

(기계로 한국어 번역)

Getting started with the FPGA demo bundle for AMD

Xillybus Ltd.
www.xillybus.com

Version 3.4

이 문서는 영어에서 컴퓨터에 의해 자동으로 번역되었으므로 언어가 불분명할 수 있습니다. 이 문서는 원본에 비해 약간 오래되었을 수 있습니다.

가능하면 영문 문서를 참고하시기 바랍니다.

This document has been automatically translated from English by a computer, which may result in unclear language. This document may also be slightly outdated in relation to the original.

If possible, please refer to the document in English.

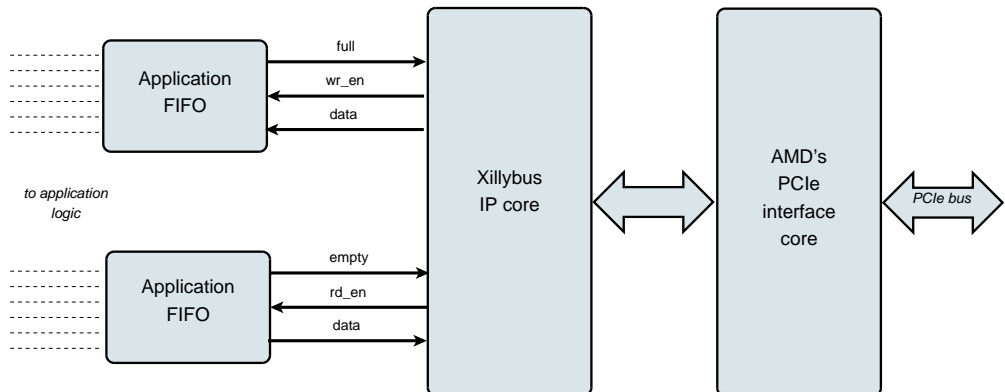
1 소개	4
2 전제 조건	6
2.1 하드웨어	6
2.2 FPGA 프로젝트	6
2.3 개발 소프트웨어	7
2.4 FPGA design 사용 경험	8
3 demo bundle의 implementation	9
3.1 개요	9
3.2 파일 개요	10
3.3 Vivado로 bitstream 파일 생성하기	11
3.4 AMD의 PCIe IP core 설정	13
3.5 ISE 제품군으로 비트 파일 생성	13
3.6 bitfile 로드	15
4 수정	16
4.1 맞춤형 logic와 통합	16
4.2 사용자 지정 프로젝트에 포함	17
4.3 다른 보드 사용	18
4.3.1 일반적인	18
4.3.2 PCIe에 Xillybus 사용	18
4.3.3 Spartan-6 PCIe 보드 작업	18
4.3.4 Virtex-6 PCIe 보드 작업	19
4.3.5 Virtex-5 PCIe 보드 작업	19
4.3.6 Kintex-7, Virtex-7 및 Artix-7 보드 작업(PCIe)	20
4.3.7 Ultrascale 및 Ultrascale+ 보드 작업(PCIe)	21
4.3.8 Versal ACAP 보드 작업(PCIe)	21
4.3.9 XillyUSB로 작업하기	23
4.4 PCIe lanes의 수를 나타내는 PRSNT 핀	23
4.5 PCIe lanes 및/또는 link speed의 번호 변경	24

4.5.1	소개	24
4.5.2	작업 절차	24
4.5.3	PIPE 주파수가 변경되었습니까?	26
4.5.4	timing constraints 적용	28
4.5.5	PIPE clock 모듈 업데이트	29
4.6	FPGA 부품 번호 변경	30
5	문제 해결	31
5.1	implementation 중 오류	31
5.2	PCIe 하드웨어 문제	31

1

소개

Xillybus는 FPGA와 Linux 또는 Microsoft Windows를 실행하는 host 간의 데이터 전송을 위한 DMA 기반 종단 간 솔루션입니다. FPGA logic 디자이너와 소프트웨어 프로그래머에게 간단하고 직관적인 인터페이스를 제공합니다.



