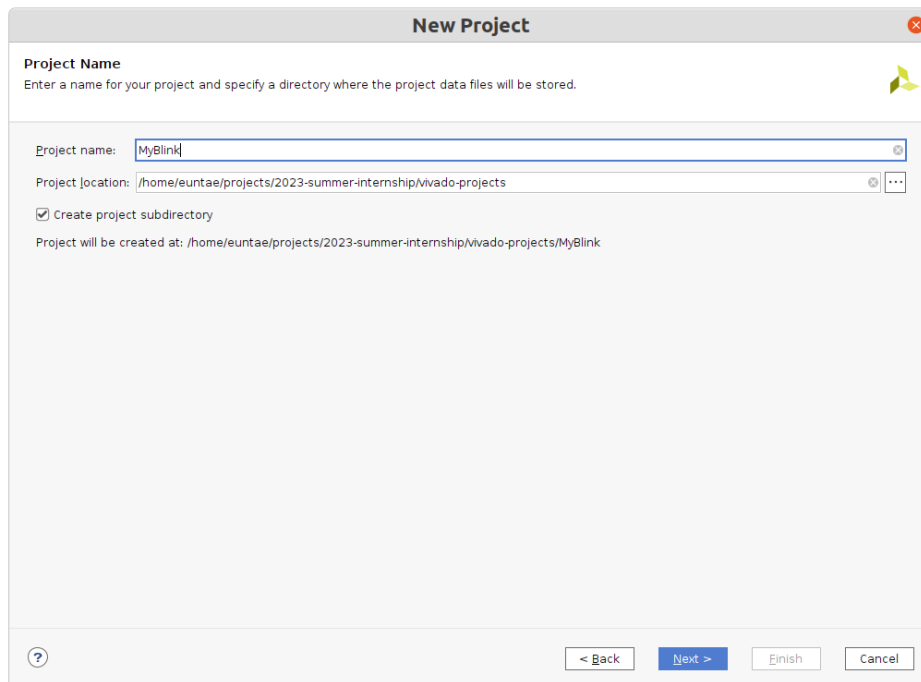
프로젝트 생성을 위해 상단에 있는 **Create Project**를 선택합니다.



그럼 다음과 같이 프로젝트 생성 마법사가 실행됩니다.

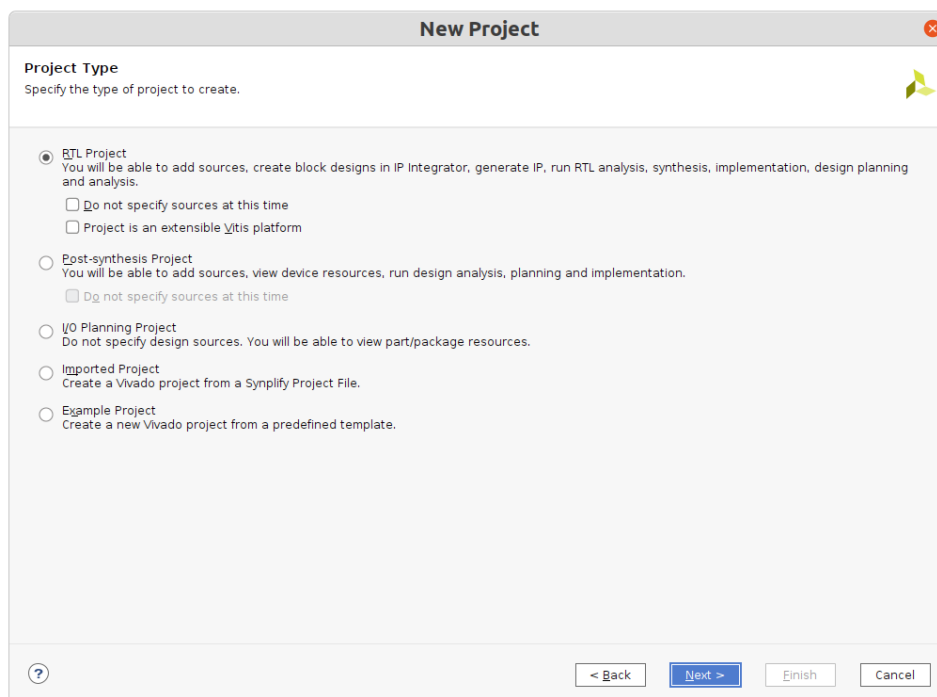


원하는 임의의 프로젝트 이름을 입력합니다.



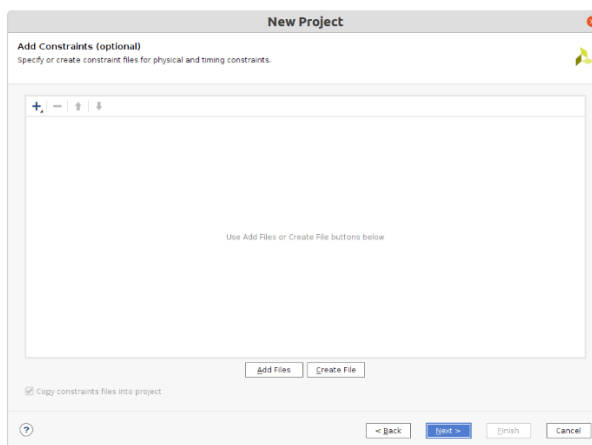
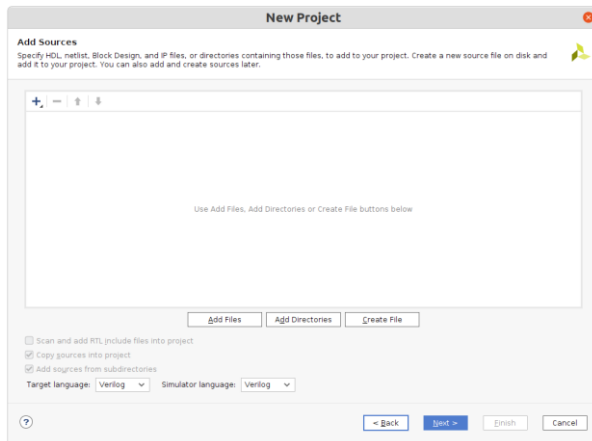
The 'New Project' dialog box is shown at the 'Project Name' step. The title bar says 'New Project'. Below the title bar, it says 'Project Name' and 'Enter a name for your project and specify a directory where the project data files will be stored.' There is a text input field for 'Project name:' containing 'MyBlink'. Below it is a text input field for 'Project location:' containing '/home/euntae/projects/2023-summer-internship/vivado-projects'. There is a checkbox labeled 'Create project subdirectory' which is checked. Below the checkbox, it says 'Project will be created at: /home/euntae/projects/2023-summer-internship/vivado-projects/MyBlink'. At the bottom, there are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.


RTL Project를 선택합니다.



The 'New Project' dialog box is shown at the 'Project Type' step. The title bar says 'New Project'. Below the title bar, it says 'Project Type' and 'Specify the type of project to create.' There are five radio button options: 'RTL Project' (selected), 'Post-synthesis Project', 'I/O Planning Project', 'Imported Project', and 'Example Project'. Each option has a description. Below the 'RTL Project' option, there are two checkboxes: 'Do not specify sources at this time' and 'Project is an extensible Vitis platform'. At the bottom, there are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

소스코드와 design constraint는 프로젝트를 생성한 다음에 추가하겠습니다. 일단 그냥 넘어갑니다.



