

# DIGCOM-XA1.0 사용 자료

주식회사 인트모션

# 머 리 말

DIGCOM-XA1.0은 Artix7 FPGA를 사용한 디지털 논리 실습 키트이다. 본 문서는 DIGCOM-XA1.0을 사용하기 위해서 필요한 Vivado Design Suite 설계 소프트웨어를 설치하는 방법과 Vivado를 사용하여 논리회로를 설계하는 방법 및 설계된 논리회로를 DIGCOM-XA1.0에서 실행하는 방법을 소개한다.

DIGCOM-XA1.0에서 실행되는 예제는 한빛아카데미 출판사의 교재 “디지털 시스템 설계 및 실습, with VHDL & Verilog”에서 모두 설명하고 있다. 그러나 교재에서는 ISE Design Suite를 사용하여 설명을 했으며, Vivado Design Suite에서는 schematic 설계를 지원하지 않기 때문에 본 문서에서는 교재에서 schematic으로 설계된 부분을 삭제했거나 다른 방법으로 설계하는 방법을 설명하였다. 따라서 DIGCOM-XA1.0에서 실행되는 예제는 “디지털 시스템 설계 및 실습, with VHDL & Verilog”을 참고하며, 본 문서는 교재와 다른 내용만을 설명한다.

# 목 차

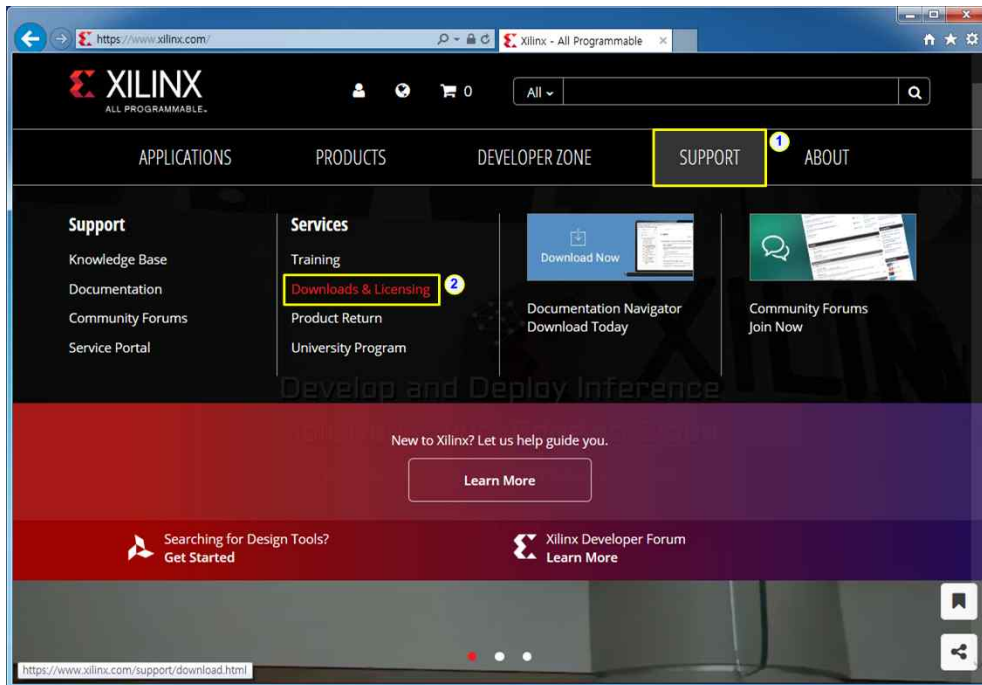
- I. Vivado Design Suite 설치
  - Vivado 2016. 4 설치
  - Vivado 2017. 2 설치
- II. Vivado 2016. 4 사용
- III. DIGCOM-XA1.0 실습 예제
- IV. DIGCOM-XA1.0 핀 할당

# I. Vivado Design Suite 설치

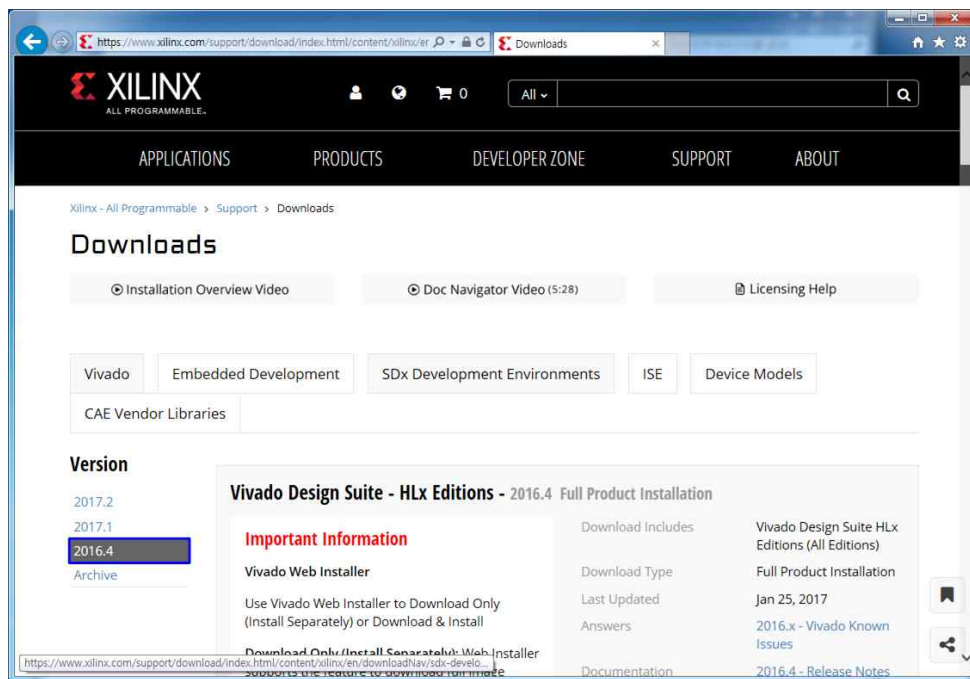
## - Vivado 2016. 4 설치

DIGCOM-XA1.0에서 실행되는 예제는 모두 Vivado 2016. 4 version에서 테스트 되었다. 따라서 최신의 version을 사용하는 것 보다는 Vivado 2016. 4를 설치하여 실행하는 것을 권장한다 Vivado 2016. 4는 윈도우 7과 윈도우 10에서 모두 실행이 된다.

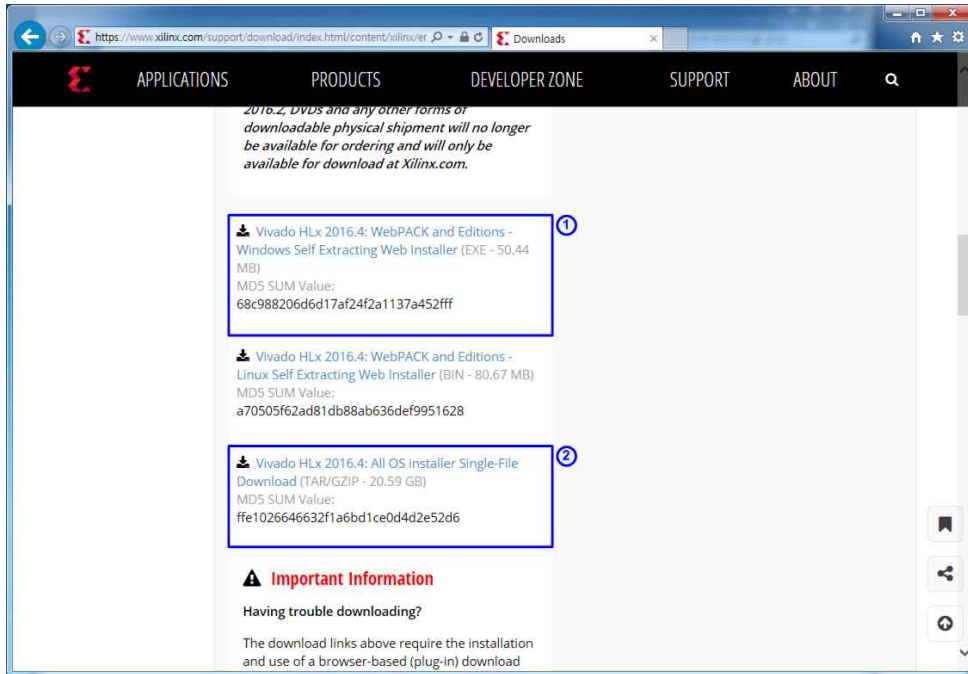
1. <http://www.xilinx.com>에 접속하여 [SUPPORT]⇒[Downloads & Licensing]을 클릭한다.



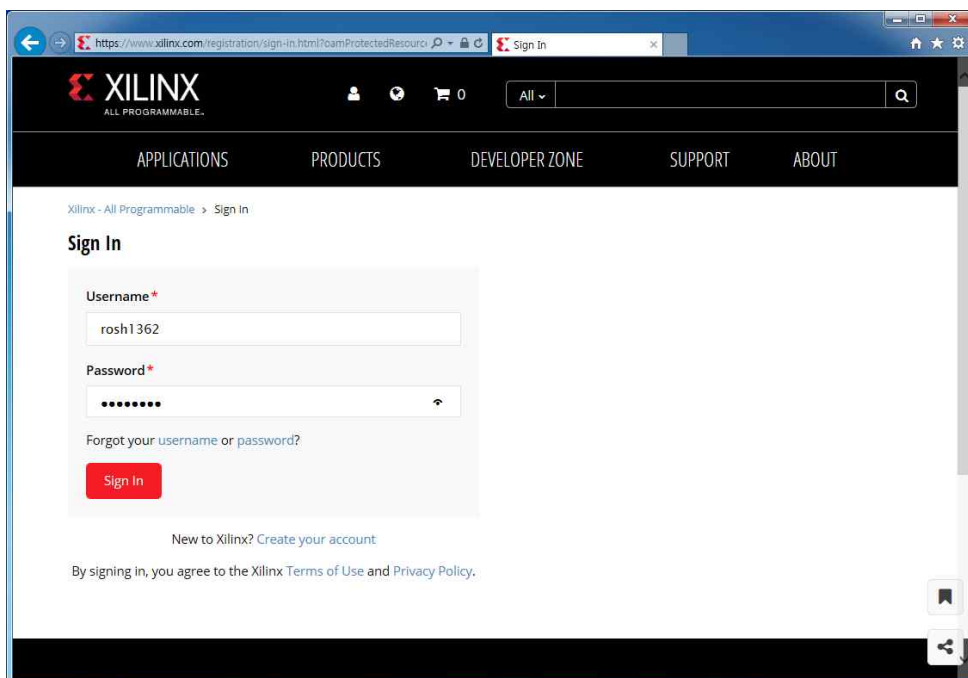
2. Download 화면에 설치할 Vivado version이 표시되며, [2016. 4]를 선택한다.



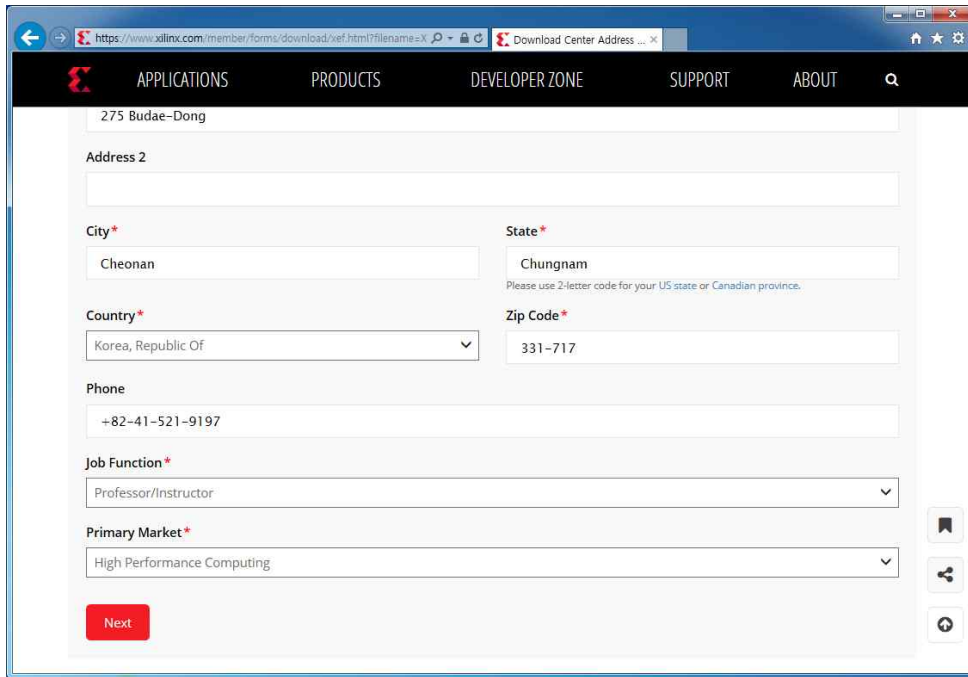
3. 같은 화면 아래에서 그림에서와 같이 ①을 선택하면 Vivado 파일을 다운로드하면서 설치를 하며, ②를 선택하면 전체 파일을 다운로드 한 후에 설치를 시작한다. ①을 선택하면 네트워크 연결 상태에 따라서 오히려 설치시간이 오래 걸릴 수 있기 때문에 ② 방법을 추천한다.



4. 로그인 화면이 나오면 [Username]와 [Password]를 입력하여 로그인을 하며, 회원이 아니면 회원으로 가입을 한 후 로그인을 한다.



5. 사용자에게 대한 정보를 확인하는 화면이 나오며, 화면 아래에 [Next]를 클릭하여 다운로드를 시작한다.

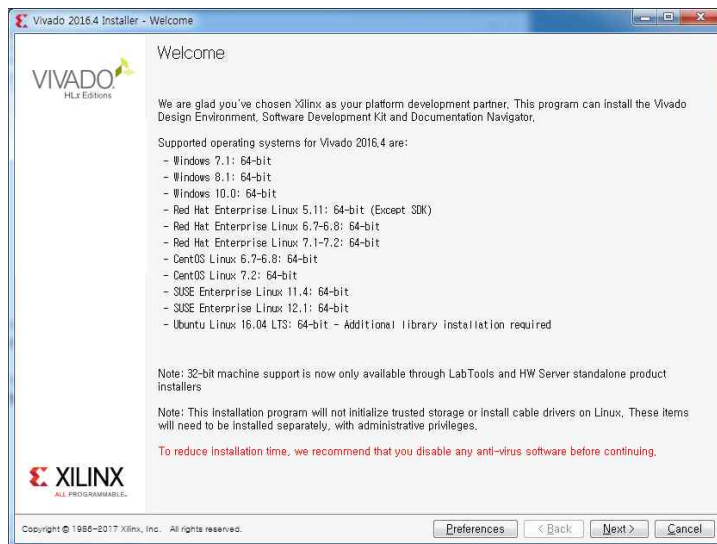


The screenshot shows a web browser window with the URL <https://www.xilinx.com/member/forms/download/xfef.html?filename=X>. The page has a navigation bar with 'APPLICATIONS', 'PRODUCTS', 'DEVELOPER ZONE', 'SUPPORT', and 'ABOUT'. The main content area is a registration form with the following fields:

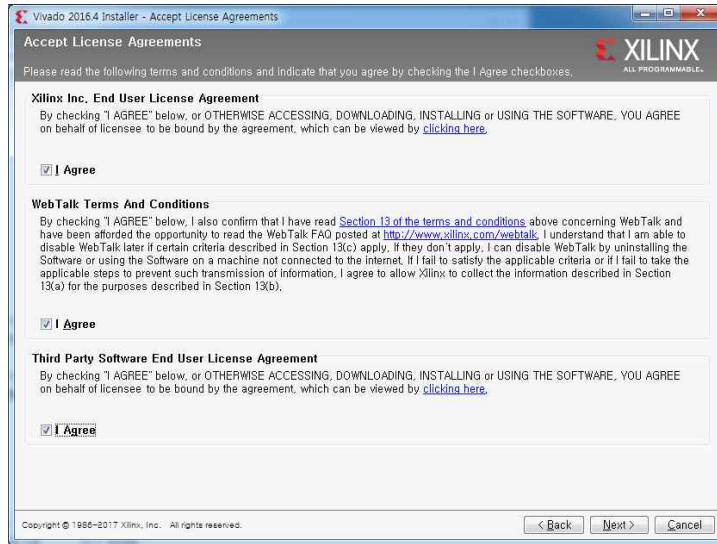
- Address 1: 275 Budae-Dong
- Address 2: (empty)
- City\*: Cheonan
- State\*: Chungnam (Note: Please use 2-letter code for your US state or Canadian province.)
- Country\*: Korea, Republic Of
- Zip Code\*: 331-717
- Phone: +82-41-521-9197
- Job Function\*: Professor/Instructor
- Primary Market\*: High Performance Computing

A red 'Next' button is located at the bottom left of the form. On the right side, there are icons for bookmark, share, and refresh.

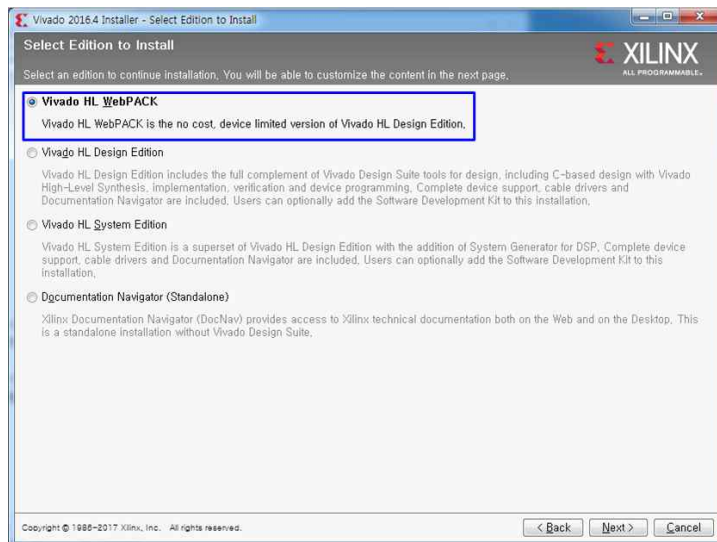
6. 다운로드한 압축파일을 해제하고 [setup]을 클릭하여 설치를 시작하면 아래 그림과 같이 설치 화면이 나오며 [Next]를 클릭하여 설치를 시작한다.



7. 라이선스 등의 동의하는 화면에서 모두 동의를 한 후에 [Next]를 클릭한다.