위에 표시된 것처럼 FPGA의 application logic은 표준 FIFOs와만 상호 작용하면 됩니다. 예를 들어, 다이어그램에서 하위 FIFO에 데이터를 쓰는 것은 Xillybus IP core가 FIFO의 다른 쪽 끝에서 데이터를 전송할 수 있음을 감지하게 합니다. 곧 IP core는 FIFO에서 데이터를 읽고 host로 보내 userspace software에서 읽을 수 있도록 합니다. 데이터 전송 메커니즘은 FIFO와만 상호 작용하는 FPGA의 application logic에 투명합니다.

다른 한편으로 Xillybus IP core는 PCI Express의 Transport Layer level을 활용하여 데이터 흐름을 구현하여 TLP 패킷을 생성 및 수신합니다. 하위 계층의 경우 개발 도구의 일부인 AMD의 공식 PCIe core에 의존하며 추가 라이선스가 필요하지 않습니다(WebPACK에 디션을 사용하는 경우에도).

컴퓨터의 응용 프로그램은 named pipes처럼 동작하는 device files와 상호 작용합니다. Xillybus IP core 및 driver는 FPGA의 FIFOs와 host의 관련 device files 간에 데이터를

효율적이고 직관적으로 전송합니다.

XillyUSB에서 MGT transceiver는 위에서 언급한 PCIe 인터페이스 대신 데이터 전송에 사용되는 USB 3.0 인터페이스를 구현하는 데 사용됩니다.

IP core는 온라인 웹 애플리케이션을 사용하여 고객의 사양에 따라 즉시 구축됩니다. streams의 수, 방향 및 기타 속성은 design의 대역폭 성능, 동기화 및 단순성 간의 최적 균형을 달성하기 위해 고객이 정의합니다. 이 가이드에 설명된 대로 demo bundle의 준비 단계를 거친 후 <https://xillybus.com/custom-ip-factory>에서 사용자 지정 IP core를 빌드하고 다운로드하는 것이 좋습니다.

이 가이드는 실제 애플리케이션 시나리오 테스트를 위해 사용자 제공 데이터 소스 및 데이터 소비자에 연결할 수 있는 Xillybus IP core와 함께 FPGA를 빠르게 설정하는 방법을 설명합니다. IP core는 웹사이트에서 다운로드할 수 있는 demo bundle에 포함되어 있습니다.

이름에도 불구하고 demo bundle은 데모 키트가 아니라 유용한 작업을 있는 그대로 수행할 수 있는 완전한 기능의 starter design입니다.

궁금한 분들을 위해 Xillybus 구현 방법에 대한 간략한 설명은 [Xillybus host application programming guide for Linux](#) 또는 [Xillybus host application programming guide for Windows](#)의 부록 A에서 찾을 수 있습니다.

2

전제 조건

2.1 하드웨어

Xillybus FPGA demo bundle은 다운로드 페이지에 나열된 대로 여러 보드 및 장치와 함께 작동하도록 패키징되어 있습니다(아래 2.2 섹션 참조).

다른 보드의 소유자는 핀 배치에 필요한 변경을 수행하고 MGT의 reference clock이 제대로 처리되는지 확인한 후 자신의 하드웨어에서 demo bundle을 실행할 수 있습니다. 이것은 숙련된 FPGA 엔지니어라면 누구나 쉽게 이해할 수 있습니다. 이에 대한 자세한 내용은 4.3 섹션을 참조하십시오.

2.2 FPGA 프로젝트

Xillybus demo bundle은 Xillybus 사이트의 다운로드 페이지에서 다운로드할 수 있습니다. PCIe 기반 cores의 경우:

<https://xillybus.com/pcie-download>

그리고 XillyUSB의 경우:

<https://xillybus.com/usb-download>

demo bundle에는 간단한 테스트를 위한 Xillybus IP core의 특정 구성이 포함되어 있습니다. 따라서 특정 응용 프로그램에 대해서는 상대적으로 성능이 좋지 않습니다.

사용자 지정 IP cores는 IP Core Factory 웹 애플리케이션을 사용하여 구성, 자동 구축 및 다운로드할 수 있습니다. 이 도구를 사용하려면 <https://xillybus.com/custom-ip-factory>를 방문하십시오.