여기에서는 사용할 디바이스를 선택합니다. 상단의 **Boards** 탭으로 넘어가서 **Vendor** 중에 **digilentinc.com**을 선택합니다. 그리고 Nexys A7-100T를 선택합니다. 보드 파일이 아직 설치되어 있지 않는 경우에는  버튼을 눌러 설치하면 됩니다.

New Project

Default Part
Choose a default Xilinx part or board for your project.

Parts | **Boards**

To fetch the latest available boards from git repository, click on 'Refresh' button. [Dismiss](#)

[Reset All Filters](#)

Vendor: Name: Board Rev:

Search:

Display Name	Preview	Status	Vendor	File Version	Part	I/O Pin Count	Board Rev	Available
Nexys A7-100T		⊖	digilentinc.com	1.2	xc7a100tcsg324-1	324	D.0	210
Nexys A7-50T		⬇	digilentinc.com	1.2				
Nexys Video		⬇	digilentinc.com	1.2				
USB104 A7		⬇	digilentinc.com	1.3				

[Refresh](#) Catalog was last updated on 03/06/2023 12:14:31 AM

[? < Back](#) [Next >](#) [Finish](#) [Cancel](#)

만약 찾는 보드가 리스트에 뜨지 않는다면 하단의 **Refresh** 버튼을 눌러 리스트를 다시 받아온 뒤에 다시 시도합니다.

[Refresh](#) Catalog was last updated on 03/06/2023 12:14:31 AM

Project Summary를 통해 설정이 올바르게 됐는지 확인하고 **Finish**를 눌러 프로젝트를 생성합니다.

New Project

VIVADO ML Editions

New Project Summary

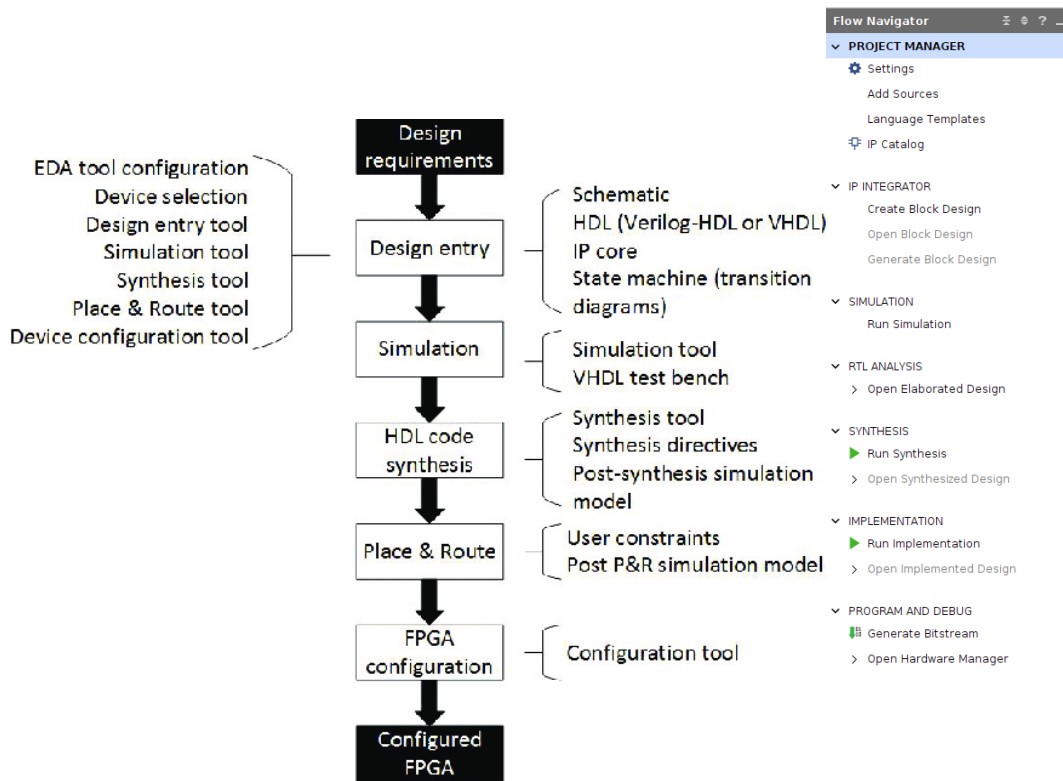
- A new RTL project named 'MyBlink' will be created.
- No source files or directories will be added. Use Add Sources to add them later.
- No constraints files will be added. Use Add Sources to add them later.
- The default part and product family for the new project:
Default Board: Nexys A7-100T
Default Part: xc7a100tcsg324-1
Family: Artix-7
Package: csg324
Speed Grade: -1

XILINX To create the project, click Finish

[? < Back](#) [Next >](#) [Finish](#) [Cancel](#)

Flow Navigator

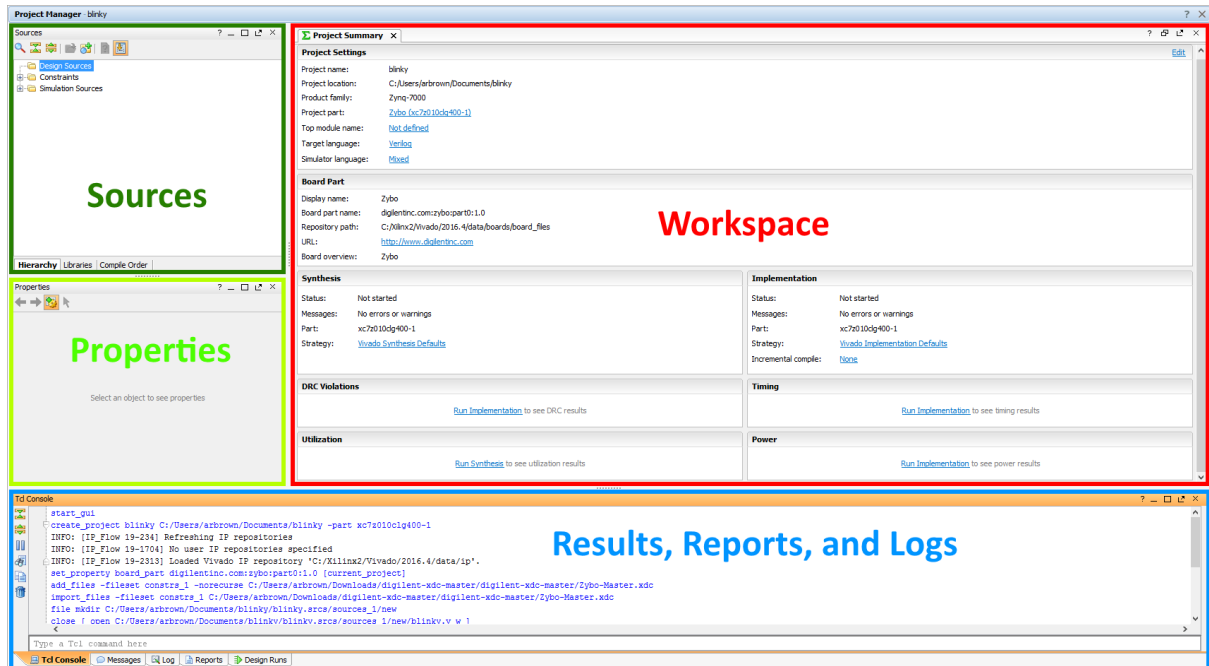
Vivado IDE 좌측에 붙어있는 패널을 Flow Navigator라고 합니다.