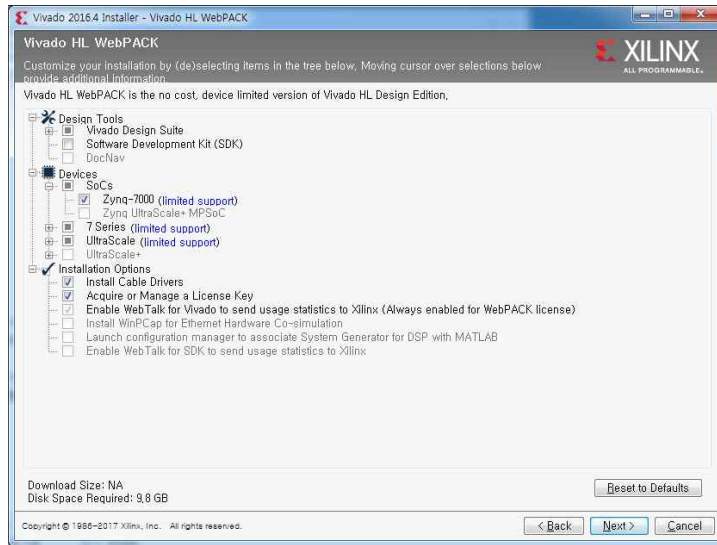


8. 설치할 “edition”을 선택해야 하며, [Vivado HL WebPACK]을 선택하고 [Next]를 클릭한다.

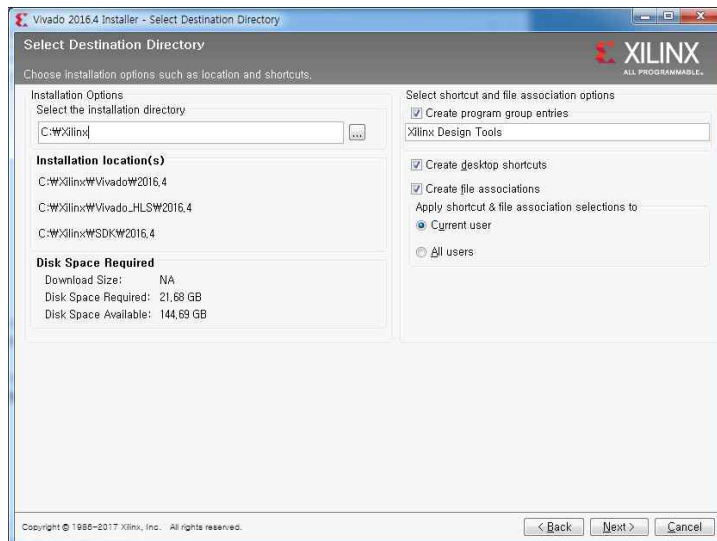




9. 설치할 항목을 선택하는 화면에서 [Next]를 클릭한다. 이 화면에서 설치시간을 줄이기 위해서는 [Device] 항목에서 Artix-7만 남기고 나머지를 모두 해제해도 된다.



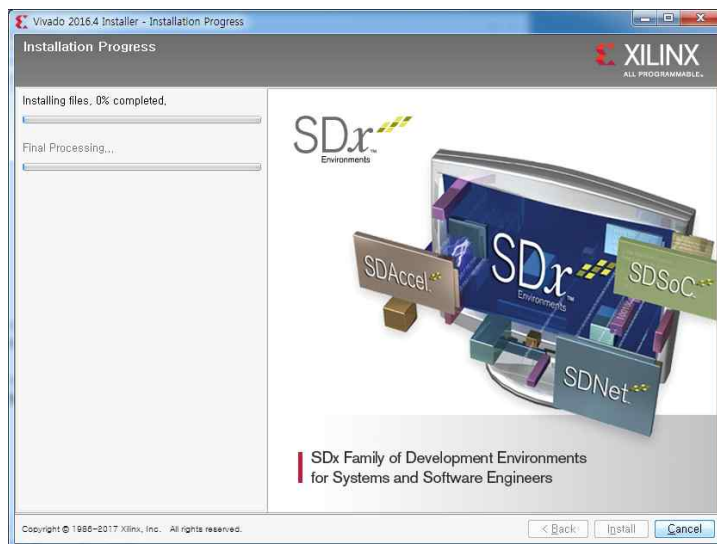
10. 설치할 폴더를 선택하고 [Next]를 클릭한다.



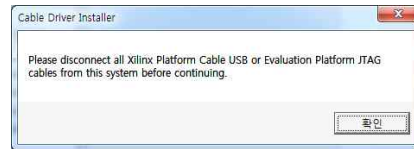
11. 설치에 대한 정보를 확인한 후 [Install]을 클릭하면 설치를 시작한다.



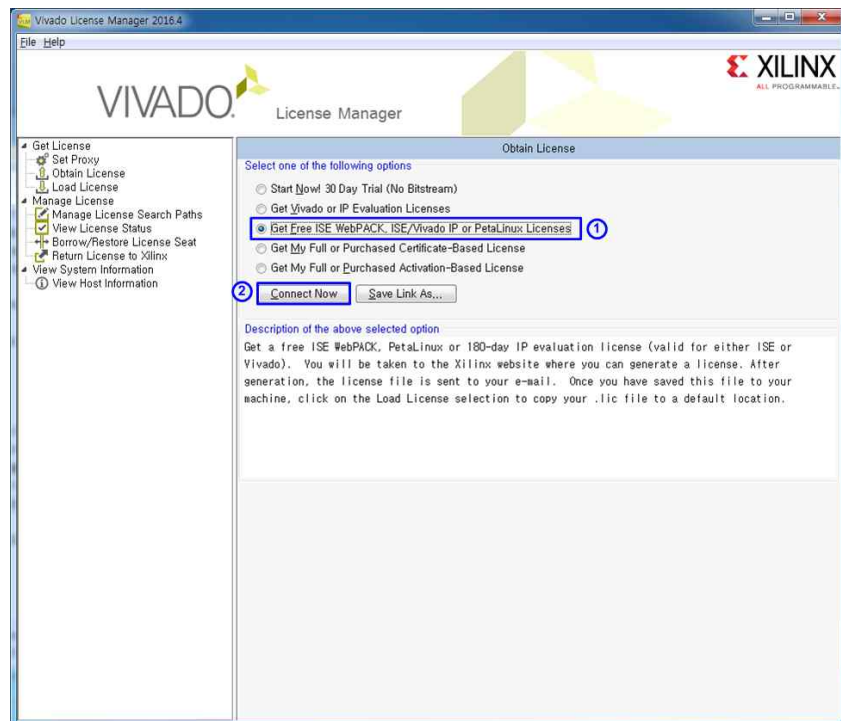
12. 아래 그림과 같이 설치 진행을 보여준다.



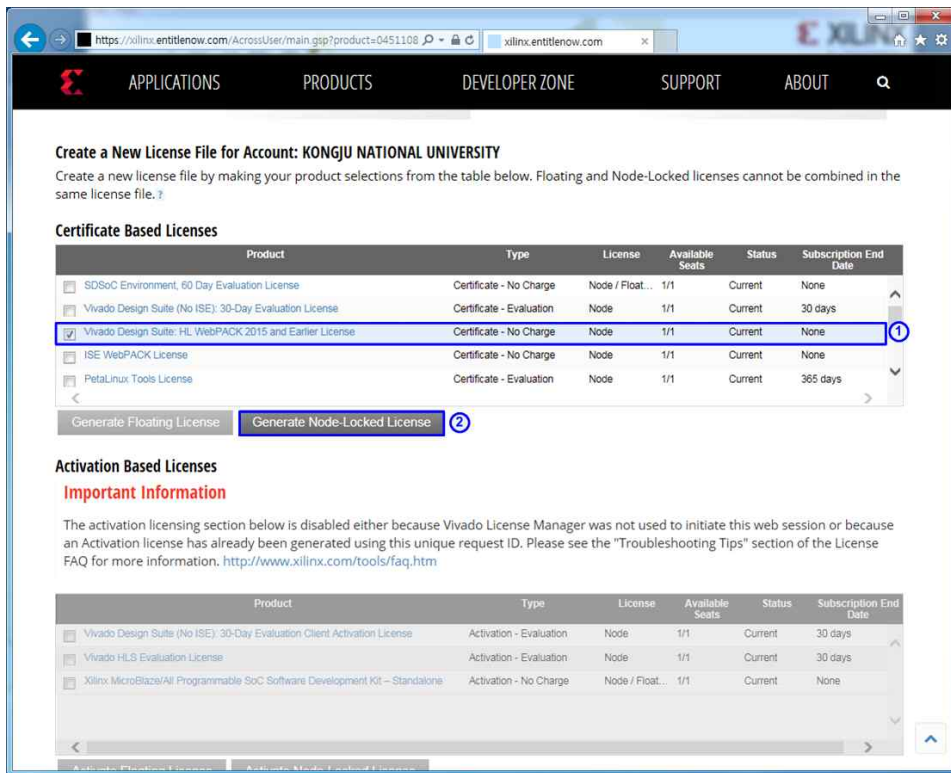
13. 설치 과정에서 아래 그림과 같이 Platform Cable을 제거하라는 메시지가 나오면 [확인]을 클릭하고 설치를 진행한다. 만일 Platform Cable이 연결되어 있으면 연결을 제거한다.



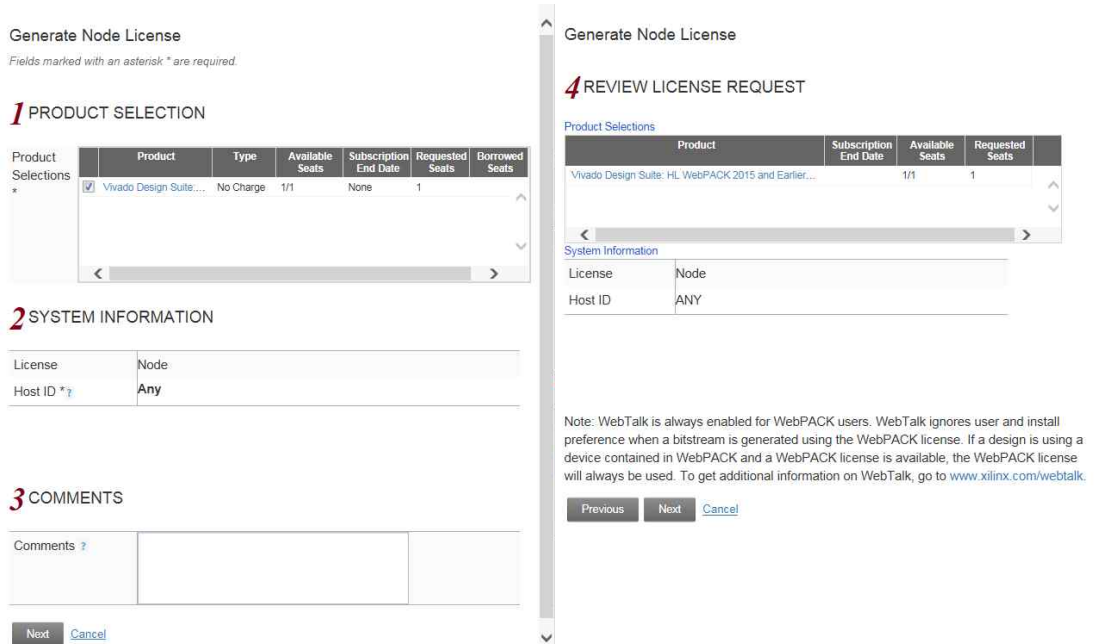
14. 설치를 마치면 아래와 같이 [License Manager] 창이 뜨며, [Get Free ISE WebPACK, ISE/Vivado IP or PetaLinux Licenses]를 선택하고 [Connect Now]를 클릭한다.



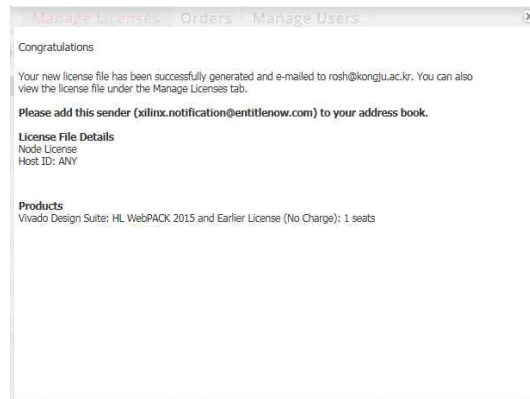
15. 라이선스를 선택하는 화면에서 [Vivado Design Suite : HL WebPACK 2015 and Earlier License]를 선택하고, [Generate Node-Locked License]를 클릭한다.



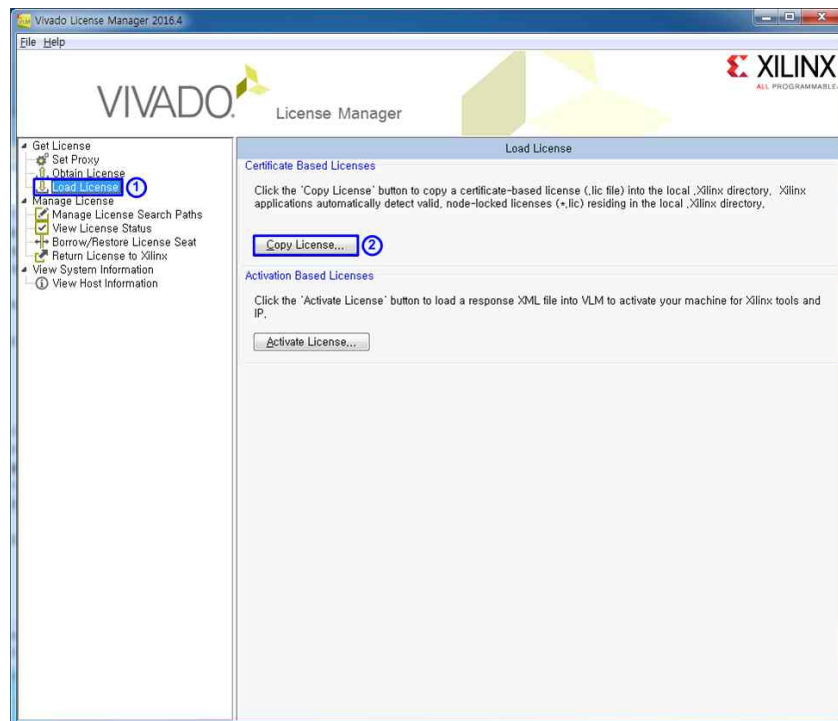
16. 라이선스를 확인하는 화면에서 [Next] ⇒ [Next]를 클릭한다.



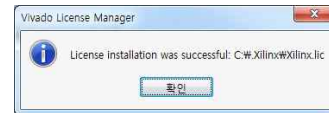
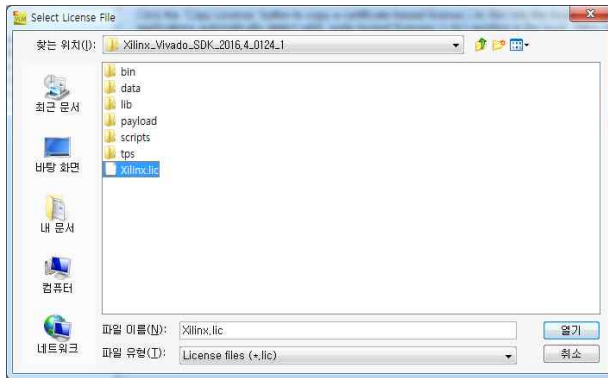
17. 라이선스 파일이 이메일로 전송된 것을 확인하는 화면을 보여준다.



18. [License Manager] 화면이 뜨면 ①[Load License]를 선택하고 ②[Copy License]를 클릭한다.



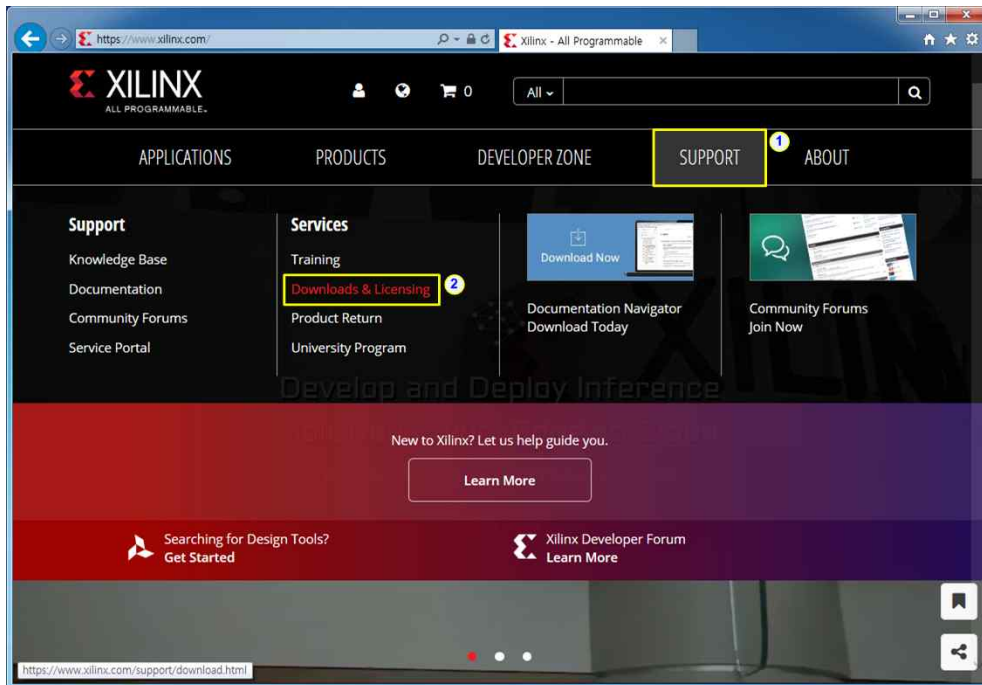
19. 라인선스 파일 [Xilinx.lic]을 선택하고 [열기]를 클릭하면, 라이선스가 설치됐다는 메시지를 확인한다.



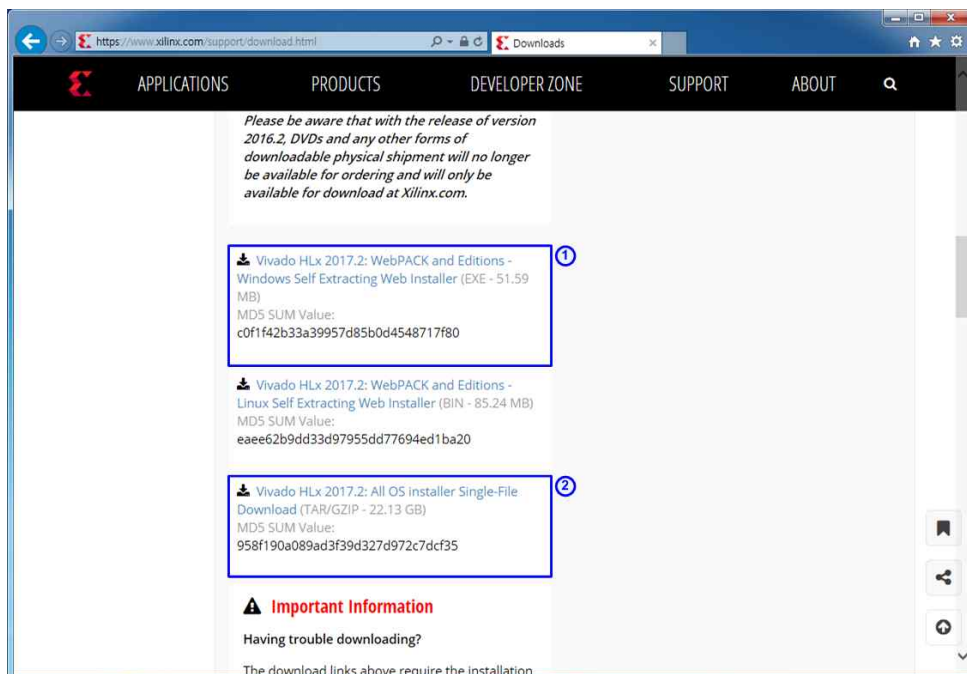
## - Vivado 2017. 2 설치

본 문서가 작성될 때 Vivado의 최신 version은 2017. 2 이지만, DIGCOM-XA1.0의 예제는 Vivado 2016. 4 version에서 오류없이 실행이 되므로 Vivado 2016. 4를 설치할 것을 권장한다 Vivado 2017. 2는 윈도우 7과 윈도우 10에서 모두 실행이 된다.