Xillybus IP core를 포함하여 다운로드한 번들은 “evaluation”라는 용어와 합리적으로 일치하는 한 무료로 사용할 수 있습니다. 여기에는 최종 사용자 designs에 core 통합, 실제 데

이더 실행 및 현장 테스트가 포함됩니다. core 사용의 유일한 목적이 특정 애플리케이션에 대한 기능과 적합성을 평가하는 것이라면 core 사용 방법에는 제한이 없습니다.

2.3 개발 소프트웨어

FPGA 제품군에 따라 Xillybus의 demo bundle(및 Xillybus와 관련된 다른 designs)의 implementation에 권장되는 도구가 아래에 나열되어 있습니다.

PCIe용 Xillybus :

현재로서는 컴퓨터에 설치된 Vivado 버전이 PCIe용 Xillybus와 함께 작동하기에 적합하다는 것이 거의 확실합니다. 그럼에도 불구하고 더 자세한 요구 사항은 다음과 같습니다.

- Virtex-5 FPGAs를 사용할 때 Xilinx ISE 13.1 버전이 선호됩니다(단락 4.3.5 참조).
- Spartan-6 및 Virtex-6의 경우 Xilinx ISE 13.2 이상을 사용하십시오.
- Gen2 인터페이스가 있는 Kintex-7 및 Virtex-7(XT/HT가 아닌 모든 Virtex-7 및 485T도 포함)의 경우 Vivado 2014.1 이상이 선호되는 도구입니다. ISE 개정판 중 버전 14.2 이상을 권장합니다.
- Gen3 인터페이스가 있는 Virtex-7의 경우(485T를 제외한 XT/HT), Vivado 2014.1 이상이 선호됩니다. ISE를 선택한 경우 수정 버전 14.6 이상이 필요합니다.
- Artix-7의 경우 Vivado 2014.1 이상을 사용해야 합니다. ISE 14.6 이상도 괜찮습니다.
- Kintex / Virtex Ultrascale의 경우 Vivado 2015.2 이상을 사용해야 합니다. ISE 개정판은 이러한 장치를 지원하지 않습니다.
- Ultrascale+ FPGAs에는 Vivado 2017.3 이상이 필요합니다.
- Versal APAC FPGAs에는 Vivado 2021.2 이상이 필요합니다.

XillyUSB :

- Ultrascale+를 제외한 모든 FPGAs에는 Vivado 2015.2 이상을 사용해야 합니다.
- Ultrascale+의 경우 Vivado 2018.3 이상을 사용해야 합니다.

이 소프트웨어는 AMD의 웹사이트(<https://www.amd.com>)에서 직접 다운로드할 수 있습니다.

이 소프트웨어의 모든 버전이 적합합니다. FPGA가 WebPACK Edition의 적용을 받는 경우 이 버전은 다운로드하여 라이선스 비용 없이 무제한 사용할 수 있으므로 선호하는 선택이 될 수 있습니다.

Xillybus의 implementation은 AMD에서 제공하는 일부 IP cores에 의존합니다. 모든 소프트웨어 에디션은 추가 라이선스 없이 이러한 IP cores를 지원합니다.

2.4 FPGA design 사용 경험

design이 demo bundles 목록에 표시되는 보드용인 경우 FPGA design에 대한 이전 경험이 없어도 FPGA에서 demo bundle을 작동할 수 있습니다. 다른 보드를 사용하는 경우 AMD의 도구 사용, 특히 핀 배치 및 clocks 정의에 대한 지식이 필요합니다.

demo bundle을 최대한 활용하려면 logic design 기술을 잘 이해하고 HDL 언어(Verilog 또는 VHDL)를 마스터해야 합니다. 그럼에도 불구하고 Xillybus demo bundle은 실험할 간단한 starter design을 제공하므로 이러한 학습을 위한 좋은 출발점입니다.

3

demo bundle의 implementation

3.1 개요

아래의 모든 내용을 읽지 않으려는 Vivado 사용자의 경우 첫 번째 implementation을 실행하기 위한 단계는 다음과 같습니다.

- 작업 디렉토리에 demo bundle의 압축을 풉니다.
- Vivado를 시작하거나 Vivado가 이미 실행 중인 경우 열려 있는 프로젝트를 닫습니다.
- Tools > Run Tcl Script...를 선택하고 verilog/ 또는 vhdl/(demo bundle 내부)에서 **xillydemo-vivado.tcl**을 선택합니다.
- “Generate Bitstream”(또는 “Generate Device Image”)를 클릭합니다.
- 성공적인 implementation 후 vivado/xillydemo.runs/impl_1/에서 bitstream 파일을 찾습니다.