위 그림은 개략적인 FPGA design flow와(좌) Vivado의 Flow Navigator를 나타냅니다. 소스코드를 추가하고 편집, 관리하는 등 Design entry부터 Synthesis, Place & Route(Implementation), FPGA configuration(Program and Debugs)까지 Flow Navigator는 FPGA design flow 순서대로 Vivado에서 제공하는 기능들을 배치하여 사용자들이 손쉽게 직관적으로 각 step을 오갈 수 있도록 합니다.

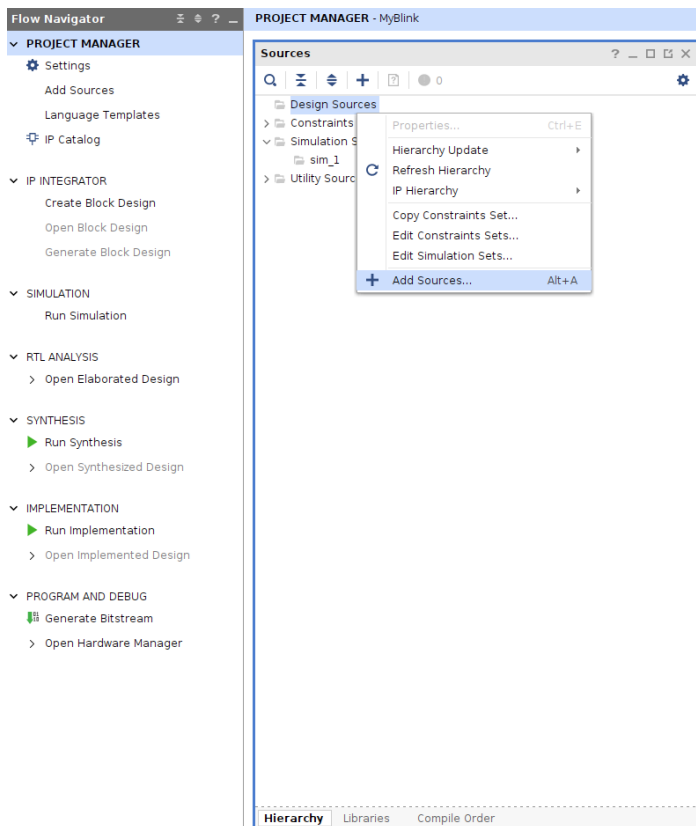
Project Manager의 UI는 다음 그림과 같이 구성되어 있습니다. Visual Studio나 Eclipse 등의 여느 IDE와 비슷한 구조로 이뤄져 있습니다. 자세한 내용은 다음 링크에 있으니 참고해 보시면 좋을 것 같습니다.

<https://digilent.com/reference/programmable-logic/guides/getting-started-with-vivado>



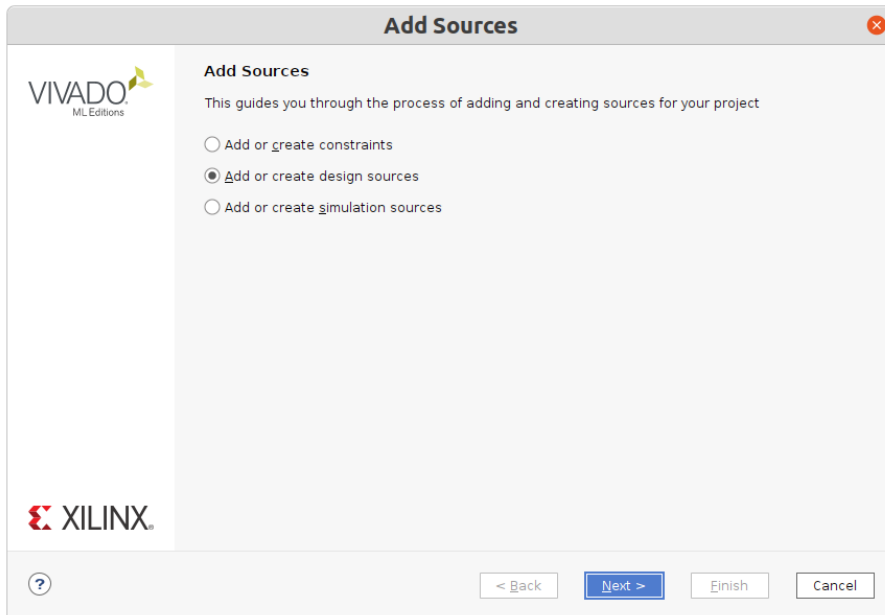
소스코드 생성

Verilog 모듈 하나를 새로 생성해보겠습니다. **Design Sources**에 마우스 오른쪽 버튼을 눌러 **Add Sources...**를 선택합니다.

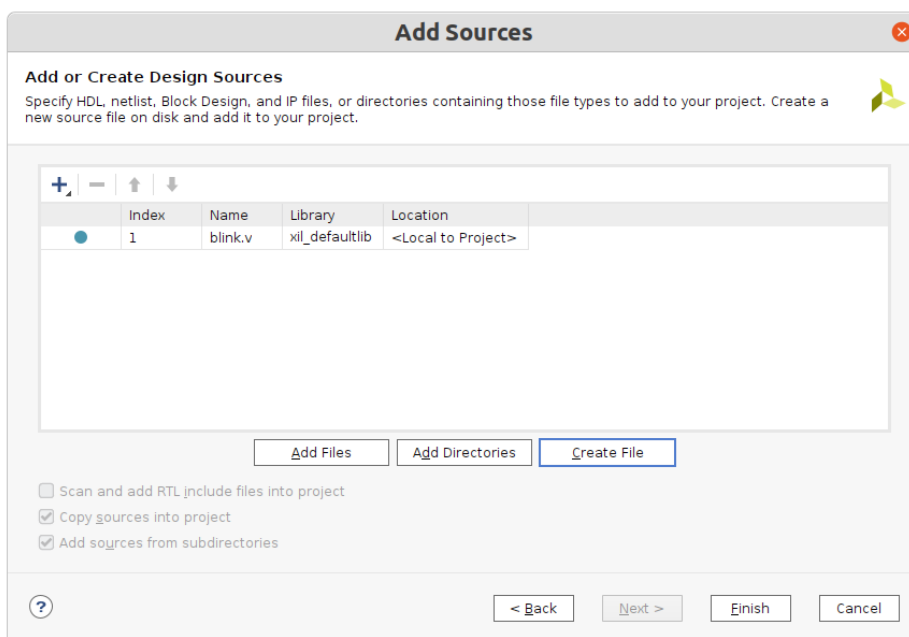


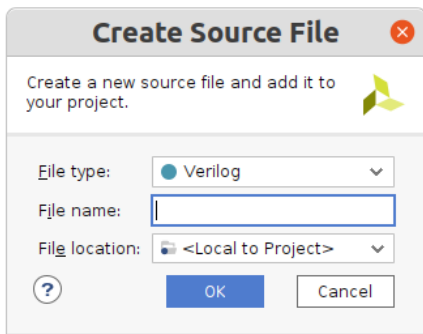
Verilog나 VHDL로 작성된 새로운 하드웨어 모듈 소스코드를 추가하려면 **Design source**를 선택

해야 합니다. Constraint는 우리가 작성한 모듈의 입출력 포트와 실제 디바이스 간의 핀 매핑, 클럭 등 각종 정보를 기술하는 파일을 의미합니다. Constraint는 .v나 .vhd가 아니라 .xdc라는 별도의 확장자를 갖습니다. Simulation source는 시뮬레이션, 즉 테스트벤치와 관련된 코드를 의미합니다. 테스트벤치는 design source와 마찬가지로 Verilog나 VHDL로 작성됩니다.