1. <http://www.xilinx.com>에 접속하여 [SUPPORT]⇒[Downloads & Licensing]을 클릭한다.

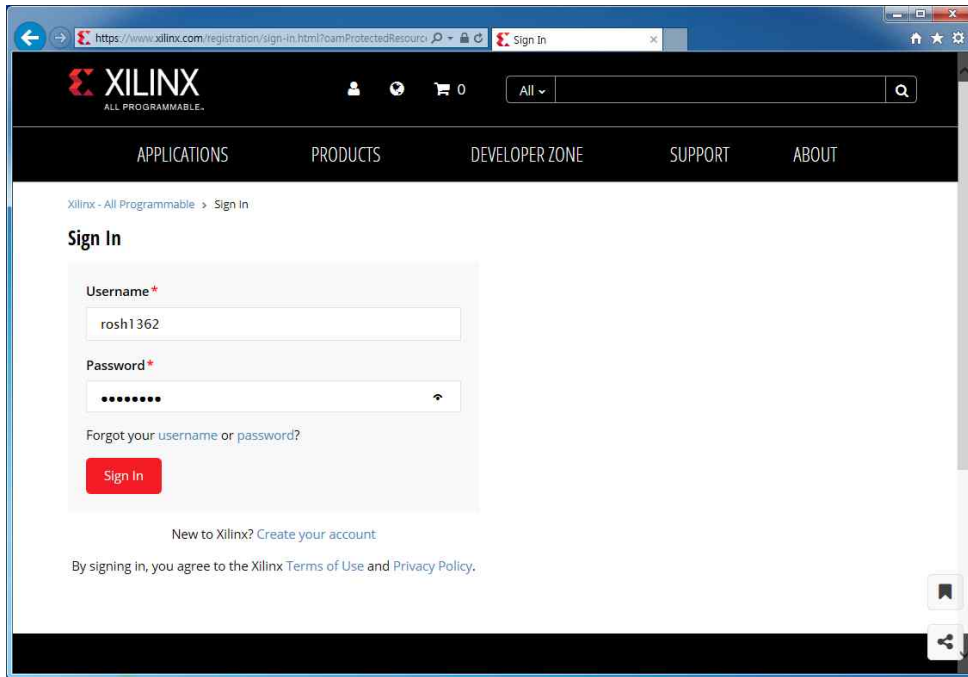


2. Download 화면에 설치할 Vivado version이 표시되며, 초기화면에 최종 version인 2017.2을 선택되어 있으며, 화면 아래에 설치 방법을 선택할 수 있다. 아래 그림에서와 같이 ①을 선택하면 Vivado 파일을 다운로드하면서 설치를 하며, ②를 선택하면 전체 파일을 다운로드 한 후에 설치를 시작한다. ①을 선택하면 네트워크 연결상태에 따라서 오히려 설치시간이 오래 걸릴 수 있기 때문에 ② 방법을 추천한다.(설치 방법에서는 2017. 2 version 설치에 대해서 설명하지만 DIGCOM-XA1.0은 2016. 4에서 매우 안정적으로 동작을 하므로 2016. 4 version 설치를 권장합니다.)



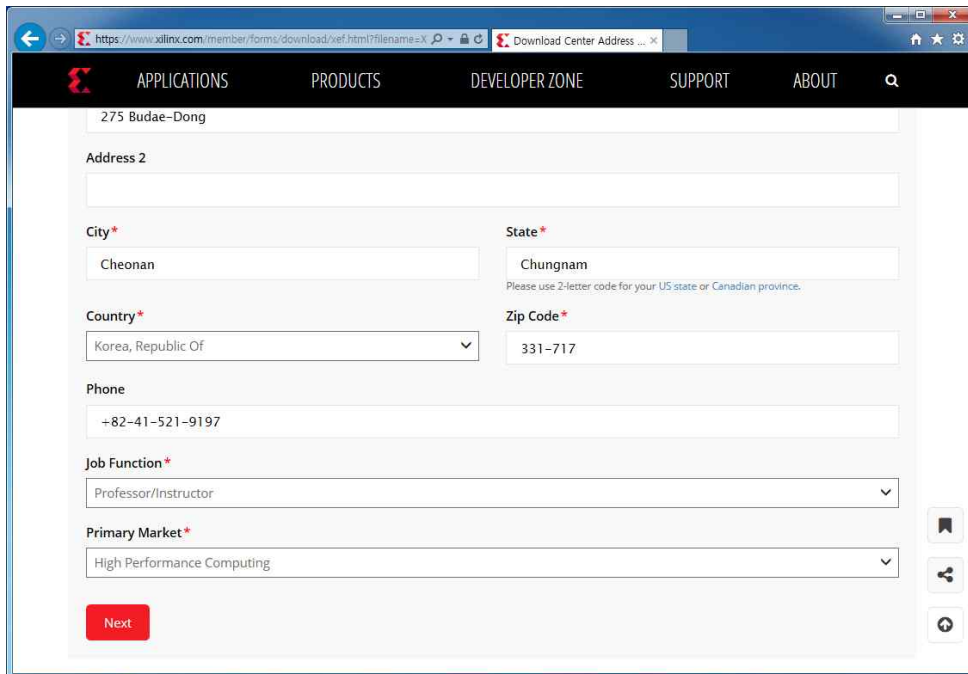


3. 로그인 화면이 나오면 [Username]와 [Password]를 입력하여 로그인을 하며, 회원이 아니면 회원으로 가입을 한 후 로그인을 한다.



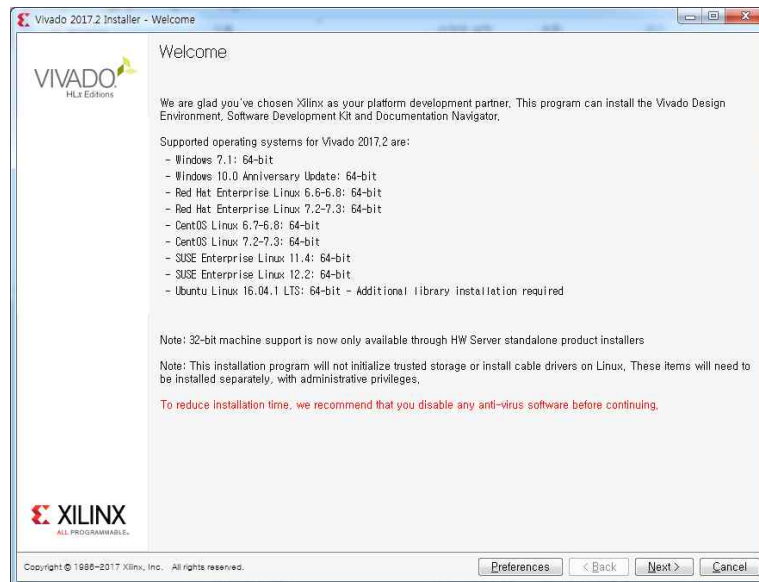
The screenshot shows the Xilinx website's sign-in page. The browser address bar displays 'https://www.xilinx.com/registration/sign-in.html?oamProtectedResource...'. The page header includes the Xilinx logo and navigation links: APPLICATIONS, PRODUCTS, DEVELOPER ZONE, SUPPORT, and ABOUT. The main content area is titled 'Sign In' and contains a form with the following fields: 'Username\*' (input: rosh1362), 'Password\*' (masked with dots), and a 'Forgot your username or password?' link. A red 'Sign In' button is positioned below the form. At the bottom of the form area, there is a link 'New to Xilinx? Create your account' and a disclaimer: 'By signing in, you agree to the Xilinx Terms of Use and Privacy Policy.'

4. 사용자에 대한 정보를 확인하는 화면이 나오며, 화면 아래에 [Next]를 클릭하여 다운로드를 시작한다.

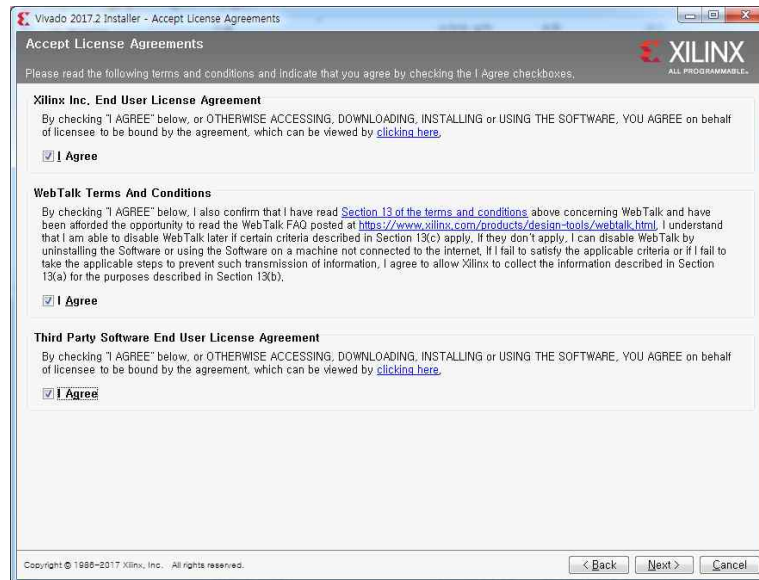


The screenshot shows the Xilinx member download form. The browser address bar displays 'https://www.xilinx.com/member/forms/download/def.html?filename=X...'. The page header includes the Xilinx logo and navigation links: APPLICATIONS, PRODUCTS, DEVELOPER ZONE, SUPPORT, and ABOUT. The form contains the following fields: 'Address 2' (empty), 'City\*' (input: Cheonan), 'State\*' (input: Chungnam), 'Country\*' (dropdown: Korea, Republic Of), 'Zip Code\*' (input: 331-717), 'Phone' (input: +82-41-521-9197), 'Job Function\*' (dropdown: Professor/Instructor), and 'Primary Market\*' (dropdown: High Performance Computing). A red 'Next' button is located at the bottom left of the form.

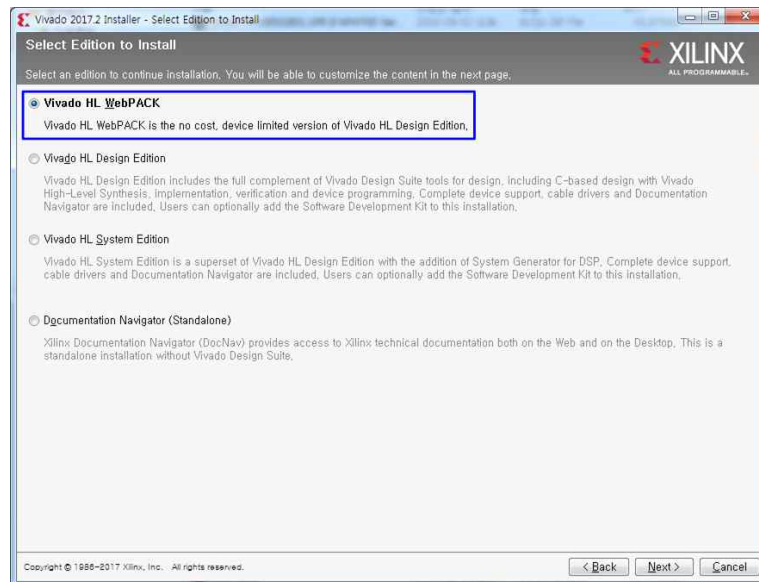
5. 다운로드한 압축파일을 해제하고 [setup]을 클릭하여 설치를 시작하면 아래 그림과 같이 설치 화면이 나오며 [Next]를 클릭하여 설치를 시작한다.



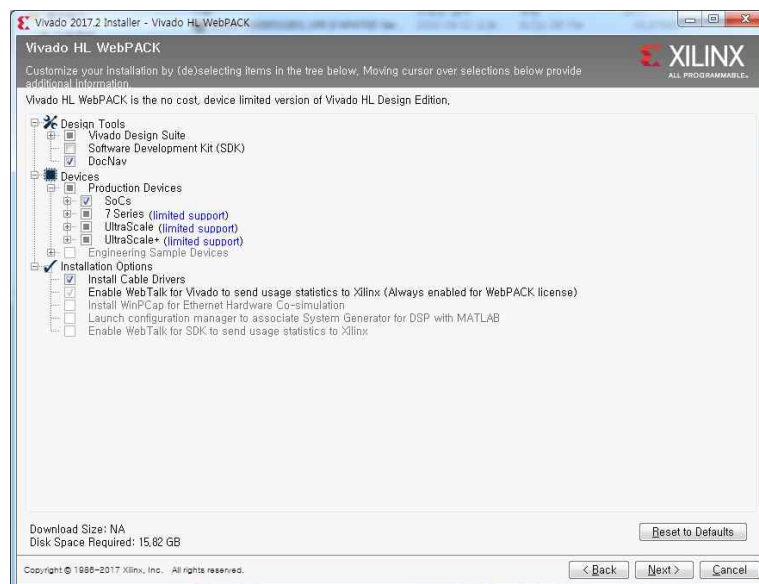
6. 라이선스 등의 동의하는 화면에서 모두 동의를 한 후에 [Next]를 클릭한다.



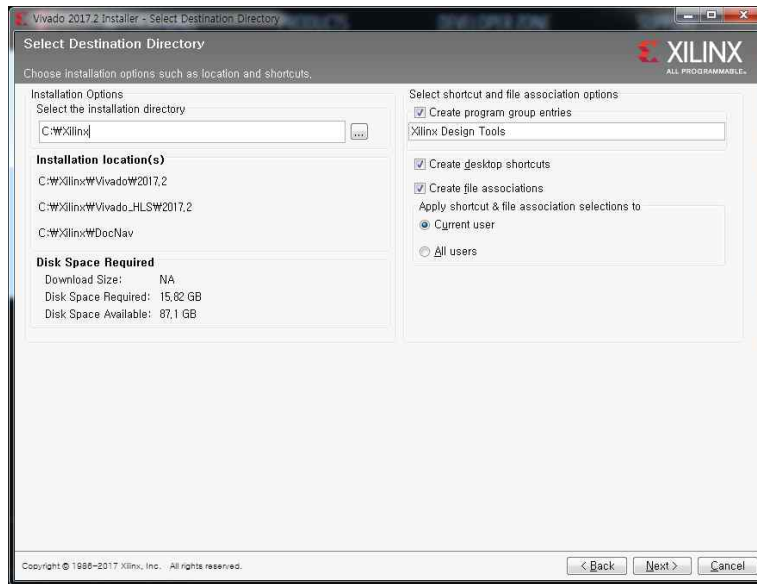
7. 설치할 “edition”을 선택해야 하며, [Vivado HL WebPACK]을 선택하고 [Next]를 클릭한다.



8. 설치할 항목을 선택하는 화면에서 [Next]를 클릭한다. 이 화면에서 설치시간을 줄이기 위해서는 [Device] 항목에서 Artix-7만 남기고 나머지를 모두 해제해도 된다.



9. 설치할 폴더를 선택하고 [Next]를 클릭한다.



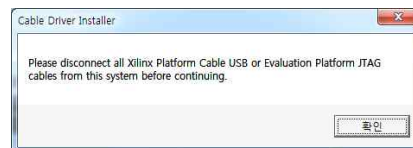
10. 설치에 대한 정보를 확인한 후 [Install]을 클릭하면 설치를 시작한다.



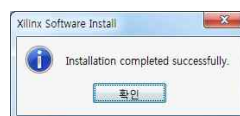
11. 아래 그림과 같이 설치 진행을 보여준다.



12. 설치 과정에서 아래 그림과 같이 Platform Cable을 제거하라는 메시지가 나오면 [확인]을 클릭하고 설치를 진행한다. 만일 Platform Cable이 연결되어 있으면 연결을 제거한다.



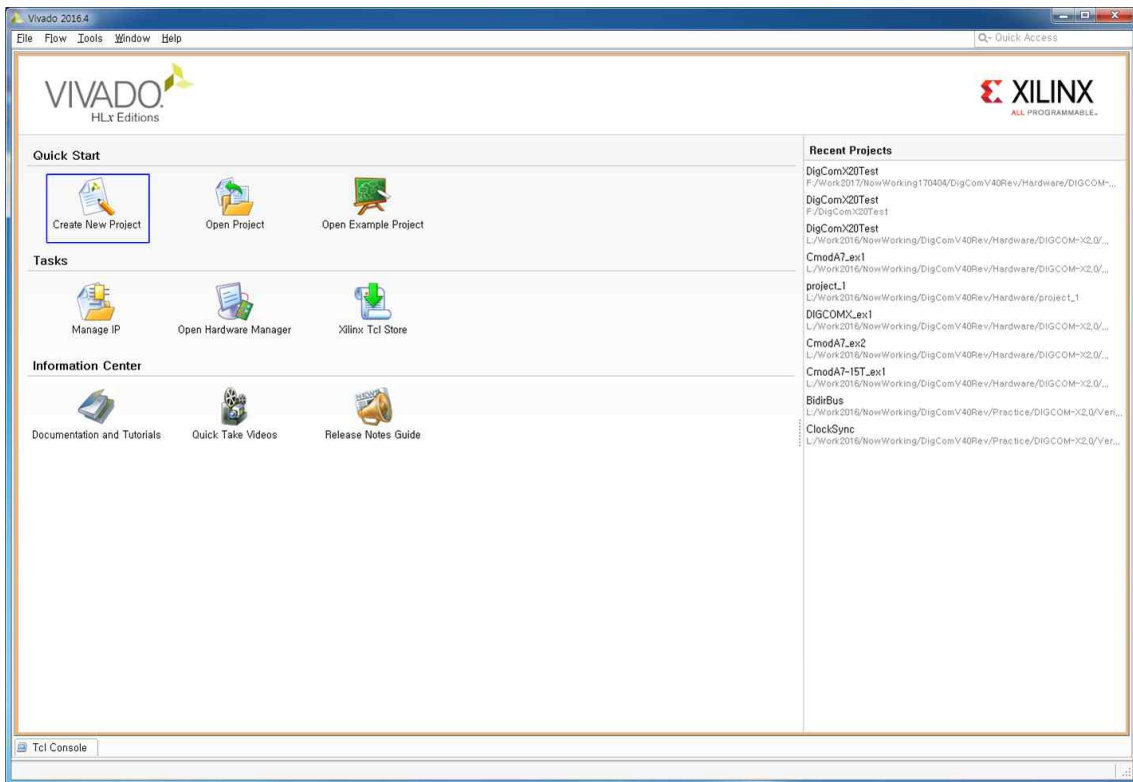
13. 설치를 마치면 아래와 같은 메시지가 출력된다.