이제 더 긴 이야기로 넘어가겠습니다. Xillybus의 demo bundle의 implementation에는 세 가지 가능한 방법이 있으며 bit stream 파일을 얻는 방법은 다음과 같습니다.

- 번들에 있는 프로젝트 파일을 그대로 사용합니다. 이것은 가장 간단한 방법이며 ML506 (Virtex-5)를 제외하고 demo bundles 목록에 나타나는 보드로 작업할 때 적합합니다.
- 다른 FPGA와 일치하도록 파일 수정. 이것은 다른 보드 및/또는 다른 FPGAs와 함께 작업할 때 적합합니다. 이것은 Virtex-5로 작업할 때도 필요합니다. 이에 대한 자세한 정보는 4.3 단락에서 확인하십시오.

- Vivado(또는 ISE) 프로젝트를 처음부터 설정합니다. demo bundle을 기존 application logic와 통합할 때 필요할 수 있습니다. 4.2 단락에 자세한 내용이 있습니다.

이 섹션의 나머지 부분에서는 첫 번째 작업 절차가 자세히 설명되어 있으며 가장 간단하고 가장 일반적으로 선택됩니다. 다른 두 가지 작업 절차는 첫 번째 절차를 기반으로 하며 차이점은 위에 제공된 단락에 자세히 설명되어 있습니다.

중요한:

평가 번들은 성능보다는 단순성을 위해 구성됩니다. 지속적이고 지속적인 데이터 흐름이 필요한 애플리케이션, 특히 고대역폭의 경우 훨씬 더 나은 결과를 얻을 수 있습니다. 이러한 시나리오의 경우 사용자 지정 IP core는 웹 응용 프로그램과 함께 쉽게 구축 및 다운로드됩니다.

3.2 파일 개요

번들은 다음 디렉터리 중 일부로 구성됩니다(어떤 디렉터리가 있는지는 의도한 FPGA에 따라 다름).

- core– Xillybus IP core는 여기에 저장됩니다.
- instantiation templates– core용 instantiation templates 포함(Verilog 및 VHDL)
- verilog– demo bundle에 대한 프로젝트 파일과 Verilog의 소스를 포함합니다('src' 하위 디렉토리에 있음).
- vhdl– demo bundle에 대한 프로젝트 파일과 VHDL의 소스를 포함합니다('src' 하위 디렉토리에 있음).
- vivado-essentials– Vivado에서 사용하기 위한 logic용 정의 파일 및 빌드 디렉토리.
- blockplus– 이 디렉토리는 Virtex-5에만 관련됩니다. 단락 4.3.5를 참조하십시오.

각 demo bundle은 demo bundle을 다운로드한 사이트의 웹 페이지에 나열된 대로 특정 보드용입니다. 다른 보드를 사용하거나 특정 구성 저항이 보드에서 추가 또는 제거된 경우 constraints 파일을 그에 따라 편집해야 합니다.

Vivado 프로젝트의 경우 이 파일은 vivado-essentials/xillydemo.xdc이고 ISE 프로젝트의 경우 verilog/ 또는 vhdl/ 아래의 선택한 'src' 디렉토리에 있는 UCF 파일입니다.

또한 vhdl 디렉토리에는 Verilog 파일이 포함되어 있지만 크게 변경할 필요는 없습니다.

Xillybus의 IP core와 application logic 사이의 인터페이스는 xillydemo.v 파일 또는 xillydemo.vhd 파일(해당 'src' 하위 디렉토리)에서 이루어집니다. 자신의 데이터로 Xillybus를 사용하기 위해 편집하는 파일입니다.

3.3 Vivado로 bitstream 파일 생성하기

ISE 사용자: [3.4](#) 단락으로 건너뛰십시오.

Vivado는 비교적 복잡한 구조에서 많은 중간 파일을 생성하므로 프로젝트를 제어하기가 어렵습니다. 번들의 파일 구조를 컴팩트하게 유지하기 위해 Vivado 프로젝트 생성을 위해 Tcl의 script가 제공됩니다. 이 script는 새 하위 디렉토리 "vivado"를 만들고 필요에 따라 이 디렉토리를 파일로 채웁니다.