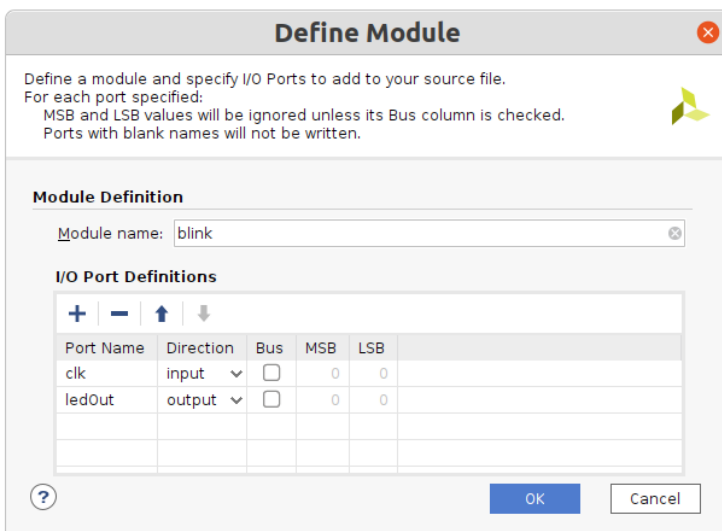
코드를 새로 작성할 것이기 때문에 Add Files 대신 **Create File**을 선택합니다. 기존에 작성된 코드를 프로젝트로 불러올 경우에는 Add Files를 선택하면 됩니다. 일반적으로 모듈의 이름을 소스 파일 이름으로 짓습니다.





입출력 포트를 정의합니다. UI를 이용하는 대신 소스코드를 편집하여 직접 포트를 정의할 수도 있습니다. UI를 이용하면 포트 선언을 자동으로 생성해줍니다.

모듈 blink는 클럭 신호인 clk를 입력 포트, LED의 on/off를 제어하는 ledOut를 출력 포트 갖습니다. 만약 입출력 포트가 1비트보다 큰 버스인 경우(즉, 비트 벡터), **Bus**를 체크하고 **MSB**와 **LSB**에 비트 길이 정보를 입력합니다. 예를 들어, ledOut이 8비트 신호라고 가정한다면, MSB에 7, LSB에는 0을 입력하면 됩니다.



소스코드 작성

소스코드를 생성하면 다음과 같이 앞서 입력한 포트 정보를 바탕으로 포트와 모듈 선언이 자동으로 생성됩니다. 자동 생성된 포트 선언은 Verilog 2001* 스타일로 작성되어 있는 것을 확인할 수 있습니다.

```
module blink(input clk, output ledOut);
endmodule
```

* 포트 선언과 관련하여 Verilog 1995와 2001 표준의 차이점은 다음 링크를 참고해주세요.

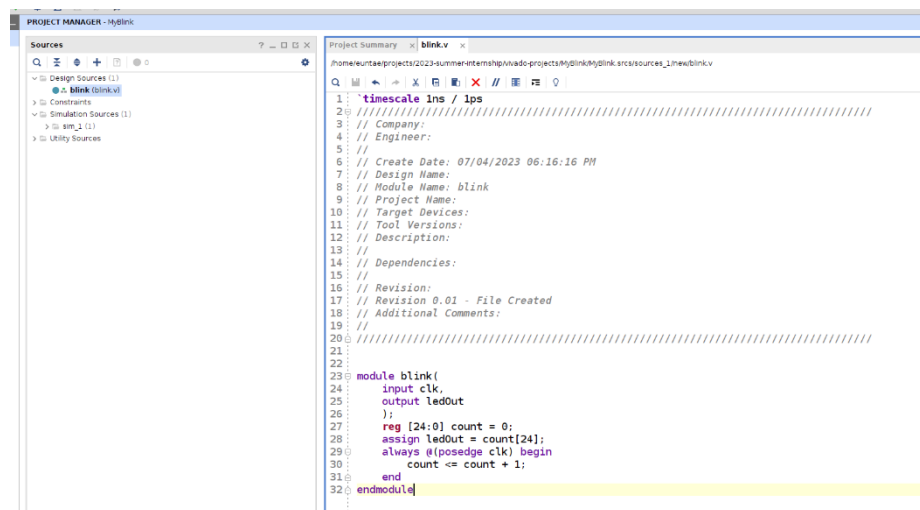
<https://www.chipverify.com/verilog/verilog-ports>

이제 모듈 blink의 내용을 채워주겠습니다. LED를 일정 주기로 깜빡이게 하기 위해 이번 예제에서는 카운터를 사용했습니다. 카운트 값을 저장하기 위한 25비트 레지스터인 count는 매 클럭 사이 클마다 1씩 증가하며, LED on/off 신호인 ledOut은 count의 MSB와 연결됩니다. 즉, blink 모듈은 count의 최상위 비트가 1일 때는 on을, 0일 때는 0을 출력합니다.

[blink.v]

```
`timescale 1ns / 1ps

module blink(
    input clk,
    output ledOut
);
    reg [24:0] count = 0;
    assign ledOut = count[24];
    always @(posedge clk) begin
        count <= count + 1;
    end
endmodule
```



Design Constraint 작성

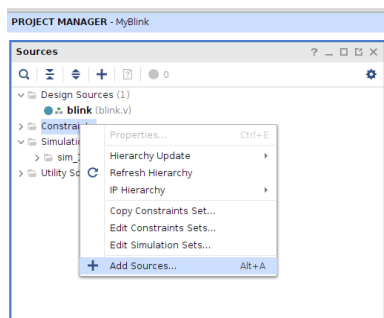
이제 ledOut을 LED[0]와 연결할 차례입니다. 그리고 클럭 신호인 clk를 실제 공급되는 클럭 신호와 대응시키는 작업이 필요합니다. 설계한 회로를 실제 FPGA에 올려 동작 시키려면 핀 매핑이나 클럭 사양 등 하드웨어 정보를 기술해야 합니다. 이런 정보들을 design constraint라고 합니다. Design constraint는 HDL 코드와 달리 .xdc(Xilinx Design Constraint, XDC)라는 별도의 확장자를 갖는(참고로 Vivado의 전신인 ISE 시절에는 .ucf라는 포맷을 사용했습니다) 일련의 Tcl 코드로 구성되어 있습니다.