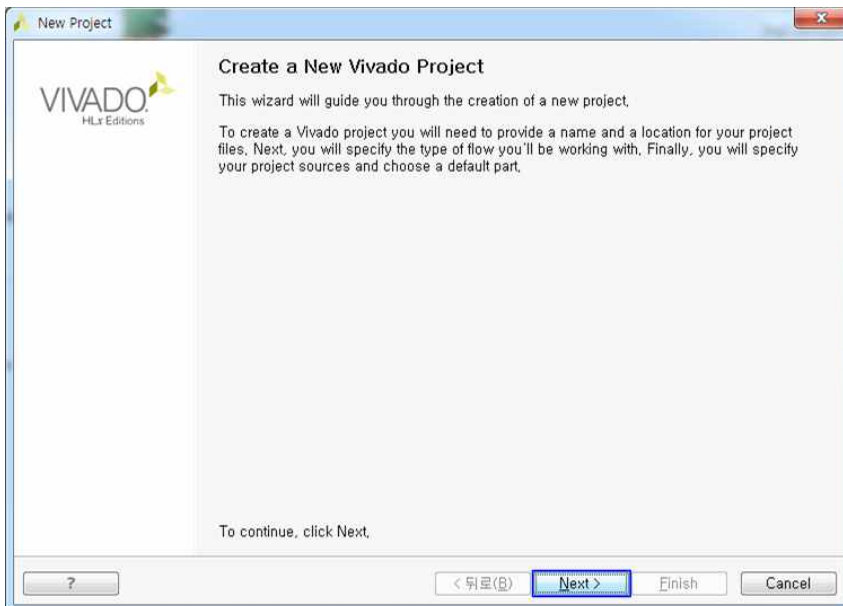
### Ⅲ. Vivado 2016. 4 사용

#### 1. 프로젝트의 생성

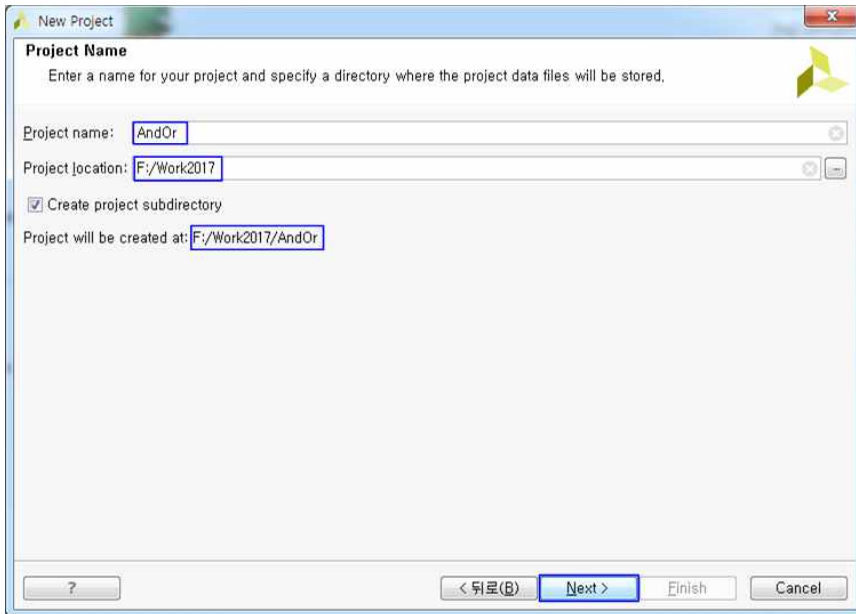
Vivado 2016.4를 실행시킨 [Create New Project]를 클릭하여 프로젝트 생성을 시작한다.



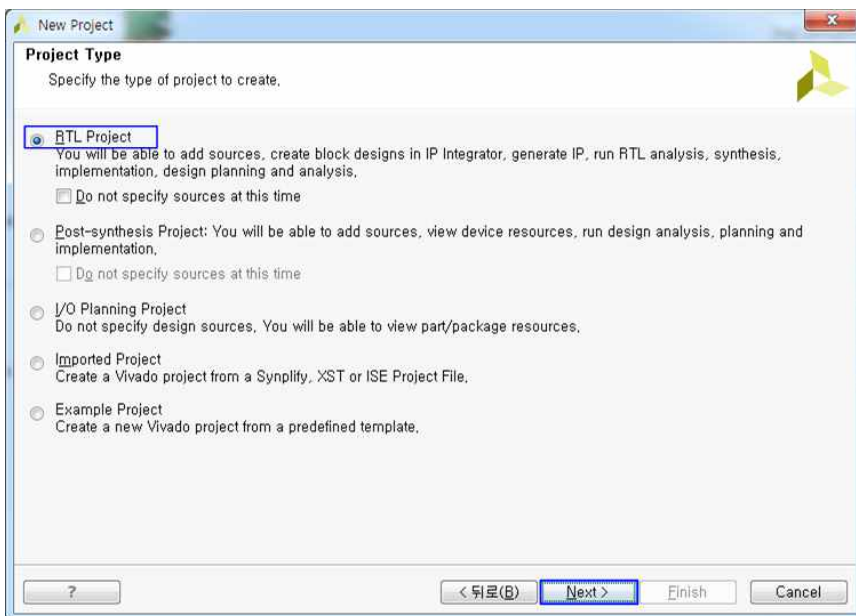
[Create a New Vivado Project] 창이 뜨면 [Next]를 클릭한다.



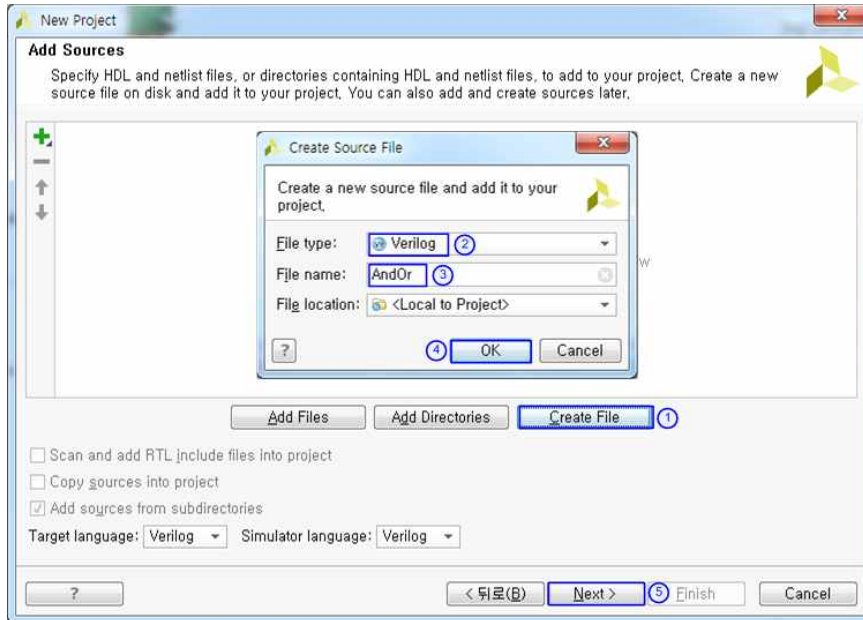
[Project name]창에서 [Project name]을 “AndOr”, [Project location]을 “F:/Work2017”로 각각 지정한다. 이 때 프로젝트는 “F:/Work2017/AndOr”에 생성된다. 그리고 [Next]를 클릭한다.



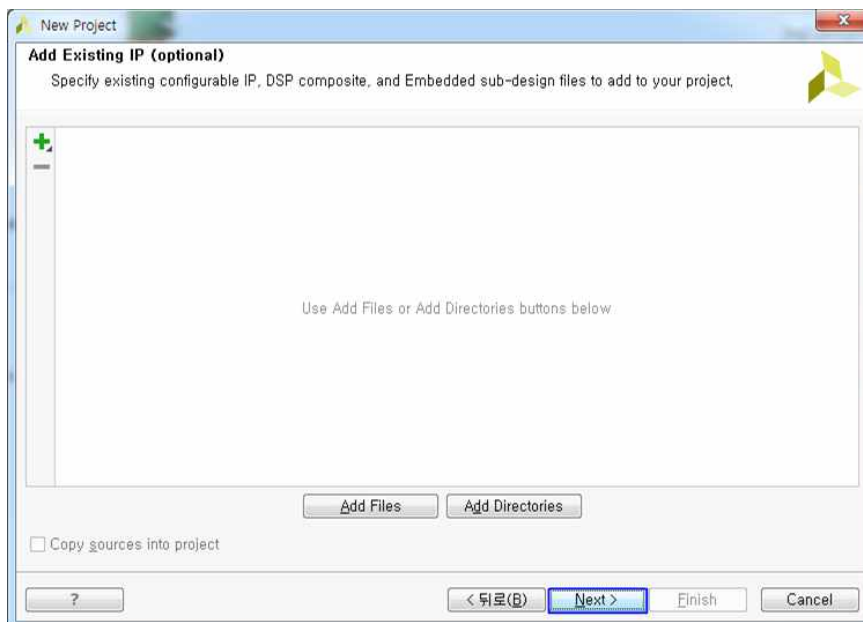
[Project Type]창에서 [RTL Project]가 체크된 것을 확인하고 [Next]를 클릭한다.



[Add Source]창에서는 설계에 사용될 언어와 설계 파일 이름을 지정해 준다. 이 창에서 설계 파일을 지정하기 위하여 [Create File]을 클릭하면 [Create Source File] 창이 뜨며, [File Type]에서 [Verilog]로 선택된 것을 확인하고 [File name]에 "AndOr"를 입력한 후에 [OK] 버튼을 클릭하면, "AndOr.v"파일과 파일이 생성되는 위치를 확인하고, [Next]를 클릭한다.

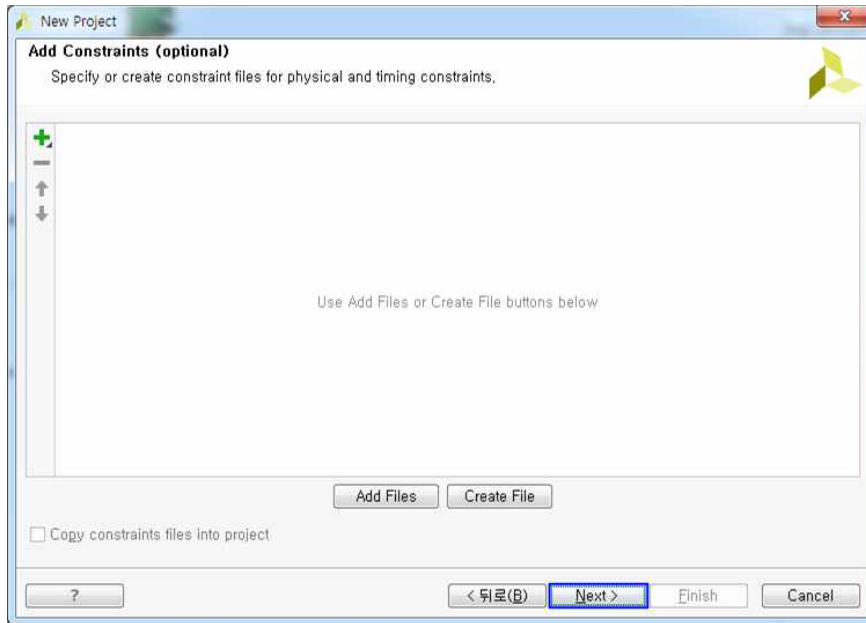


[Add Existing IP (optional)] 창에서는 프로젝트에 사용될 IP(Intellectual Property)를 추가 하지만 추가할 IP가 없으면 [Next]를 클릭한다.

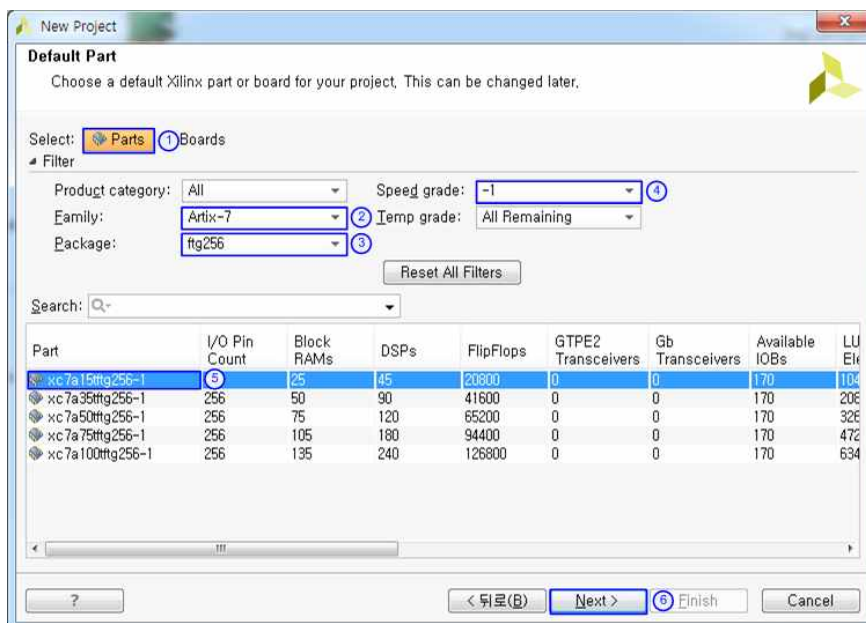




[Add Constraints (optional)] 창에서는 타이밍 또는 배치 등과 같은 설계의 성능을 향상시킬 수 있는 제약 조건을 지정해 주는 파일을 추가하거나 핀 할당을 지정할 수 있다. 본 프로젝트에서는 설계를 마친 후에 핀 할당만을 지정할 것이기 때문에 [Next]를 클릭한다.



[Default Part] 창에서는 사용될 FPGA 또는 보드를 지정하며, DIGCOM-XA1.0 키트는 XC7A15T-1FTG256I FPGA를 사용하므로 이에 해당하는 FPGA를 선택해야 한다. 그러므로 [Filter] 항목에서 [Select]를 [Parts]로 선택하고, [Family]를 [Artix-7], [Package]를 [ftg256] 그리고 [Speed grade]를 [-1]로 각각 선택하면 여기에 해당하는 FPGA 들이 나열되고, 이 중에서 [xc7a15tftg256-1]을 선택하고 [Next]를 클릭한다.

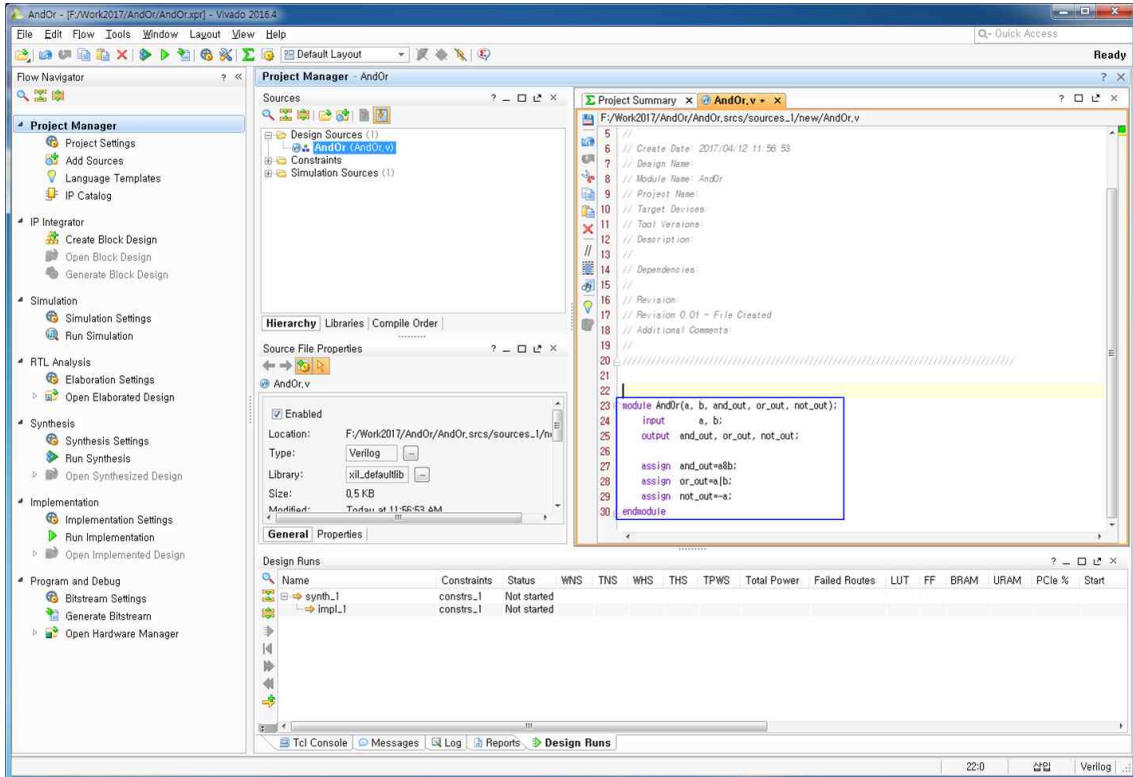


[New Project Summary] 창에서 생성될 프로젝트를 확인한 후에 [Finish]를 클릭하면 새 프로젝트가 생성된다.

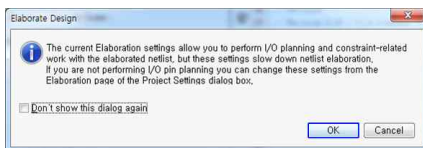




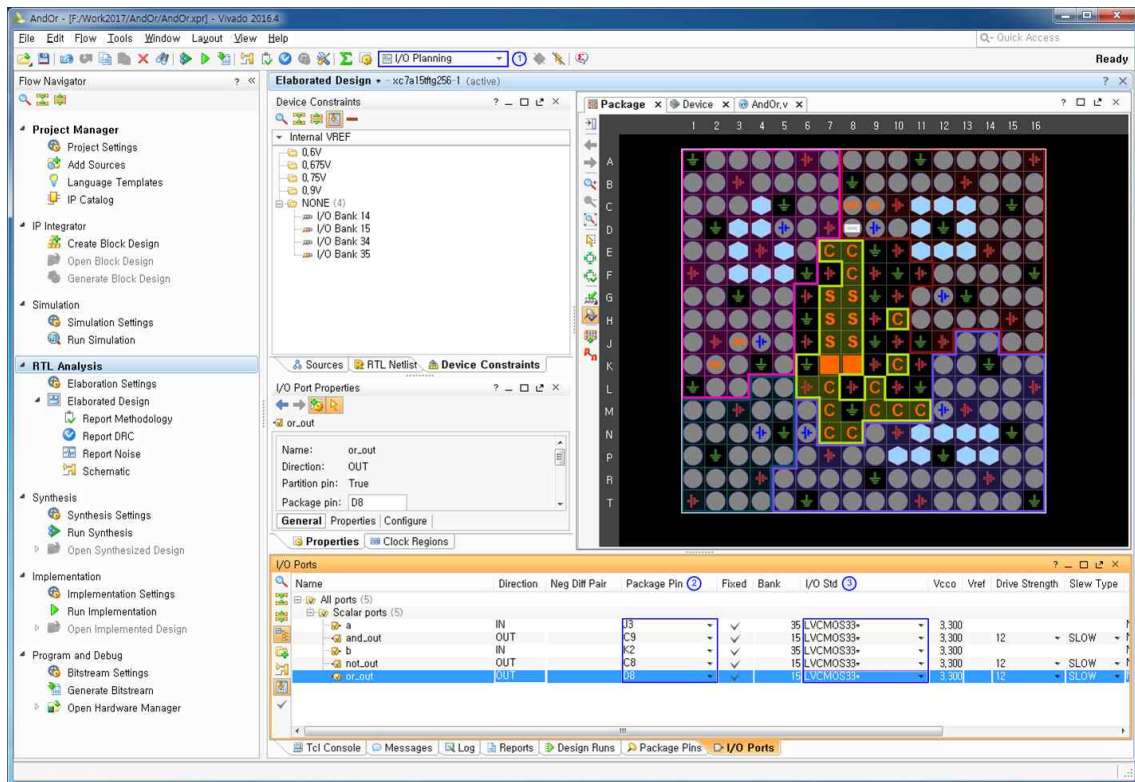
[AndOr.v] 탭에 Verilog 소스코드 입력을 하기 위한 템플릿이 나오며, 이 템플릿을 이용하여 설계하고자 하는 Verilog 소스코드를 입력한 후, 메인 메뉴에서 [File] ⇒ [Save File]을 클릭하여 [AndOr.v]를 저장한다.