프로젝트는 src/ 하위 디렉토리의 파일에 의존합니다(이러한 파일의 복사본이 만들어지지 않음). PCIe 블록과 logic에서 사용하는 FIFOs는 vivado-essentials/에 정의되어 있습니다. Vivado는 또한 프로젝트의 implementation 중에 이 디렉토리를 중간 파일로 채웁니다.

Vivado를 시작하세요. 프로젝트가 열려 있지 않은 상태에서 Tools > Run Tcl Script...를 선택하고 기본 설정에 따라 verilog/ 또는 vhd/ 하위 디렉터리에서 **xillydemo-vivado.tcl**을 선택합니다. 1분 이내에 일련의 이벤트가 발생합니다. 프로젝트 배포의 성공 여부는 Vivado 창 하단에 있는 "Tcl Console" 탭을 선택하고 다음과 같이 표시되는지 확인할 수 있습니다.

```
INFO: Project created: xillydemo
```

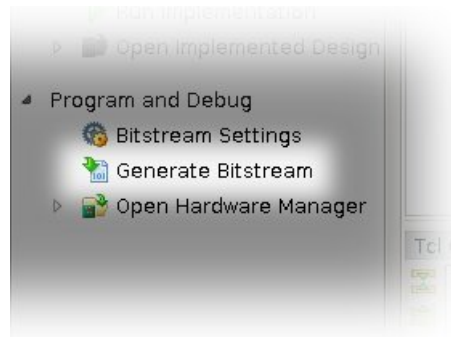
이것이 Tcl console의 마지막 줄이 아닌 경우 문제가 발생한 것입니다. 대부분 잘못된 버전의 Vivado가 사용되었기 때문입니다.

이 단계에서 Warnings가 나타나지만 오류는 없습니다. 그러나 프로젝트가 이미 생성된 경우(즉, script가 이미 실행된 경우) script를 다시 실행하려고 하면 다음 오류가 발생합니다.

```
ERROR: [Common 17-53] User Exception: Project already exists on disk,
please use '-force' option to overwrite:
```

새로운 "vivado" 하위 디렉터리는 Tcl script가 포함된 디렉터리에 생성된다는 점에 유의하십시오.

프로젝트가 생성된 후 implementation을 시작합니다. 왼쪽의 Flow Navigator 막대에서 "Generate Bitstream"(또는 "Generate Device Image")를 클릭합니다.



synthesis 및 implementation을 시작해도 되는지 묻는 팝업 창이 나타날 가능성이 높습니다. “Yes”를 선택합니다.

Vivado는 일련의 프로세스를 실행합니다. 일반적으로 몇 분 정도 걸립니다. 몇 개의 warnings가 발행되고 그 중 어느 것도 중요로 분류되지 않습니다(그러나 일부 critical warnings는 Tcl script 실행의 로그에 여전히 남아 있을 수 있음).

bitstream의 implementation이 성공적으로 완료되었다는 팝업 창이 나타납니다. 다음에 무엇을 할 것인지에 대한 선택권을 줄 것입니다. “Cancel”을 포함한 모든 옵션이 좋습니다.

bitstream 파일 xillydemo.bit은 vivado/xillydemo.runs/impl_1/에서 찾을 수 있습니다. Versal FPGAs의 경우 파일은 대신 xillydemo.pdi입니다.

implementation은 결코 실패할 것으로 예상되지 않습니다. 그러나 언급할 가치가 있는 한 가지 오류 조건이 있습니다.