Digilent는 자사 보드의 master XDC 파일을 배포하고 있습니다. master XDC 파일에는 해당 보드의 모든 매핑과 클럭 신호 등을 미리 기술해 두었기 때문에 필요한 핀 정보만 주석을 지워서 포

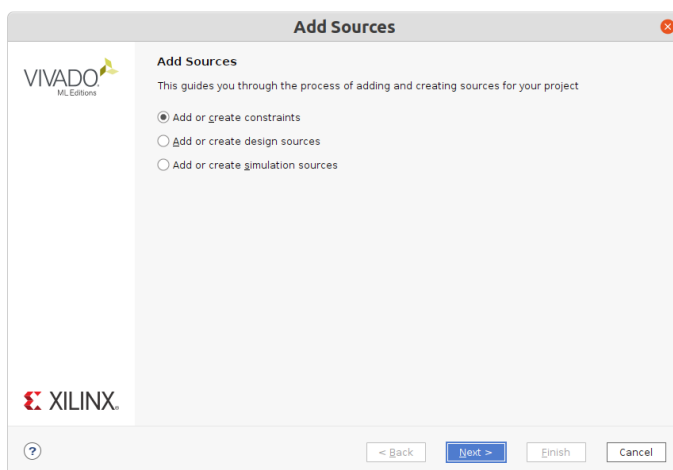
트 이름에 맞게 수정하면 됩니다. 파일은 다음 링크에서 받을 수 있습니다.

<https://github.com/Digilent/digilent-xdc/archive/master.zip>

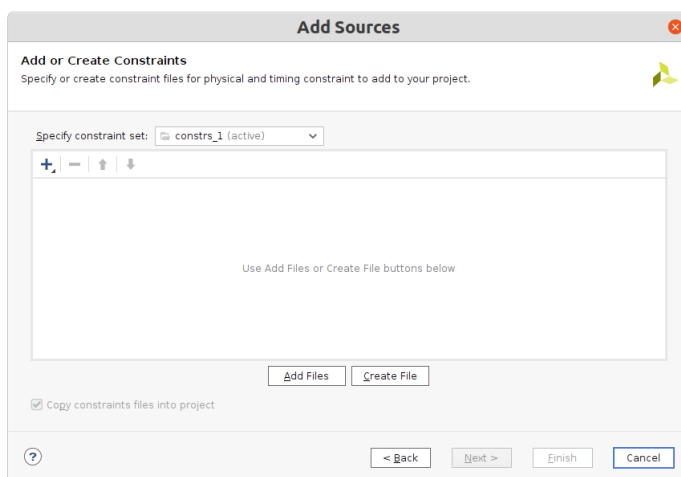
현재 실습에서는 Nexys A7-100T를 사용하기 때문에 Nexys-A7-100T-Master.xdc를 사용하겠습니다. 이제 이 파일을 프로젝트로 불러와야 합니다. Project Manager에서 **Add Sources...**를 선택하여 소스 파일을 추가합니다.



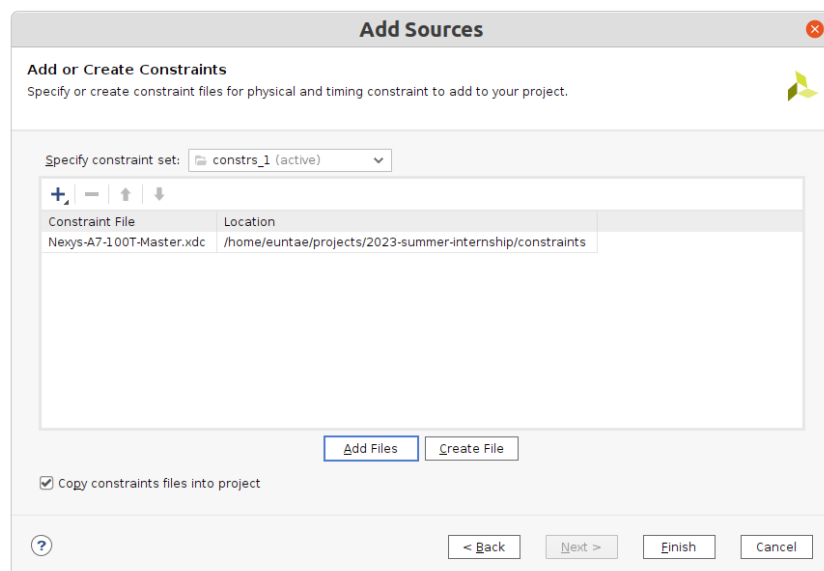
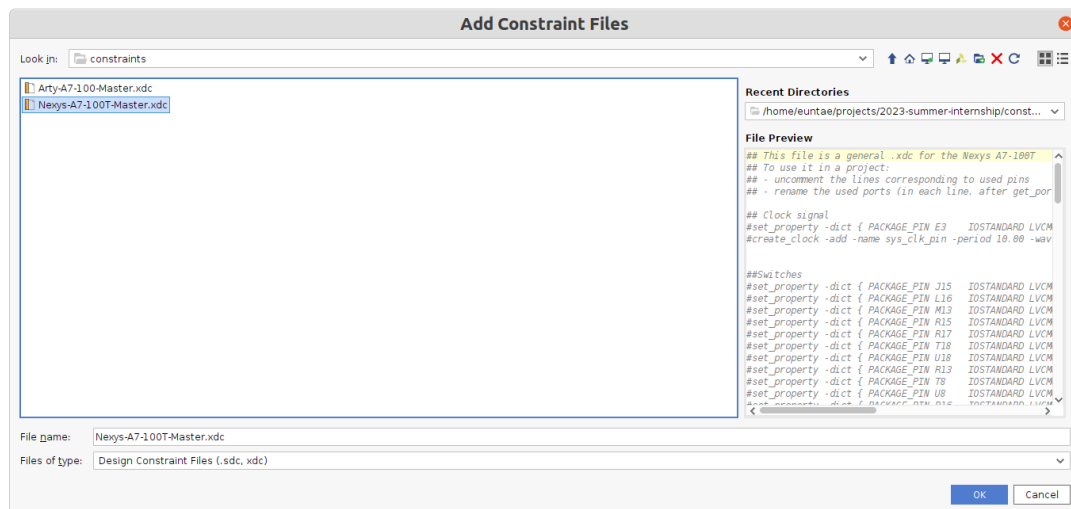
이번에는 Design Source가 아니라 Constraint를 선택합니다.



기존에 작성된 master XDC를 불러오는 것이므로 Create File대신 **Add Files**를 선택합니다.



Constraint 파일이 있는 경로를 찾아가 Nexys-A7-100T-Master.xdc 파일을 선택합니다.



다음 그림과 같이 master XDC 파일은 기본적으로 모든 코드가 주석 처리된 상태입니다.