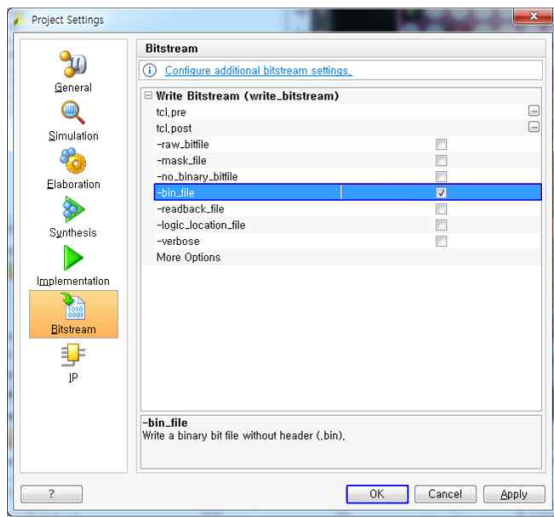
[RTL Analysis] ⇒ [Open Elaborated Design]을 클릭하면 [Elaborated Design] 창이 뜨며, [OK]를 클릭한다.



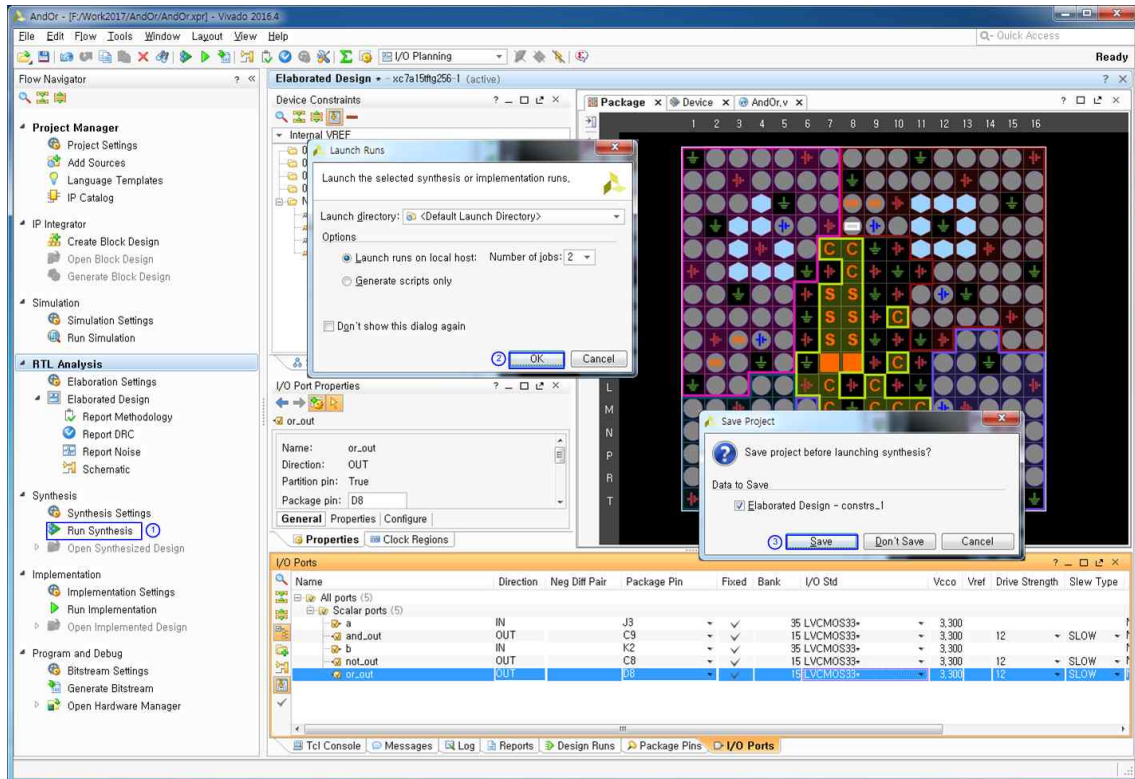
[Elaborated Design] 화면에서 입출력 포트에 대한 핀을 할당할 수 있으며, 그림과 같이 [Select or save window layout]을 [I/O Planning]을 선택한 후, 핀 번호를 할당하고, [I/O Std]를 [LVCMOS33]으로 선택한다. 입출력 포트의 핀 할당은 DIGCOM-XA1.0의 핀 할당표를 참고한다.



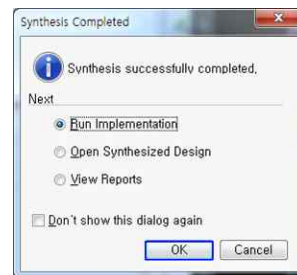
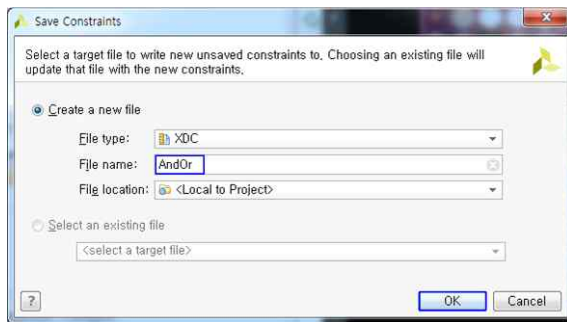
DIGCOM-XA1.0에 설계된 파일을 프로그램할 수 있는 파일의 종류는 .bit와 .bin 등 2가지가 있다. .bit 파일은 JTAG 프로그래밍 케이블을 사용하여 FPGA에 프로그래밍 하지만 전원이 꺼지면 FPGA 내용은 지원진다. .bin 파일은 QuadSPI에 프로그래밍 되며, 전원이 켜질 때마다 QuadSPI에서 FPGA로 프로그래밍 되므로 매번 다시 프로그래밍 할 필요가 없다. .bit 파일을 생성할 때 동시에 .bin 파일을 생성하기 위해 [Project Manager] ⇒ [Project Settings]을 클릭하면 [Project Settings] 화면이 뜨며, [Bitstream]을 클릭하고 [-bin\_file]을 체크한다.



합성(Synthesis)하기 위해 [Flow Navigator] 화면에서 [Synthesis] ⇒ [Run Synthesis]를 클릭하면 [Launch Runs] 창이 뜨며, 이 창에서 [OK]를 클릭하면 다시 [Save Project] 창이 뜨며, 이 창에서 [Save]를 클릭한다.

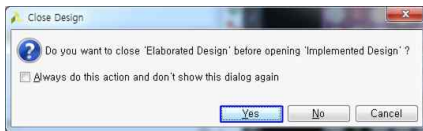
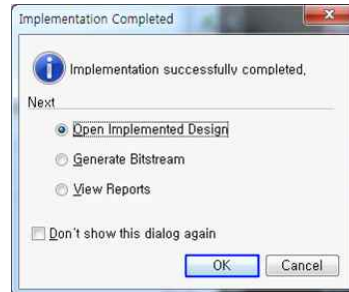
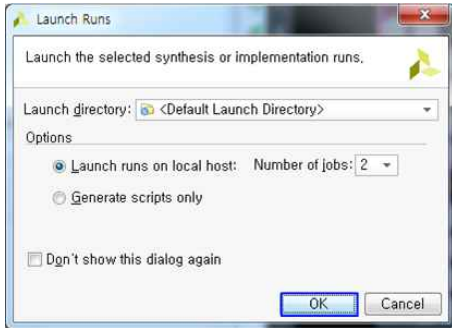


[Save Constraints] 창이 뜨면 [File name]에 Constraint 파일 이름인 “AndOr”를 입력하고 [OK]를 클릭하면 합성을 하고 마치면 [Synthesis Completed] 창이 뜨며, [OK]를 클릭한다.

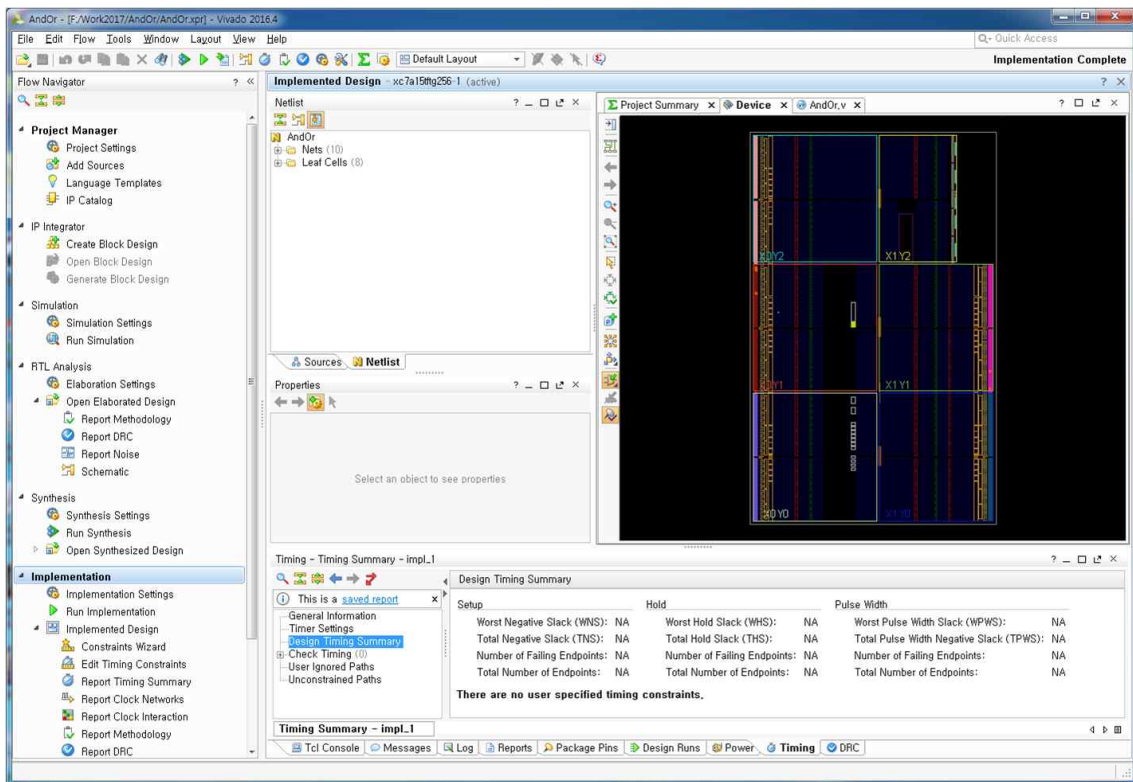




[Launch Runs] 창이 뜨면 다시 [OK]를 클릭하고, [Implementation Completed] 창이 뜨면 [Open Implemented Design]이 체크된 상태에서 [OK]를 클릭하고, [Close Design] 창이 뜨면 [Yes]를 클릭하여 [Elaborated Design] 화면을 close한다.



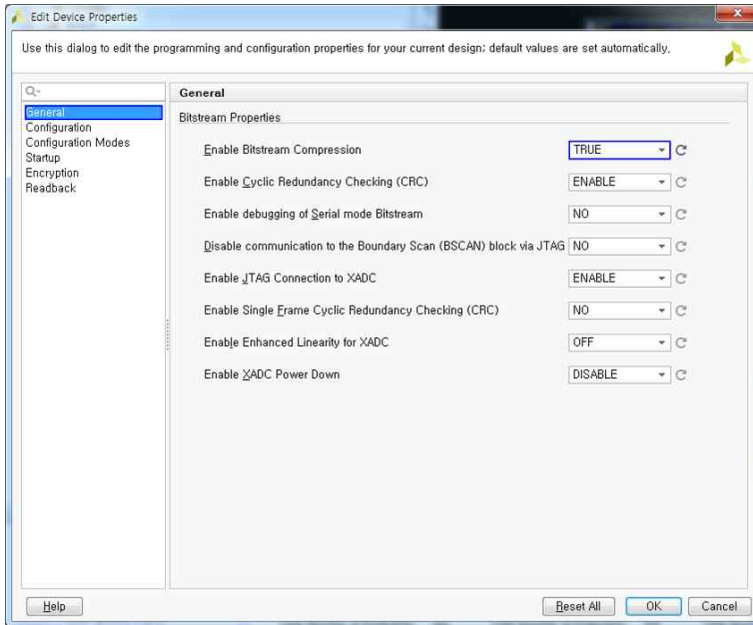
그림과 같이 창의 우측에 구현된 설계를 볼 수 있다.



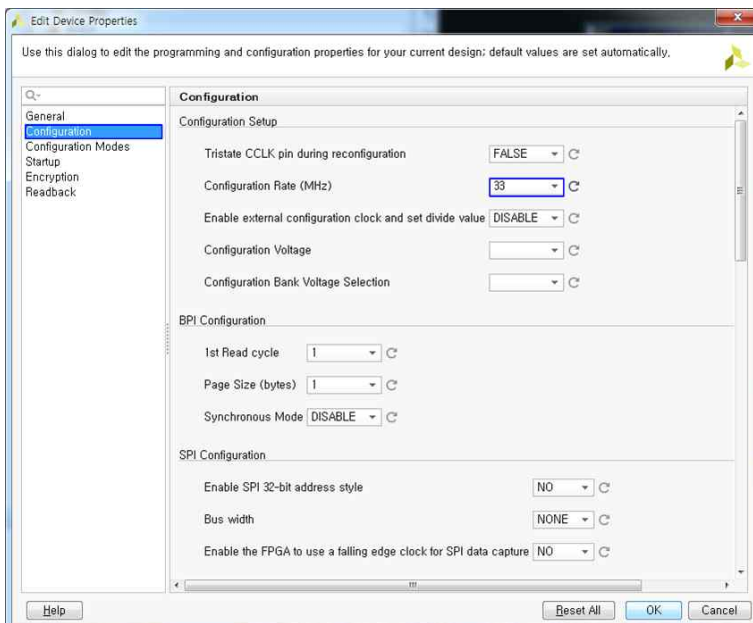


### 3. 프로그래밍(Programming) 및 실행

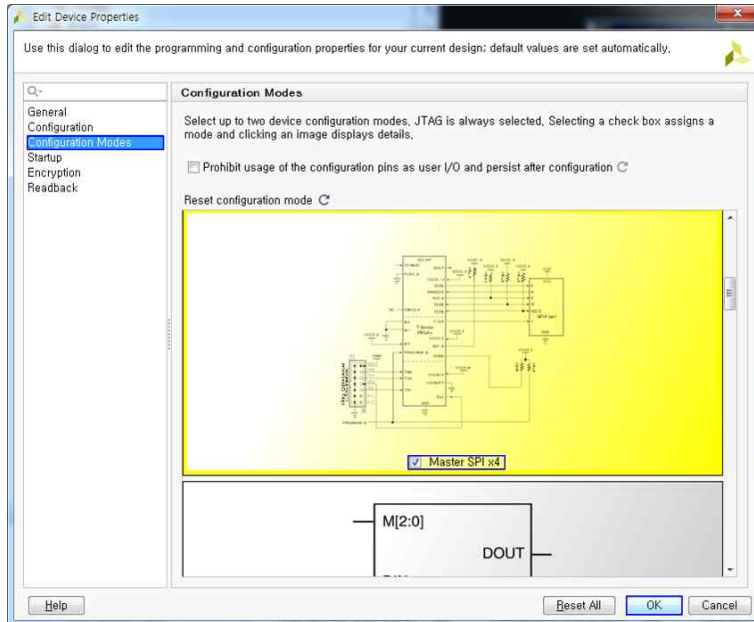
.bin 파일의 프로그래밍 속도를 높이기 위해 메뉴에서 [Tools] ⇒ [Edit Device Properties]를 선택하면 [Edit Device Properties] 창이 뜬다. 이 창에서 [General] ⇒ [Enable Bitstream Compression]을 [TRUE]로 선택한다



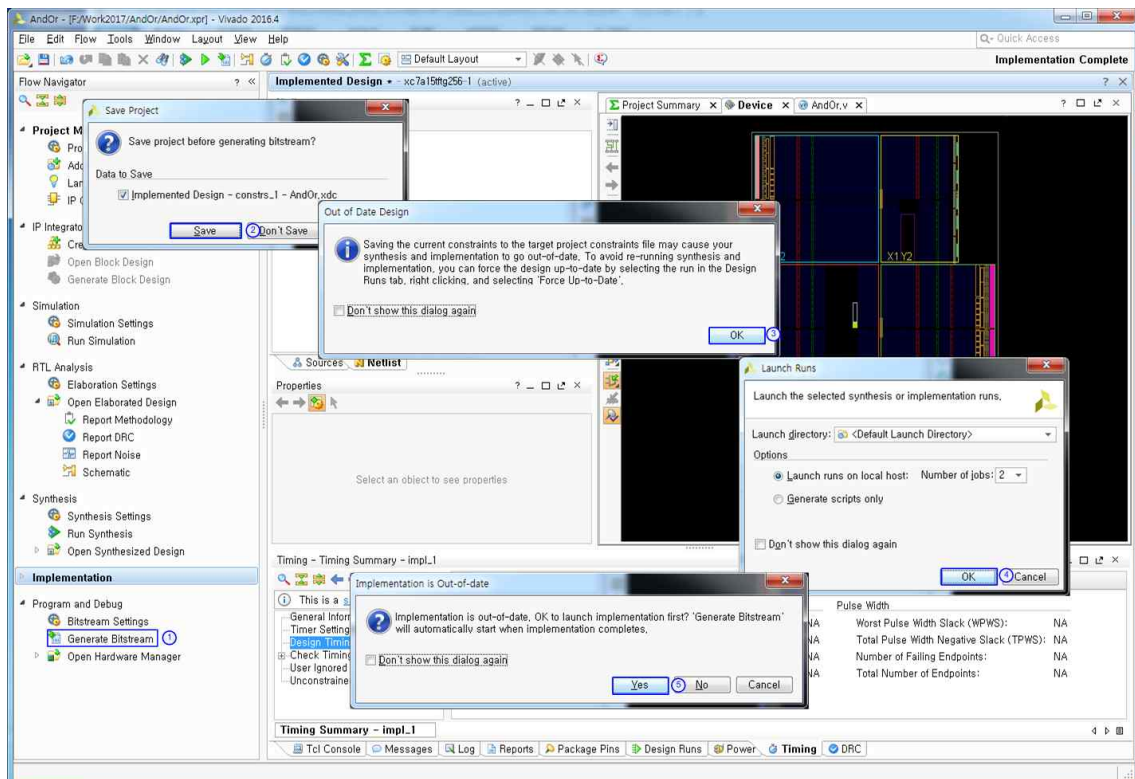
[Configuration] ⇒ [Configuration Rate (MHz)]를 [33]으로 선택한다.