사용자 정의 logic이 통합된 경우 “Timing constraints weren’t met”이 발생할 수 있다는 오류가 표시됩니다. 이는 도구가 timing에 대한 요구 사항을 달성하지 못했음을 의미합니다. 이 경우 design은 구문상 올바르지만 주어진 clock 속도 및/또는 I/O 요구 사항과 관련하여 특정 경로를 충분히 빠르게 만들려면 수정이 필요합니다. 더 나은 timing을 위해 design을 수정하는 프로세스를 종종 *timing closure*라고 합니다.

timing constraint 오류는 일반적으로 critical warning로 발표되므로 Vivado는 사용자가 FPGA의 안정적인 동작을 보장하지 않는 bitstream 파일을 생성하는 것을 막지 않습니다. 이러한 bitstream 생성을 방지하기 위해 timing 오류는 작은 Tcl script, 즉 route 실행 종료 시 자동으로 실행되는 “showstopper.tcl” 덕분에 오류로 전환됩니다. 이 안전 조치를 해제하려면 Flow Navigator의 “Project Manager”에서 “Project Settings”를 클릭하십시오. “Implementation” 버튼을 선택하고 “route_design” 설정까지 아래로 스크롤합니다. 그런 다음 tcl.post에서 showstopper.tcl을 제거합니다.

Vivado 사용자는 다음 섹션을 건너뛰고 3.6 단락으로 바로 이동할 수 있습니다.

3.4 AMD의 PCIe IP core 설정

이 부분은 Spartan-6 이외의 FPGAs용 ISE 툴체인에만 관련됩니다. Vivado를 사용하는 경우 3.3 단락을 참조하십시오. Spartan-6 FPGAs로 작업하는 사람들은 3.5 단락으로 바로 이동할 수 있습니다.

PCIe용 Coregen IP core의 다소 특이한 조직은 implementation 프로젝트에 XCO 파일을 포함하는 것을 허용하지 않지만 대신 core 생성 소프트웨어가 포함할 Verilog 파일을 생성합니다. 이것은 Virtex-5, Virtex-6 및 series-7로 작업할 때만 해당됩니다.

Virtex-5 FPGAs는 XCO 파일을 특수하게 처리해야 합니다. 단락 4.3.5를 참조하십시오.

blockplus 디렉토리 또는 pcie_core 디렉토리(번들에 있음)에서 프로젝트 파일(.xise)을 찾아 더블 클릭하여 ISE를 여십시오. "Design Utilities"에서 "Regenerate all cores"를 클릭하고 프로세스가 완료될 때까지 기다립니다. 그런 다음 ISE를 닫습니다. 추가 조치가 필요하지 않습니다.

이 절차는 AMD PCIe core용 래퍼인 Verilog 파일 세트를 생성합니다. 이 파일은 demo bundle의 bitstream을 생성하는 프로젝트에서 사용됩니다. 파일은 Verilog 또는 VHDL을 기본 언어로 선택했는지에 관계없이 Verilog로 생성됩니다.

이러한 Verilog 파일을 기본 프로젝트에 수동으로 포함할 필요는 없지만 전체 프로젝트의 implementation을 시도하기 전에 이러한 파일을 한 번 생성해야 합니다. 메인 프로젝트의 implementation 반복 이전이 아니더라도 이 절차를 반복할 필요가 없습니다.

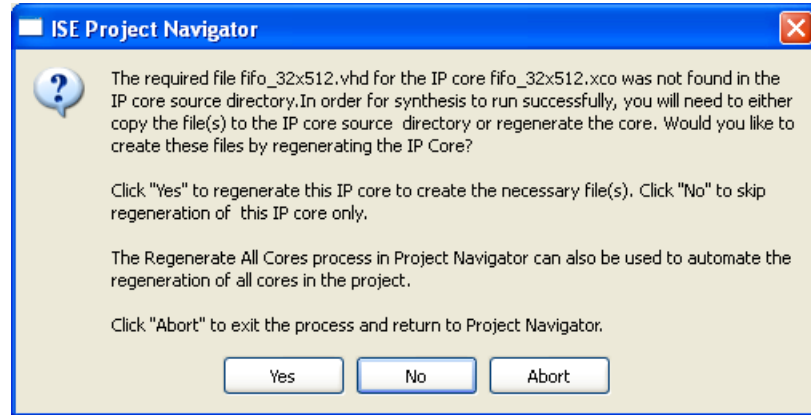
3.5 ISE 제품군으로 비트 파일 생성

Virtex 또는 series-7 제품군에 속하는 FPGA로 작업할 때 위의 3.4 단락에 설명된 대로 PCIe 래퍼를 준비했는지 확인하십시오.