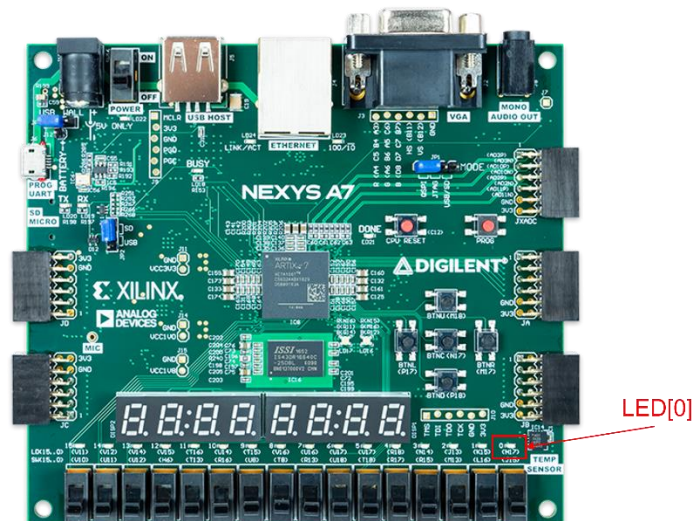
```

Project Summary x Mink.v x Nexys-A7-100T-Master.xdc
/home/tae/projects/2023-summer-internship/vivado/projects/hglink/hglink.srcs/constrs_1/imports/constraints/Nexys-A7-100T-Master.xdc

1: ## This file is a general .xdc for the Nexys A7-100T
2: ## To use it in a project:
3: ## - uncomment the lines corresponding to used pins
4: ## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project
5:
6: ## Clock signal
7: set_property -dict { PACKAGE_PIN E3      IOSTANDARD LVCMOS33 } [get_ports { CLK100MHZ }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
8: create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100MHZ}];
9:
10:
11: ##Switches
12: set_property -dict { PACKAGE_PIN J15      IOSTANDARD LVCMOS33 } [get_ports { SW[0] }]; #IO_L24N_T3_R50_15 Sch=sw[0]
13: set_property -dict { PACKAGE_PIN L16      IOSTANDARD LVCMOS33 } [get_ports { SW[1] }]; #IO_L3N_T0_D05_EWCLK_14 Sch=sw[1]
14: set_property -dict { PACKAGE_PIN M13      IOSTANDARD LVCMOS33 } [get_ports { SW[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
15: set_property -dict { PACKAGE_PIN R15      IOSTANDARD LVCMOS33 } [get_ports { SW[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
16: set_property -dict { PACKAGE_PIN R17      IOSTANDARD LVCMOS33 } [get_ports { SW[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
17: set_property -dict { PACKAGE_PIN T18      IOSTANDARD LVCMOS33 } [get_ports { SW[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
18: set_property -dict { PACKAGE_PIN U18      IOSTANDARD LVCMOS33 } [get_ports { SW[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
19: set_property -dict { PACKAGE_PIN R13      IOSTANDARD LVCMOS33 } [get_ports { SW[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
20: set_property -dict { PACKAGE_PIN T8       IOSTANDARD LVCMOS18 } [get_ports { SW[8] }]; #IO_L24N_T3_34 Sch=sw[8]
21: set_property -dict { PACKAGE_PIN U8       IOSTANDARD LVCMOS18 } [get_ports { SW[9] }]; #IO_25_34 Sch=sw[9]
22: set_property -dict { PACKAGE_PIN R16      IOSTANDARD LVCMOS33 } [get_ports { SW[10] }]; #IO_L15P_T2_D05_RDWR_B_14 Sch=sw[10]
23: set_property -dict { PACKAGE_PIN T13      IOSTANDARD LVCMOS33 } [get_ports { SW[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
24: set_property -dict { PACKAGE_PIN H6       IOSTANDARD LVCMOS33 } [get_ports { SW[12] }]; #IO_L24P_T3_35 Sch=sw[12]
25: set_property -dict { PACKAGE_PIN U12      IOSTANDARD LVCMOS33 } [get_ports { SW[13] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
26: set_property -dict { PACKAGE_PIN U11      IOSTANDARD LVCMOS33 } [get_ports { SW[14] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
27: set_property -dict { PACKAGE_PIN V10      IOSTANDARD LVCMOS33 } [get_ports { SW[15] }]; #IO_L21P_T3_D05_14 Sch=sw[15]
28:
29: ## LEDs
30: set_property -dict { PACKAGE_PIN H17      IOSTANDARD LVCMOS33 } [get_ports { LED[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
31: set_property -dict { PACKAGE_PIN K13      IOSTANDARD LVCMOS33 } [get_ports { LED[1] }]; #IO_L24P_T3_R51_15 Sch=led[1]
32: set_property -dict { PACKAGE_PIN J15      IOSTANDARD LVCMOS33 } [get_ports { LED[2] }]; #IO_L7N_T2_A25_15 Sch=led[2]
33: set_property -dict { PACKAGE_PIN N14      IOSTANDARD LVCMOS33 } [get_ports { LED[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
34: set_property -dict { PACKAGE_PIN R18      IOSTANDARD LVCMOS33 } [get_ports { LED[4] }]; #IO_L17P_T1_D09_14 Sch=led[4]
35: set_property -dict { PACKAGE_PIN V17      IOSTANDARD LVCMOS33 } [get_ports { LED[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
36: set_property -dict { PACKAGE_PIN U17      IOSTANDARD LVCMOS33 } [get_ports { LED[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
37: set_property -dict { PACKAGE_PIN U16      IOSTANDARD LVCMOS33 } [get_ports { LED[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
38: set_property -dict { PACKAGE_PIN V16      IOSTANDARD LVCMOS33 } [get_ports { LED[8] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
39: set_property -dict { PACKAGE_PIN T15      IOSTANDARD LVCMOS33 } [get_ports { LED[9] }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
40: set_property -dict { PACKAGE_PIN U14      IOSTANDARD LVCMOS33 } [get_ports { LED[10] }]; #IO_L22P_T3_A05_D21_14 Sch=led[10]
41: set_property -dict { PACKAGE_PIN T16      IOSTANDARD LVCMOS33 } [get_ports { LED[11] }]; #IO_L15N_T2_D05_D0UT_CS0_B_14 Sch=led[11]
42: set_property -dict { PACKAGE_PIN V15      IOSTANDARD LVCMOS33 } [get_ports { LED[12] }]; #IO_L16P_T2_CS1_B_14 Sch=led[12]
43: set_property -dict { PACKAGE_PIN V14      IOSTANDARD LVCMOS33 } [get_ports { LED[13] }]; #IO_L22N_T3_A04_D20_14 Sch=led[13]
44: set_property -dict { PACKAGE_PIN V12      IOSTANDARD LVCMOS33 } [get_ports { LED[14] }]; #IO_L20N_T3_A07_D23_14 Sch=led[14]
45: set_property -dict { PACKAGE_PIN V11      IOSTANDARD LVCMOS33 } [get_ports { LED[15] }]; #IO_L21N_T3_D05_A06_D22_14 Sch=led[15]
46:

```

이제 이 코드를 수정하여 ledOut과 LED[0]를 매핑 시켜보겠습니다.



30행 set_property 문의 주석(#)을 해제하고, get_ports 다음에 있는 LED[0]를 ledOut으로 수정합니다. get_ports 명령의 입력에는 HDL 코드에 정의한 포트 이름이 들어갑니다.

```

28:
29: ## LEDs
30: set_property -dict { PACKAGE_PIN H17      IOSTANDARD LVCMOS33 } [get_ports { ledOut }]; #IO_L18P_T2_A24_15 Sch=led[0]
31: set_property -dict { PACKAGE_PIN K13      IOSTANDARD LVCMOS33 } [get_ports { LED[1] }]; #IO_L24P_T3_R51_15 Sch=led[1]