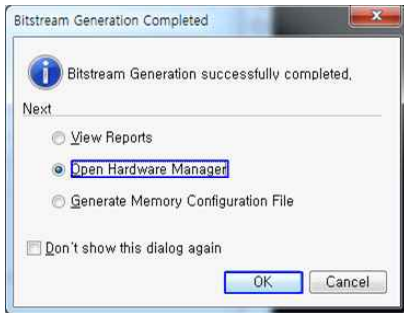
[Configuration Modes]에서 [Master SPIx4]를 선택한 후 [OK]를 클릭한다.



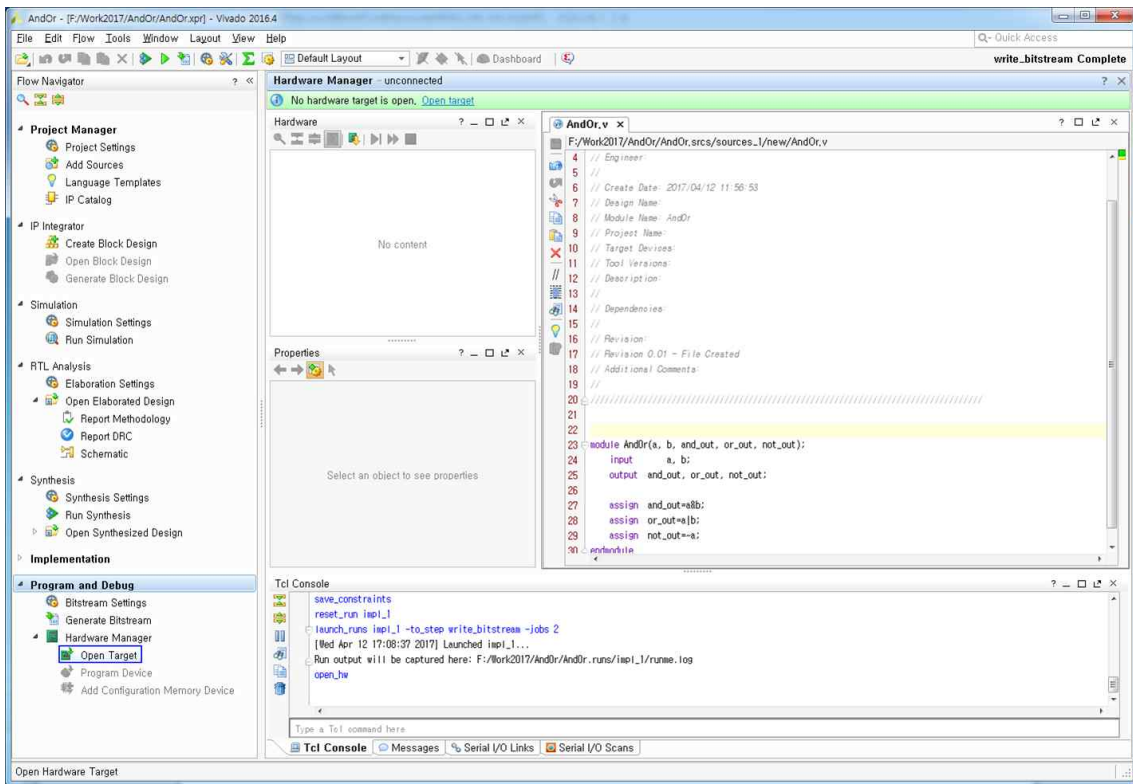
[Flow Navigator] 화면에서 [Program and Debug] ⇒ [Generate Bitstream]을 클릭하면 [Save Project] 창이 뜨고 [Save]를 클릭해서 변경된 constraint 파일을 저장하고, [Out of Date Design] 창이 뜨면 [OK] 클릭하고, [Launch Runs] 창이 뜨면 [OK]를 클릭하고, 다시 [Out of Date Design] 창이 뜨면 [OK] 클릭하면 Bitstream을 생성하기 시작한다.



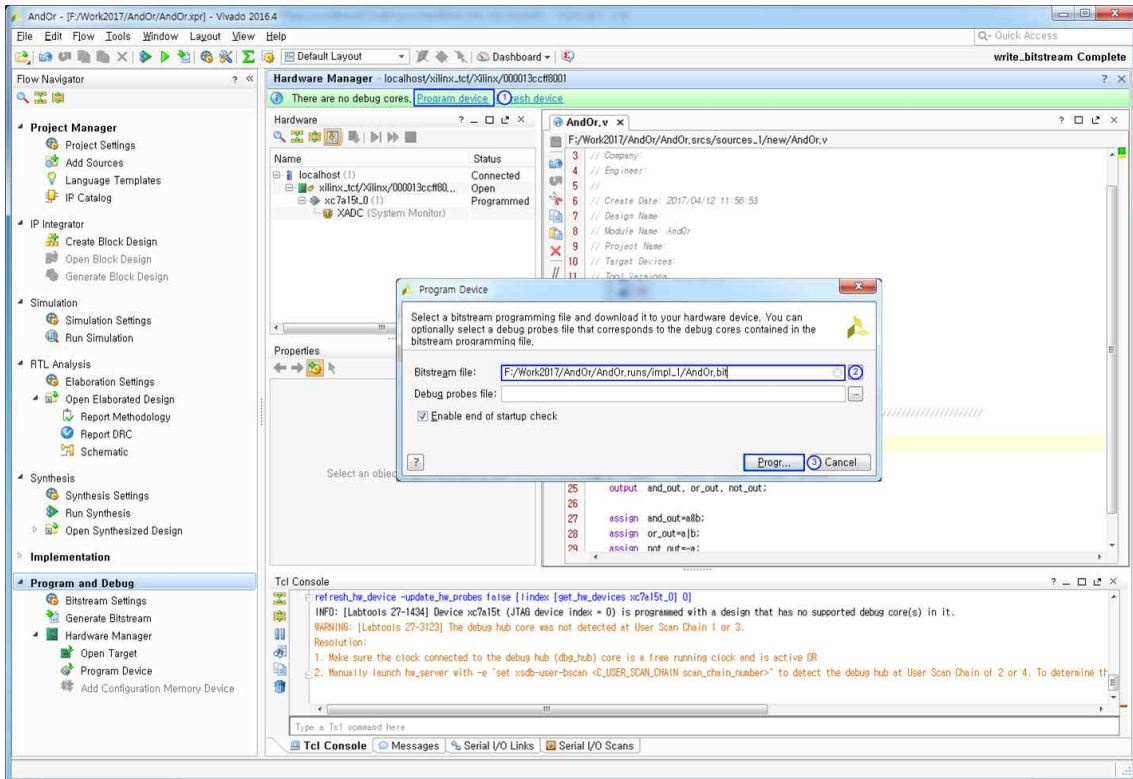
[Bitstream Generation Completed] 창이 뜨면 [Open Hardware Manager] 박스를 체크하고 [OK]를 클릭한다.



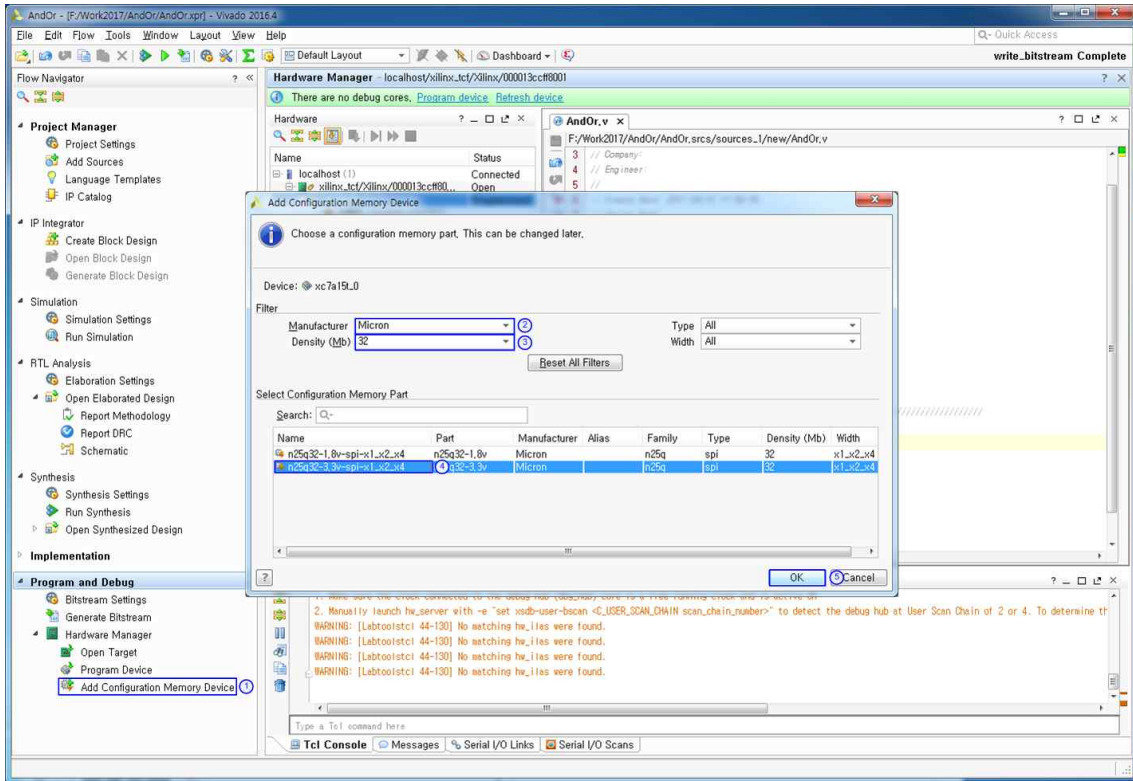
[Flow Navigator]에서 [Program and Debug] ⇒ [Hardware Manager] ⇒ [Open Target] ⇒ [Auto Connect]를 클릭한다.



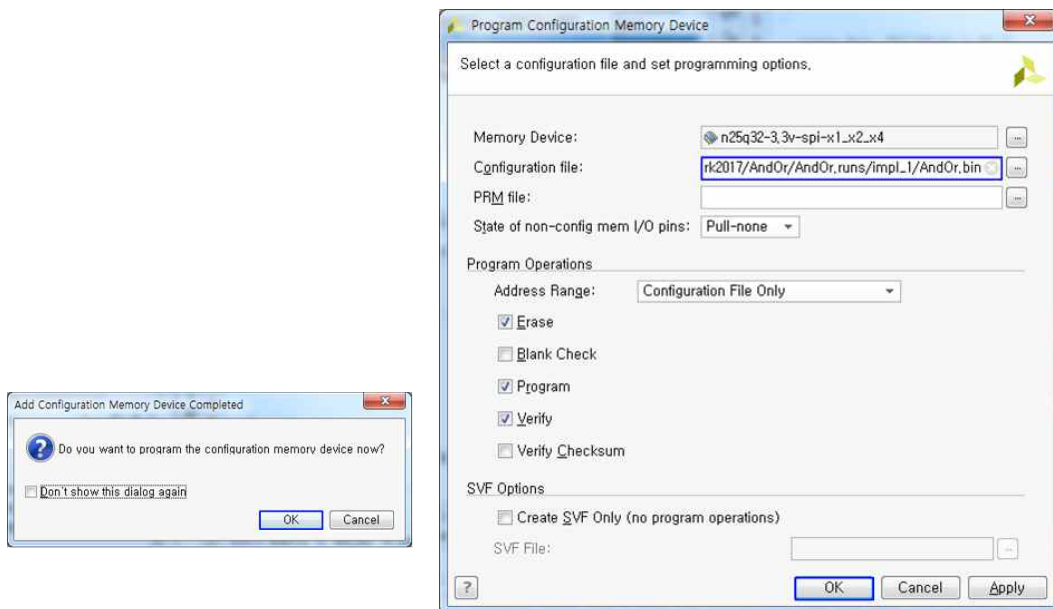
DIGCOM-XA1.0 키트의 전원이 켜진 상태에서 [Program Device] ⇒ [xc7a15t\_0]을 클릭하면 [Program Device] 창이 뜨고, 프로젝트 폴더에서 “AndOr.bit” 파일이 선택된 것을 확인한 후에 [Program]을 클릭한다. 프로그래밍이 완료되면 키트에서 동작을 확인한다.



[Flow Navigator]에서 [Program and Debug] ⇒ [Hardware Manager] ⇒ [Add Configuration Memory Device] ⇒ [xc7a15t\_0]를 클릭하면 [Add Configuration Memory Device] 창이 뜨며, [Manufacturer] ⇒ [Micron], [Density] ⇒ [32], [Select Configuration Memory Part] ⇒ [n25q32-3.3v-spi-x1\_x2\_x4]를 선택하고 [OK]를 클릭한다.



[Add Configuration Device Completed] 창이 뜨면 [OK]를 클릭하면 [Program Configuration Memory Device] 창이 뜨고 [Configuration File]을 프로젝트 폴더에서 “AndOr.bin”을 선택한 후 [OK]를 클릭한다.



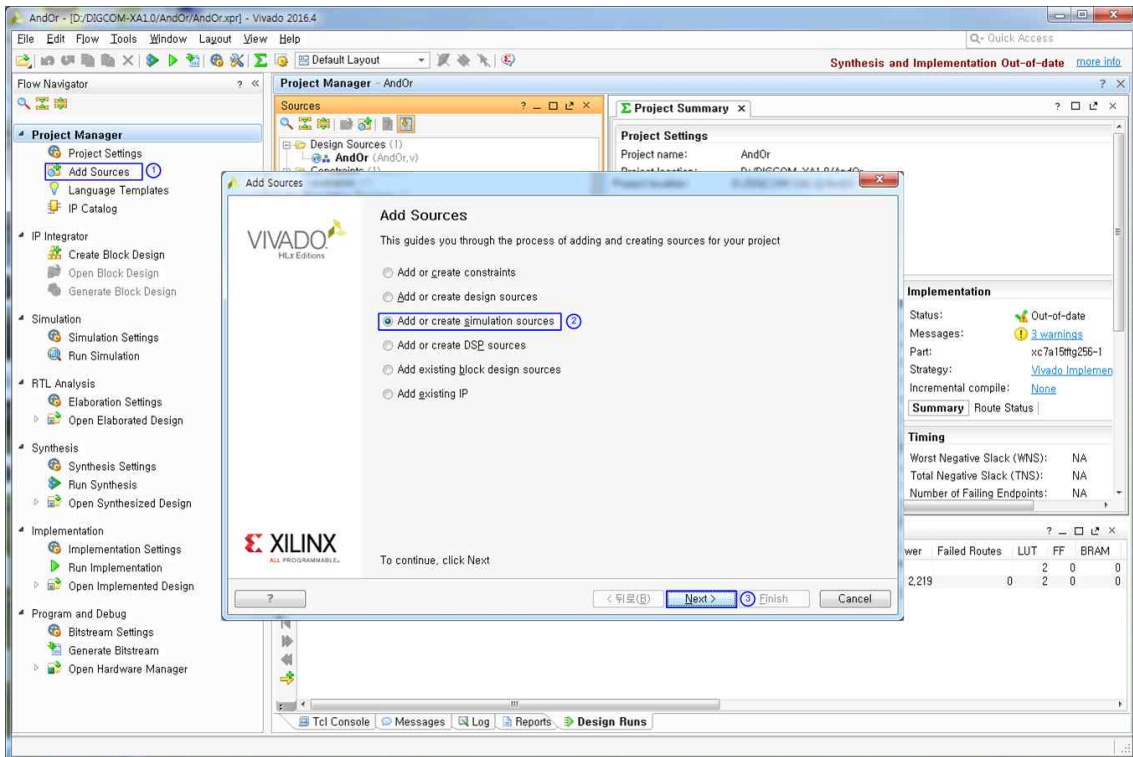
프로그래밍이 완료되면 [Program Flash] 창이 뜨고, 키트의 전원을 껐다가 다시 켜서 동작을 확인한다.



#### 4. 시뮬레이션

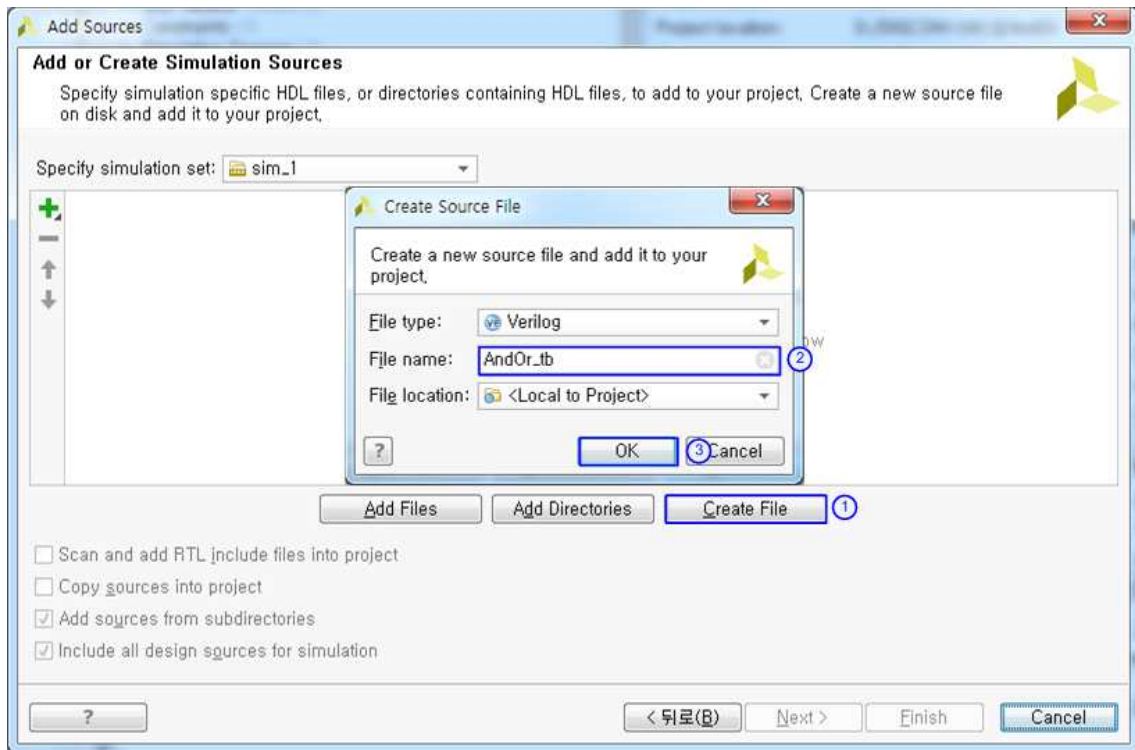
시뮬레이션 Simulation은 Verilog 또는 VHDL로 설계한 회로를 디버깅할 수 있는 가장 좋은 방법이다. 시뮬레이션하지 않고 FPGA에 다운로드하여 실행할 수도 있지만, 제대로 동작하지 않을 경우 디버깅이 쉽지 않으므로 많은 시간만 소모된다. 따라서 시뮬레이션으로 설계를 검증하고 FPGA에서 실행하는 것이 가장 확실한 방법이다.

[Flow Navigator]에서 [Project Manager] ⇒ [Add Sources]를 클릭하면 [Add Sources] 창이 뜨며, [Add or create simulation sources]를 체크하고 [Next]를 클릭한다.

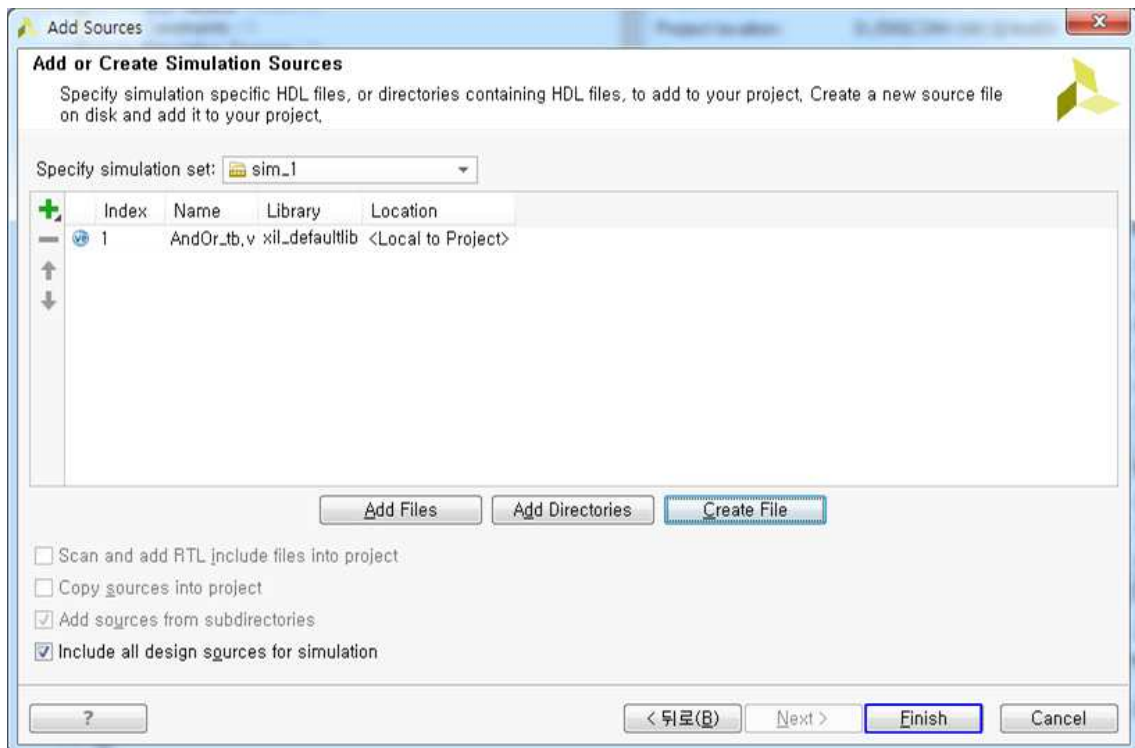




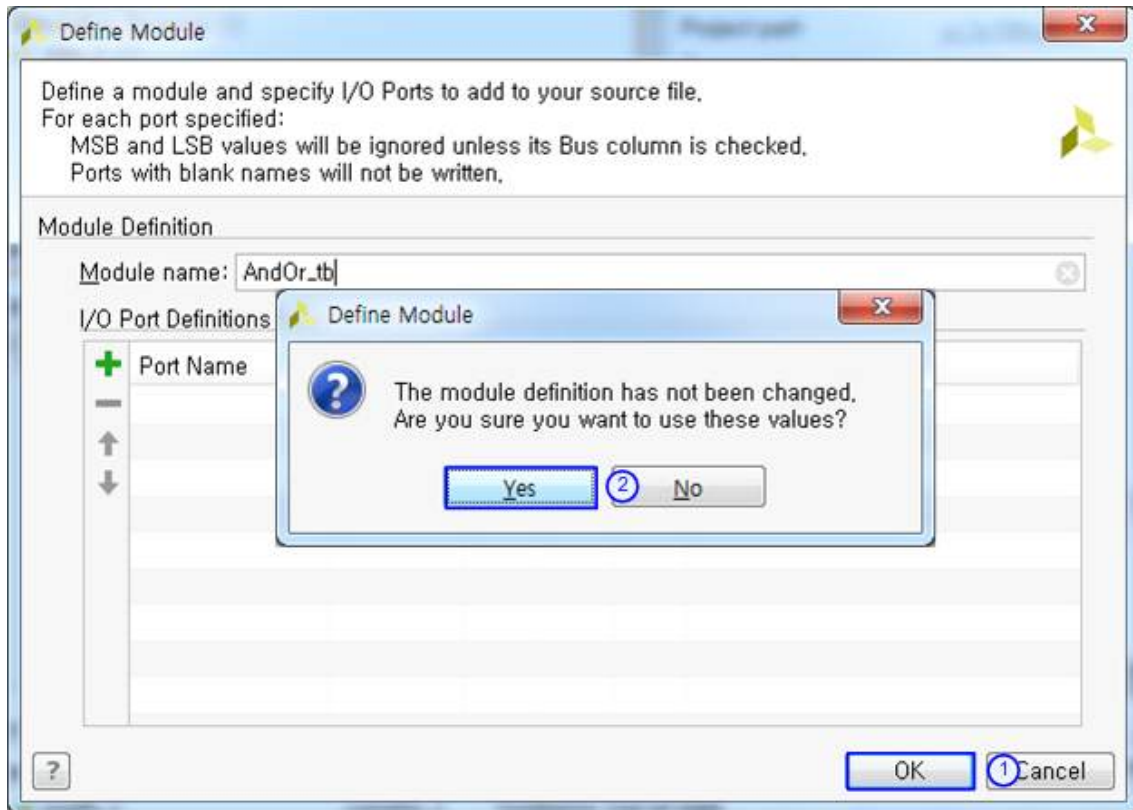
[Add Sources] 창에서 [Create File] 버튼을 클릭하면 [Create Source file] 창이 뜨며, [File name]에 시뮬레이션 테스트 벤치 파일이름 “AndOr\_tb”를 입력하고 [OK]를 클릭한다.



[Add Sources] 창에 테스트 벤치 파일이름 “AndOr\_tb.v”가 표시되며, [Finish]를 클릭한다.

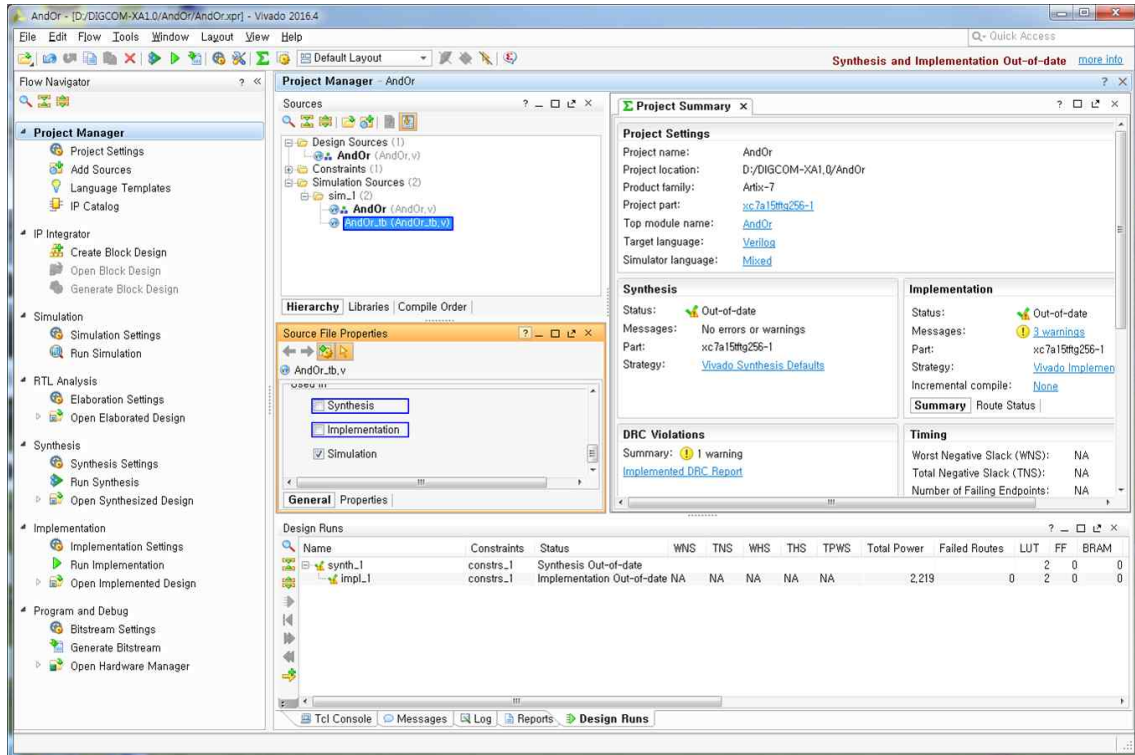


[Define Module] 창에서 입출력 포트에 대한 정의는 소스 코드 입력을 마친 후에 하므로 [OK]를 클릭하고, 모듈이 바뀌지 않았다는 창이 뜨면 다시 [Yes]를 클릭한다.

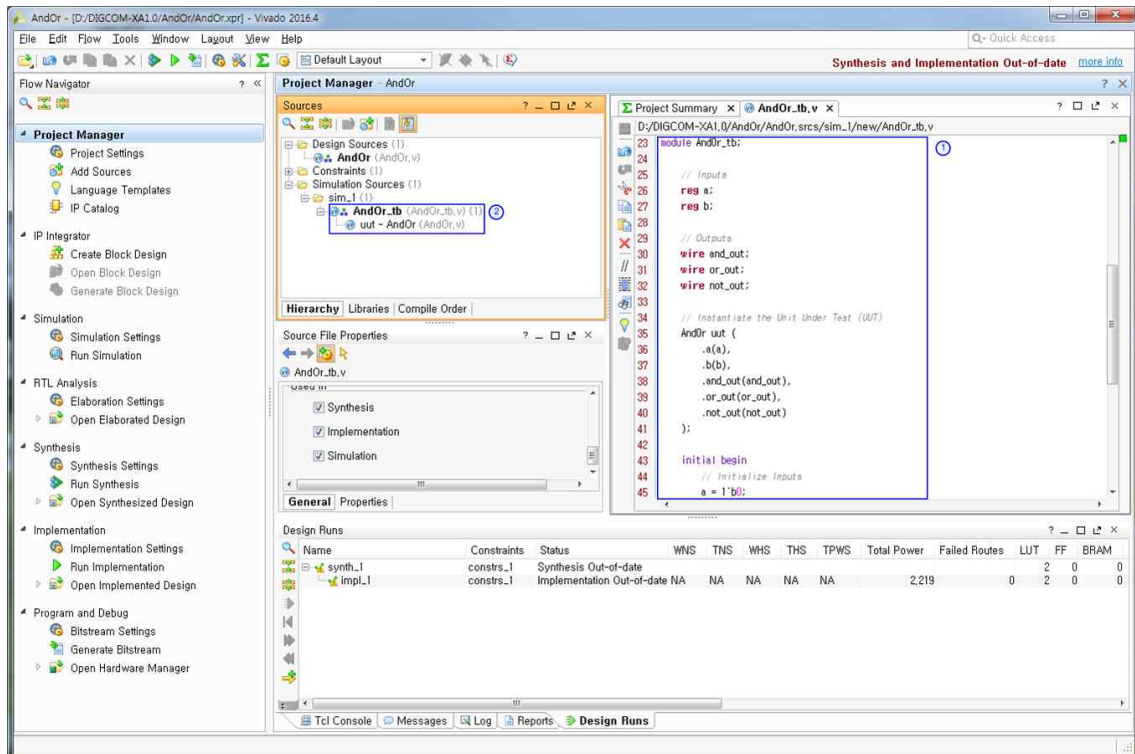




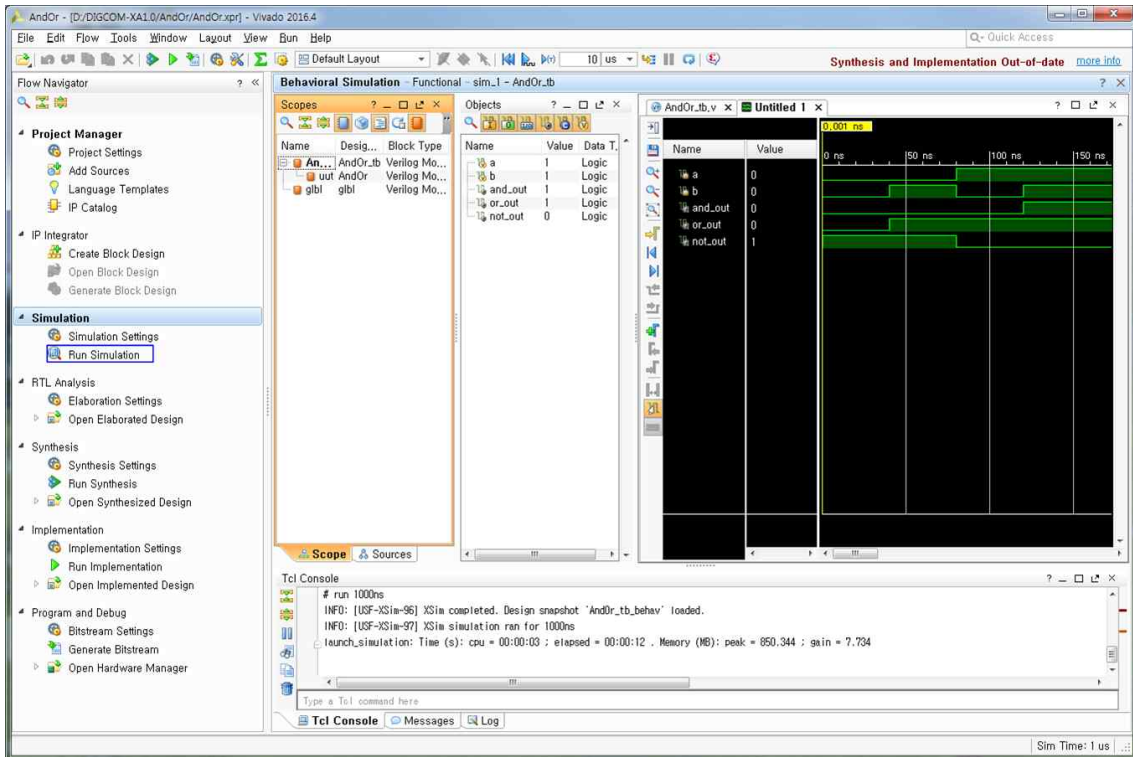
[Project Manager] 화면에 [AndOr\_tb (AndOr\_tb.v)]가 프로젝트에 포함된 것을 확인할 수 있으며, [AndOr\_tb (AndOr\_tb.v)]를 선택한 후 아래 [Source File Properties] 화면에서 [Synthesis]와 [Implementation]의 체크를 해제한다.



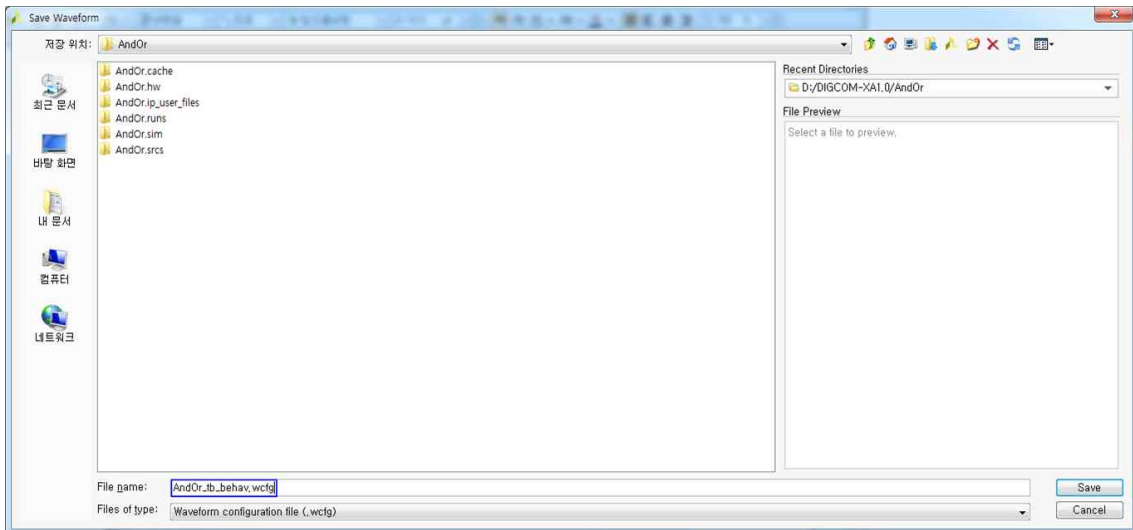
[AndOr\_tb]를 더블 클릭하고 테스트 벤치 코드를 입력한 후 메인 메뉴에서 [File] ⇒ [Save File]을 클릭하면 그림과 같이 시뮬레이션을 위한 hierarchy가 구성된다.



[Simulation] ⇒ [Run Simulation] ⇒ [Run Behavioral Simulation]을 클릭하면 시뮬레이션이 실행되고 그림과 같이 시뮬레이션 실행 결과 파형이 보인다.



파형을 저장하기 위해서 메인 메뉴에서 [File] ⇒ [Save Waveform Configuration]을 클릭하고 “AndOr\_tb\_behav.wcfg” 파일을 저장한다.



### Ⅲ. 조합논리회로 설계 (DIGCOM-XA1.0 실습 예제)

Vivado Design Suite에서는 schematic 설계를 지원하지 않으므로 교재에서 schematic으로 설계한 부분은 제외 했거나, 서브 모듈을 설계한 후에 symbol을 생성해서 schematic으로 설계한 프로젝트는 port mapping으로 대체 설계하는 과정을 설명했다. 본 문서에서는 ISE Design Suite 대신에 Vivado Design Suite로 설계를 할 때 교재에서 바뀌는 내용을 설명한다.

아래 표는 Vivado Design Suite를 사용해서 설계할 때 DIGCOM-XA1.0에서 실행하는 예제들을 보여준다.