```

그 다음에는 클럭 신호인 clk입니다. 클럭 신호는 7행과 8행에 있습니다. 마찬가지로 주석을 해제하고 get_ports의 인자를 CLK100MHZ에서 clk로 수정합니다.

```

5:
6: ## Clock signal
7: set_property -dict { PACKAGE_PIN E3      IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
8: create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {clk}];
9:

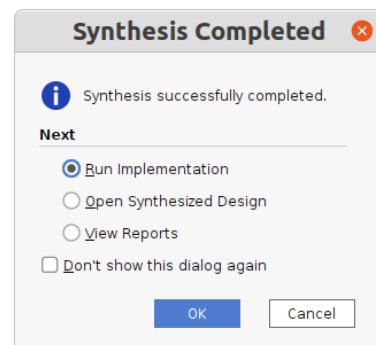
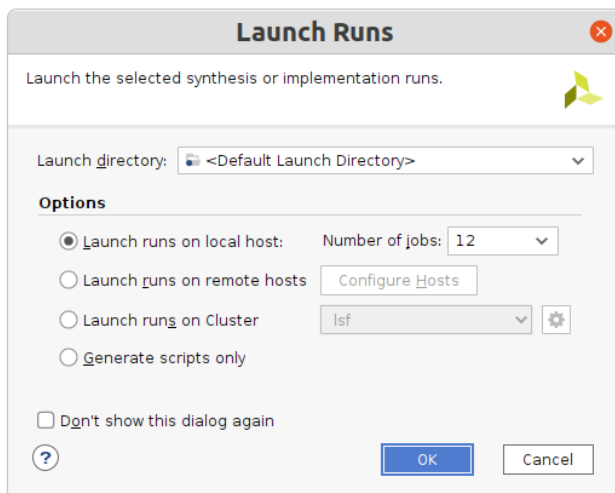
```

Synthesis

이제 작성된 소스 파일을 논리 합성하겠습니다. Flow Navigator의 **Run Synthesis**를 선택합니다.

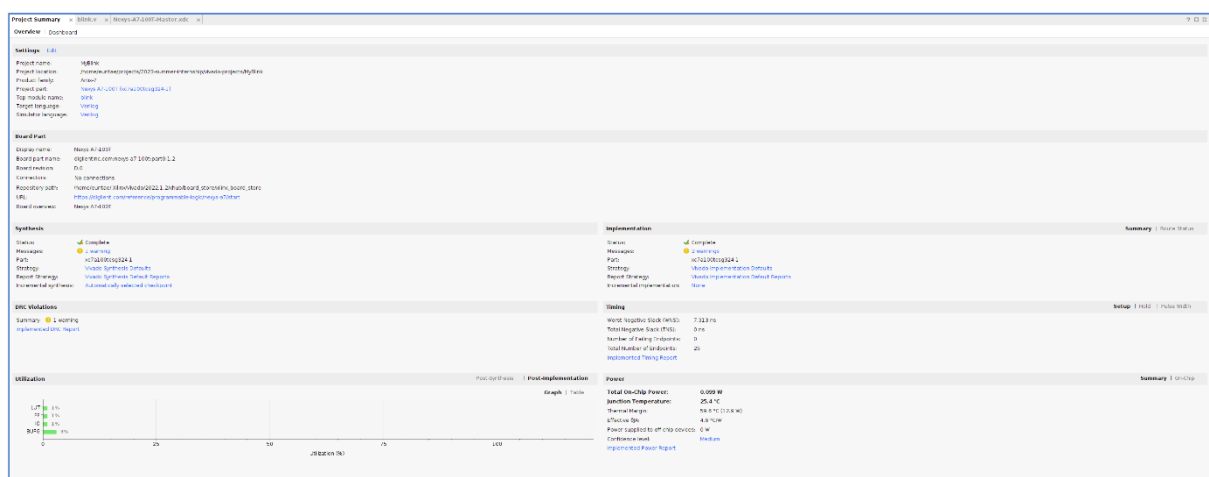
SYNTHESIS

- ▶ Run Synthesis
- > Open Synthesized Design

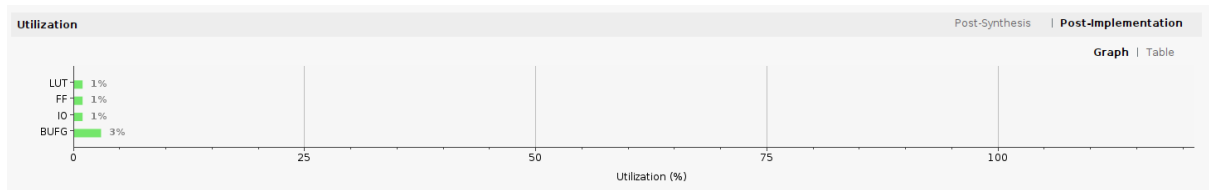


합성과 구현은 상당히 시간이 걸리는 작업입니다. 컴퓨터 코어 수에 맞게 적절한 Number of jobs를 입력합니다. 합성이 완료되고 문제가 없으면 바로 **Run Implementation**으로 넘어가도록 합니다.

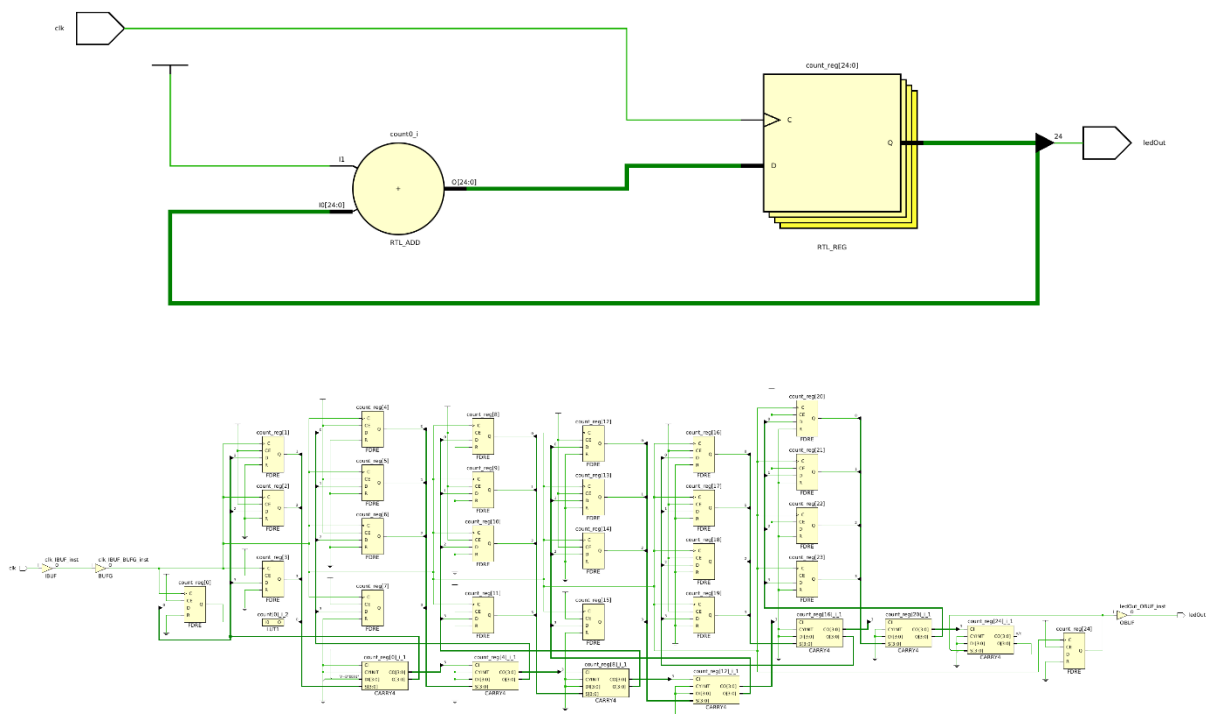
Run Implementation 대신 **Open Synthesized Design**을 통해 Post Synthesis 단계에서의 정보들을 확인하거나 관련 작업을 수행할 수 있습니다. 대표적으로 다음과 같이 Project Summary를 확인하여 합성 결과로부터 도출된 각종 리포트 정보를 확인할 수 있습니다.