절, 제목	Xilinx	
	Verilog	VHDL
3.1 기본 게이트(AND, OR, NOT)	AndOr	AndOr
Lab.1 설계 소프트웨어를 이용한 디지털 논리회로 설계	SimpleGates	SimpleGates
3.2 전가산기(Full Adder)	FullAdder1 FullAdder2 FullAdder3	FullAdder1 FullAdder2 FullAdder3
Lab.2 전감산기(Full Subtractor) 설계	FullSubtractor	FullSubtractor
3.3 디코더(Decoder)	Decoder-Assign Deocder-Case	Decoder
Lab.3 인코더(Encoder) 설계	Encoder	Encoder
3.4 입출력 장치 실습(스위치 인코딩)	SwitchEncoder	SwitchEncoder
Lab.4 7-세그먼트 FND(Flexible Numeric Display) 디코더 설계	FNDDecoderI FNDDecoderII	FNDDecoderI FNDDecoderII
3.5 멀티플렉서(Multiplexer)	Multiplexer-Case Multiplexer-if	Multiplexer
Lab.5 디멀티플렉서(Demultiplexer) 설계	DeMultiplexer-Case DeMultiplexer-if	DeMultiplexer
3.6 크기 비교기(Comparator)	Comparator	Comparator
Lab.6 수의 정렬회로 설계	Sorting	Sorting
3.7 n비트가산/감산기 (Adder/Subtractor)	nBitAddSubNameRef nBitAddSubPositionRef nBitAddSub1 nBitAddSub2	nBitAddSubNameRef nBitAddSubPositionRef nBitAddSub1 nBitAddSub2
Lab.7 BCD 가산기 설계	BCDAdder	BCDAdder
3.8 '1' 개수 카운터('1' Counter)	OneCounter	OneCounter
Lab.8 Leading one counter	LeadingOneCounter	LeadingOneCounter
3.9 패리티 발생기(Parity Generator)	ParityGenerator-BitWise ParityGenerator-Function	ParityGenerator
Lab.9 패리티 검사기(Parity Checker) 설계	ParityChecker	ParityChecker
3.10 리플 가산기(Ripple adder)	RippleAdder	RippleAdder
Lab.10 Carry look ahead 가산기 설계	CarryLookAheadAdder	CarryLookAheadAdder

### **3.1 기본 논리 게이트(AND, OR, NOT)**

Vivado Design Suite에서는 schematic 설계를 지원하지 않으므로 AndOr(sch) 프로젝트는 예제에서 제외하였으며, Lab. 1의 SimpleGates(sch) 프로젝트도 제외하였다.

### **3.2 전가산기(Full Adder)**

Vivado Design Suite에서는 schematic 설계를 지원하지 않으므로 FullAdder(sch) 프로젝트는 예제에서 제외하였으며, Lab. 1의 FullSubtractor(sch) 프로젝트도 제외하였다.

### 3.7 n비트가산/감산기(Adder/Subtractor)

Vivado Design Suite에서는 schematic 설계를 지원하지 않으므로 nBitAdderSub(sch) 프로젝트를 port mapping 방식으로 수정하였다.

#### (1) Port mapping에 의한 n비트 가산/감산기 설계(Verilog)

앞에서 전가산기와 전감산기를 HDL(verilog/VHDL)로 설계했다. 그러나 앞에서 설계한 내용은 한 비트에 대한 설계이며, 실제로 가산기와 감산기는 한 비트 이상의 입력에 대한 연산이 이루어져야 한다. 따라서 여기서는 다수의 비트 입력을 가진 가산기/감산기를 HDL에서 port mapping 방식에 의해 설계한다.

n비트 가산/감산기에서는 앞에서 이미 설계한 전가산기를 이용하여 4개의 객체(instance)를 생성한 후에 생성된 객체들 간에 포트를 서로 연결함으로써 설계한다. 이 회로의 기본적인 구성은 4비트 입력 a와 b가 피연산자이고, 입력 M이 '0'이면 가산기 동작을 실행하며 '1'이면 감산기 동작을 실행한다. 가산기와 감산기는 조합논리회로에서 가장 기본이 되는 회로지만, 뺄셈 동작에서 음수 결과가 나올 때를 고려하면 단순하지 않다.

[코드 nBitAddSubNameRef]

```
module nBitAddSub0(a, b, m, c_out, sum);
    input [3:0] a, b;
    input      m;
    output     c_out;
    output [3:0] sum;

    wire [2:0] c;

    // referenced by name
    FullAdder FA0(
        .fa    (a[0]),
        .fb    (m^b[0]),
        .fc_in (m),
        .fsum  (sum[0]),
        .fc_out (c[0])
    );

    FullAdder FA1(
        .fa    (a[1]),
        .fb    (m^b[1]),
        .fc_in (c[0]),
        .fsum  (sum[1]),
        .fc_out (c[1])
    );

    FullAdder FA2(
        .fa    (a[2]),
        .fb    (m^b[2]),
        .fc_in (c[1]),
        .fsum  (sum[2]),
        .fc_out (c[2])
    );

    FullAdder FA3(
        .fa    (a[3]),
        .fb    (m^b[3]),
        .fc_in (c[2]),
        .fsum  (sum[3]),
        .fc_out (c_out)
    );
endmodule
```

[[Xilinx/Altera]\Verilog\DigitalDesign\nBitAddSubNameRef]

위 [코드 nBitAddSubNameRef]에서 FA0 ~ FA3은 전가산기의 동작을 표현한 sub-module FullAdder를 객체화(instantiation)한 것이며, nBitAddSub0의 각 포트는 생성된 객체의 각 포트에 연결이 된다. 즉, FA0에서 sub-module 포트인 fa는 nBitAddSub0 module의 a(0), fb는 m과 b(0)의 xor 연산 결과, fc\_in은 m, fsum은 sum(0), 그리고 fc\_out은 c(0)에 각각 연결된다. 나머지 FA1 ~ FA3 객체도 같은 방법으로 연결된다.

[코드 nBitAddSubNameRef]는 referenced by name 즉, 포트 이름을 mapping함으로써 설계되는 방식이며, 포트의 순서에 의해서 mapping을 할 수도 있다. [코드 nBitAddSubPositionRef]는 포트 순서에 의해 mapping하는 설계 방법을 보여준다.

[코드 nBitAddSubPositionRef]

```
module nBitAddSub0(a, b, m, c_out, sum);
    input [3:0] a, b;
    input      m;
    output     c_out;
    output [3:0] sum;

    wire [2:0] c;

    // referenced by position
    FullAdder FA0(a[0], m^b[0], m, sum[0], c[0]);
    FullAdder FA1(a[1], m^b[1], c[0], sum[1], c[1]);
    FullAdder FA2(a[2], m^b[2], c[1], sum[2], c[2]);
    FullAdder FA3(a[3], m^b[3], c[2], sum[3], c[3]);
endmodule
```

[[Xilinx/Altera]\Verilog\DigitalDesign\nBitAddSubPositionRef]

sub-module FullAdder는 이미 앞에서 설계 되었으며 아래 [코드]와 같다.

[코드 FullAdder]

```
module FullAdder(fa, fb, fc_in, fsum, fc_out);
    input      fa, fb, fc_in;
    output     fsum, fc_out;

    assign fsum=(~fa&~fb&fc_in)|(~fa&fb&~fc_in)|(fa&~fb&~fc_in)|(fa&fb&fc_in);
    assign fc_out=(fa&fb)|(fa&fc_in)|(fb&fc_in);
endmodule
```

[코드 FullAdder] 설계가 가산기로 동작하기 위해서는 M에 '0'을 입력한다. M이 '0'이면 전가산기의 입력  $fb_i$ 에는 입력  $b_i$ 와 M의 exclusive-OR된 결과가 입력되며,  $b_i$ 와 '0'의 exclusive-OR는 항상  $b_i$ 가 되기 때문에 각 전가산기의 결과는  $a_i$ 와  $b_i$ 를 더한 결과가  $S_i$ 에 출력된다. 또한 제일 낮은 자리의 carry는 M 값 '0'이 입력되며, 만일  $a_i$ 와  $b_i$ 를 더한 결과 carry가 발생하면 한 자리 위의 carry 입력인  $z_{i+1}$ 로 입력되어  $a_i$ 와  $b_i$ 와 함께 더해지므로, 두 개의 4비트 입력 a와 b에 대한 전가산기 결과가 출력된다.

M이 '1'이면  $b_i$ 와 '1'의 exclusive-OR는 항상  $b_i$ 에 대한 1의 보수가 되기 때문에  $y_i$ 에는 항상  $b_i$ 에 대한 1의 보수가 입력된다. 또한 가장 낮은 자리의 carry에는 M 값 '1'이 입력되므로 회로의 출력은  $a+(b의\ 1의\ 보수)+1$ 이 된다. 그런데 (b의 1의 보수+1)는 b의 2의 보수이므로,  $a+(b'+1)=a-b$ , 즉 전감산기와 같이 동작한다. 예를 들어서  $a=0110_{(2)}$ ,  $b=1010_{(2)}$ 을 입력하고 M에 '1'을 입력하면, 전가산기의  $y$ 에는 b의 2의 보수인  $0110_{(2)}$ 이 입력되고, 결과는  $1100_{(2)}$ 이 된다. 이 결과는 6에서 10를 뺀 결과 -4를 2의 보수로 나타낸 이진 값이다.

(2) Port mapping에 의한 n비트 가산/감산기 설계(VHDL)

앞에서 전가산기와 전감산기를 HDL(verilog/VHDL)로 설계했다. 그러나 앞에서 설계한 내용은 한 비트에 대한 설계이며, 실제로 가산기와 감산기는 한 비트 이상의 입력에 대한 연산이 이루어져야 한다. 따라서 여기서는 다수의 비트 입력을 가진 가산기/감산기를 HDL에서 port mapping 방식에 의해 설계한다.



n비트 가산/감산기에서는 앞에서 이미 설계한 전가산기를 이용하여 4개의 객체(instance)를 생성한 후에 생성된 객체들 간에 포트를 서로 연결함으로써 설계한다. 이 회로의 기본적인 구성은 4비트 입력 a와 b가 피연산자이고, 입력 M이 '0'이면 가산기 동작을 실행하며 '1'이면 감산기 동작을 실행한다. 가산기와 감산기는 조합논리회로에서 가장 기본이 되는 회로지만, 뺄셈 동작에서 음수 결과가 나올 때를 고려하면 단순하지 않다.

[코드 nBitAddSubNameRef]

```
library ieee;
use ieee.std_logic_1164.ALL;

entity nBitAddSub0 is
port(  a, b          : in std_logic_vector(3 downto 0);
      m            : in std_logic;
      s            : out std_logic_vector(3 downto 0);
      carry        : out std_logic);
end nBitAddSub0;

architecture Behavioral of nBitAddSub0 is
  component FullAdder
    port(  x, y, z      : in std_logic;
          sum, c_out   : out std_logic);
  end component;

  signal c            : std_logic_vector(2 downto 0);
  signal sum          : std_logic_vector(3 downto 0);
  signal c_out        : std_logic;
  signal xor_out      : std_logic_vector(3 downto 0);
  signal clk100Hz     : std_logic;
  signal mxorb        : std_logic_vector(3 downto 0);

begin
  process(m, b)
  begin
    for j in b'range loop
      mxorb(j) <= m xor b(j);
    end loop;
  end process;

  FA0 : FullAdder port map (x => a(0), y => mxorb(0), z => m, sum => s(0),
    c_out => c(0));
  FA12 : for i in 1 to 2 generate
    FA1to2: FullAdder port map(x => a(i), y => mxorb(i), z => c(i-1),
      sum => s(i), c_out => c(i));
  end generate;
  FA3 : FullAdder port map(x => a(3), y => mxorb(3), z => c(2),
    sum => s(3), c_out => carry);
end Behavioral;
```

[[Xilinx/Altera]\VHDL\DigitalDesign\nBitAddSubNameRef]

위 [코드 nBitAddSubNameRef]에서 FA0 ~ FA3은 전가산기의 동작을 표현한 sub-module FullAdder를 객체화(instantiation)한 것이며, nBitAddSub0의 각 포트는 생성된 객체의 각 포트에 연결이 된다. 즉, FA0에서 sub-module 포트인 fa는 nBitAddSub0 module의 a(0), fb는 m과 b(0)의 xor 연산 결과, fc\_in은 m, fsum은 sum(0), 그리고 fc\_out은 c(0)에 각각 연결된다. 나머지 FA1 ~ FA3 객체도 같은 방법으로 연결된다. 우선 4비트 입력 b와 m을 exclusive-OR 연산을 하기 위해서 for 문장을 이용하여 4비트 mxorb 신호를 생성한 후에 port mapping에 사용하였다.

[코드 nBitAddSubNameRef]는 referenced by name 즉, 포트 이름을 mapping함으로써 설계되는 방식이며, 포트의 순서에 의해서 mapping을 할 수도 있다. [코드 nBitAddSubPositionRef]는 포트 순서에 의해 mapping하는 설계 방법을 보여준다.

[코드 nBitAddSubPositionRef]

```
library ieee;
use ieee.std_logic_1164.ALL;

entity nBitAddSub0 is
port(  a, b      : in std_logic_vector(3 downto 0);
      m        : in std_logic;
      s        : out std_logic_vector(3 downto 0);
      carry    : out std_logic);
end nBitAddSub0;

architecture Behavioral of nBitAddSub0 is
  component FullAdder
    port(  x, y, z      : in std_logic;
          sum, c_out   : out std_logic);
  end component;

  signal c          : std_logic_vector(2 downto 0);
  signal sum        : std_logic_vector(3 downto 0);
  signal c_out      : std_logic;
  signal xor_out    : std_logic_vector(3 downto 0);
  signal clk100Hz   : std_logic;
  signal mxorb      : std_logic_vector(3 downto 0);

begin

  FA0 : FullAdder
    port map (a(0), m xor b(0), m, s(0), c(0) );
  FA12 : for i in 1 to 2 generate
    FA1to2: FullAdder port map(a(i), m xor b(i), c(i-1), s(i), c(i));
  end generate;
  FA3 : FullAdder port map(a(3), m xor b(3), c(2), s(3), carry);
end Behavioral;
```

[[Xilinx/Altera]\VHDL\DigitalDesign\nBitAddSubPositionRef]

sub-module FullAdder는 이미 앞에서 설계 되었으며 아래 [코드]와 같다.

[코드 FullAdder]

```
architecture designFA of FullAdder is
begin
  sum <= (not x and not y and z) or (not x and y and not z)
        or (x and not y and not z) or (x and y and z);
  c_out <= (x and y) or (x and z) or (y and z);
end designFA;
```

[코드 FullAdder] 설계가 가산기로 동작하기 위해서는 M에 '0'을 입력한다. M이 '0'이면 전가산기의 입력  $fb_i$ 에는 입력  $b_i$ 와 M의 exclusive-OR된 결과가 입력되며,  $b_i$ 와 '0'의 exclusive-OR는 항상  $b_i$ 가 되기 때문에 각 전가산기의 결과는  $a_i$ 와  $b_i$ 를 더한 결과가  $S_i$ 에 출력된다. 또한 제일 낮은 자리의 carry는 M 값 '0'이 입력되며, 만일  $a_i$ 와  $b_i$ 를 더한 결과 carry가 발생하면 한 자리 위의 carry 입력인  $z_{i+1}$ 로 입력되어  $a_i$ 와  $b_i$ 와 함께 더해지므로, 두 개의 4비트 입력 a와 b에 대한 전가산기 결과가 출력된다.

M이 '1'이면  $b_i$ 와 '1'의 exclusive-OR는 항상  $b_i$ 에 대한 1의 보수가 되기 때문에  $y_i$ 에는 항상  $b_i$ 에 대한 1의 보수가 입력된다. 또한 가장 낮은 자리의 carry에는 M 값 '1'이 입력되므로 회로의 출력은  $a + ((b \text{의 } 1 \text{의 보수}) + '1')$ 이 된다. 그런데 (b의 1의 보수+'1')는 b의 2의 보수이므로

로,  $a+(b'+1)=a-b$ , 즉 전감산기와 같이 동작한다. 예를 들어서  $a=0110_{(2)}$ ,  $b=1010_{(2)}$ 을 입력하고 M에 '1'을 입력하면, 전가산기의 y에는 b의 2의 보수인  $0110_{(2)}$ 이 입력되고, 결과는  $1100_{(2)}$ 이 된다. 이 결과는 6에서 10를 뺀 결과 -4를 2의 보수로 나타낸 이진 값이다.

## IV. DIGCOM-XA1.0 핀 할당

- 푸시버튼 스위치(0x0 ~ 0xf)의 핀 할당

0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7
B15	B16	C16	D14	C14	A15	H13	P15
0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
N14	P13	T15	G12	R10	T10	N9	R8

- 푸시버튼 스위치(RESET, FGO, REG, STEP) 핀 할당

RESET	FGO	REG	STEP
P4	P5	R5	P6

- 슬라이드 스위치(SW0 ~ SW7) 핀 할당

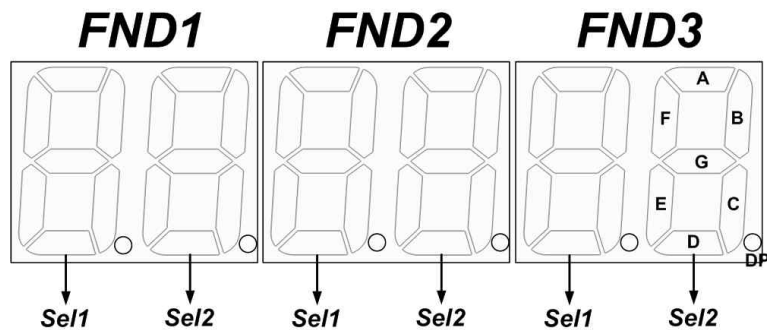
SW0	SW1	SW2	SW3	SW4	SW5	SW6	SW7
J3	K2	K3	L2	M2	N3	N2	R2

- 슬라이드 스위치(M0, M1) 핀 할당

M0	M1
R3	P3

- 7-세그먼트(FND3 ~ FND1) 핀 할당

FND3	FND3A	FND3B	FND3C	FND3D	FND3E	FND3F	FND3G	FND3DP	FND3Sel2	FND3Sel1
	D5	D11	B12	F15	D6	E6	D9	C11	C7	B7
FND2	FND2A	FND2B	FND2C	FND2D	FND2E	FND2F	FND2G	FND2DP	FND2Sel2	FND2Sel1
	D1	D4	C1	B1	B4	C3	C2	R2	H11	G11
FND1	FND1A	FND1B	FND1C	FND1D	FND1E	FND1F	FND1G	FND1DP	FND1Sel2	FND1Sel1
	D3	E2	E3	F2	C6	B6	B5	C4	G14	F13



[그림] 2자리 7-세그먼트 FND 핀 배열

■ LED(D1 ~ D7) 핀 할당

D1	D2	D3	D4	D5	D6	D7
C13	A12	C12	B11	B10	D10	B9

■ LED(D8 ~ D15) 핀 할당

D8	D9	D10	D11	D12	D13	D14	D15
C9	D8	C8	A8	A13	D13	A14	B14

■ 확장 포트 핀 할당(IO0 ~IO16) 핀 할당

IO0	IO1	IO2	IO3	IO4	IO5	IO6	IO7	IO8
R7	R6	N6	P8	P9	M6	P14	R15	H12
IO9	IO10	IO11	IO12	IO13	IO14	IO15	IO16	
M14	H14	H16	G15	F14	E13	E15	D15	

■ Oscillator : E12

OSC
E12