하단의 Utilization은 설계한 하드웨어가 FPGA의 자원을 얼마나 사용하고 있는지를 보여줍니다. FPGA는 ASIC과 다르게 미리 준비된 한정된 양의 하드웨어 자원을 조합하여 우리가 설계한 회로를 구현한다는 점을 기억해야 합니다.



논리 합성의 결과로 gate-level netlist가 생성됩니다. RTL 수준의 추상적인 회로 디자인은 물리적인 하드웨어로 구현되어야 합니다. 합성 단계에서 RTL 수준의 디자인은 standard cell library를 이용하여 실제 논리 회로로 매핑됩니다. 다음은 RTL 수준에서의 회로도(schematic)와 합성 이후의 회로도를 나타냅니다.



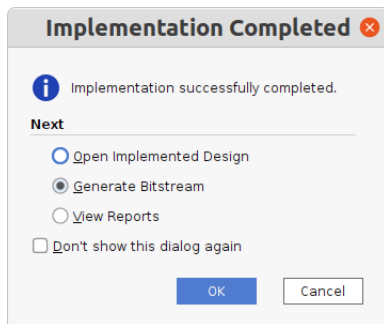
시각화된 회로도는 Open Synthesized Design의 **Schematic**을 통해 확인할 수 있습니다. 마찬가지로 RTL 회로도는 Open Elaborated Design의 Schematic을 통해 확인할 수 있습니다.

▼ Open Synthesized Design

- Constraints Wizard
- Edit Timing Constraints
- 🔧 Set Up Debug
- 🕒 Report Timing Summary
 - Report Clock Networks
 - Report Clock Interaction
- 📄 Report Methodology
 - Report DRC
 - Report Noise
 - Report Utilization
- 🔌 Report Power
- 🔍 Schematic

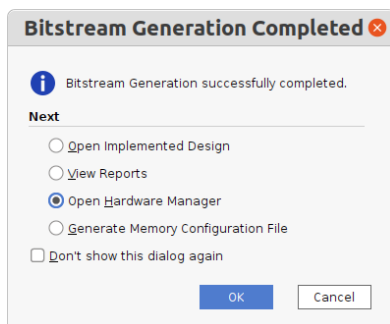
Implementation

다음은 Implementation 단계입니다. Implementation에서는 하드웨어 셀들의 물리적인 배치와 배선(Place & Route, PnR)을 수행합니다. Synthesis와 마찬가지로 Implementation이 성공적으로 완료되면 다음 단계인 **Generate Bitstream**을 선택할 수 있습니다. 실제로 FPGA에 올라가는 것이 이 Bitstream입니다.

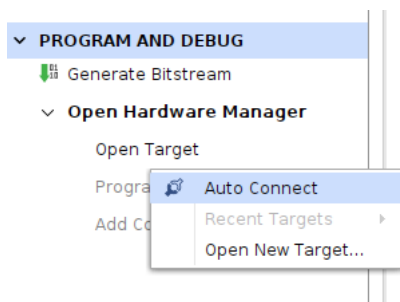


Program and Debug

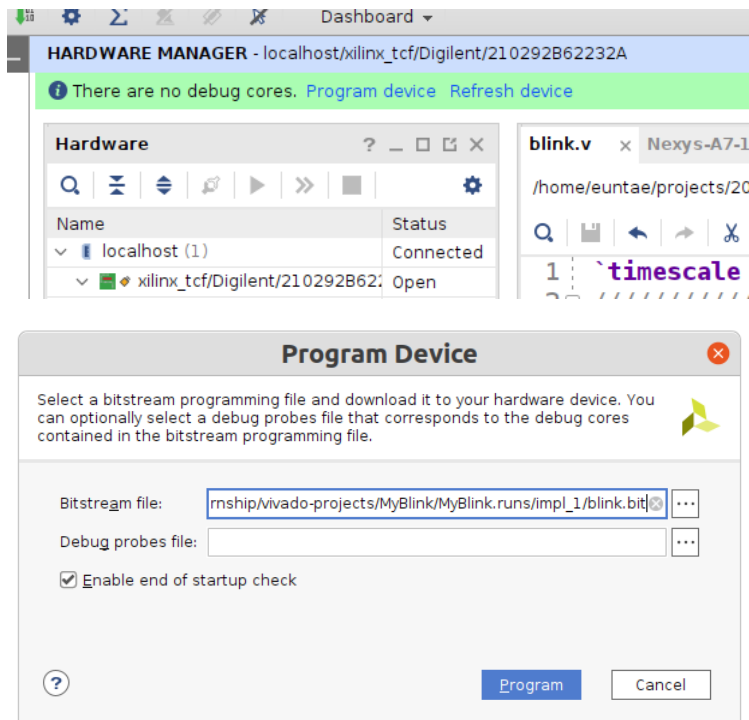
Implementation과 마찬가지로 bitstream 생성이 성공적으로 완료되면 이제 FPGA에 올릴 수 있습니다(이를 configure 또는 program 한다 등으로 표현합니다). **Open Hardware Manager**를 선택합니다.



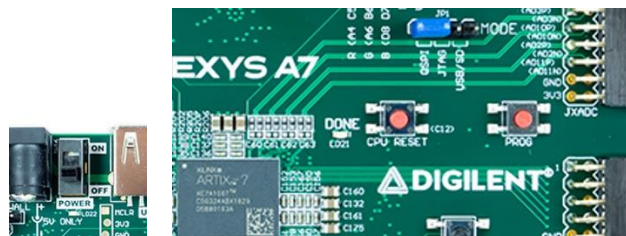
동봉된 USB 케이블을 이용해서 보드를 PC와 연결했다면(그리고 FPGA 보드의 전원이 들어와 있는지 확인하세요) **Open Target**을 클릭하고 **Auto Connect**를 선택합니다.



보드가 정상적으로 인식됐다면 보드에 bitstream을 올릴지(program할지) 여부를 물어봅니다. **Program device**를 선택합니다.



- 프로그램이 완료된 이후 LED가 깜빡이는지 확인해봅시다.
- 전원 스위치(POWER)를 이용하여 보드 전원을 껐다 키거나 PROG 버튼을 누르면 어떻게 되는지 확인해봅시다. 이후에도 LED가 깜빡일까요?



사실 결론부터 말씀드리자면, 전원이 꺼지면 FPGA에 올라간 bitstream은 날아갑니다. 이는 FPGA를 구성하고 있는 LUT(Lookup Table)가 SRAM 기반이기 때문입니다. 따라서 bitstream 정보는 휘발성입니다. 그렇다면 우리가 설계한 회로를 사용하기 위해 전원이 차단될 때마다 매번 다시 재프로그램 해줘야 할까요? 그렇지 않습니다. FPGA 보드에는 플래시 메모리가 장착되어 있어서 여기에 bitstream을 기록해두고 부팅 시 bitstream을 프로그램 해줄 수 있습니다. 이와 관련된 내용은 다음 실습 때 다시 소개해드리도록 하겠습니다.