기본 설정에 따라 'verilog' 하위 디렉터리 또는 'vhdl' 하위 디렉터리에서 'xillydemo.xise' 파일을 두 번 클릭합니다. ISE가 실행을 시작하고 올바른 설정으로 프로젝트를 엽니다. ISE는 파일이 누락되었다고 불평해서는 안 됩니다. 그렇다면 FPGA 제품군이 series-7 또는 Virtex 제품군 중 하나인 경우 PCIe 래퍼가 3.4 단락에 언급된 대로 준비되지 않았을 수 있습니다.

"Generate Programming File"을 클릭하여 implementation을 시작합니다.

첫 번째 implementation 동안 2개 또는 3개의 AMD Coregen IP cores를 재생성해야 합니다. 이 프로세스는 시간이 많이 걸리지만 다행히 한 번만 수행됩니다. 다음과 같은 팝업 창이 두세 번 나타납니다.



이 모든 항목에서 “Yes”를 클릭합니다.

이 절차는 여러 warnings를 생성하지만(FPGA 프로젝트의 implementation은 항상 수행함) 오류를 나타내지 않아야 합니다. 프로세스가 완료되면 bitfile을 xillydemo.bit로 찾을 수 있습니다.

log에서 다음 문장을 찾아 **timing constraints**가 달성되었는지 항상 확인하십시오 .

```
All constraints were met.
```

이것은 특히 프로젝트를 변경한 후에 중요합니다. constraints가 달성되지 않으면 도구는 여전히 bitfile을 생성하지만 FPGA는 예측할 수 없는 방식으로 작동할 수 있습니다(허용된 온도 범위 내의 온도 변화의 결과일 수 있음).

유사한 메시지는 ISE의 design 요약에서 찾을 수 있습니다.

timing constraints를 달성하지 못하는 것은 bus_clk의 속도를 견딜 만큼 빠르지 않은 logic을 추가한 결과일 수 있습니다. 그러나 명백한 이유 없이 오류가 발생하는 경우 AMD의 도구가 FPGA의 logic fabric에 logic 구성 요소를 배치하려고 할 때 초기 추측이 잘못되었고 나중에 실행되는 최적화 알고리즘이 이 문제를 해결할 수 없었을 수 있습니다.

후자의 경우는 placer cost table figure(초기 배치의 임의성에 대한 seed임)를 변경하여 수정할 수 있습니다. ISE Project Navigator 내부의 Processes 창에서 “Map”을 마우스 오른쪽 버튼으로 클릭하고 “Process Properties...”를 선택합니다. Property display level이 “Advanced”인지 확인하고 “Starting Placer Cost Table”을 아직 시도하지 않은 다른 번호로 변경하십시오. 이 숫자의 크기는 의미가 없습니다. 그런 다음 “Generate Programming File”로 다시 시작하십시오.

중요한:

일부 ISE 버전, 특히 ISE 14.2에서는 verilog/ 디렉터리의 빌드가 `ERROR:HDLCompiler:687 - "C:/try/xillybus-eval-kintex7-1.1/verilog/src/fifo_32x512_synth.v" Line 54: Illegal redeclaration of module fifo_32x512.`(또는 유사한 오류)와 함께 실패할 수 있습니다. 이는 AMD 도구의 버그 때문입니다. 이 문제를 해결하려면 src/ 디렉터리에서 `fifo_8x2048_synth.v`와 `fifo_32x512_synth.v` 파일을 삭제하고 "Generate Programming File"을 다시 시작하십시오. warnings 버전 중 일부는 도구가 해당 파일을 찾지 못한다는 오류를 표시할 수 있지만, implementation 버전은 정상적으로 실행될 것입니다.

3.6 bitfile 로드

초기 개발 단계에서는 JTAG을 통해 FPGA를 로드하는 것이 좋습니다. 대부분의 보드에서 Vivado를 실행하는 컴퓨터와 보드 패널의 USB 커넥터 사이에 간단한 USB 케이블이면 충분합니다. ISE로 작업하는 사람들은 bitfile을 로드하기 위해 iMPACT를 사용합니다.

JTAG을 통해 FPGA를 로드하는 방법은 보드의 지침을 참조하십시오.

XillyUSB 프로젝트의 경우 USB 인터페이스가 작동 중인 컴퓨터에 연결되어 있는 동안에도 FPGA를 언제든지 로드 및 다시 로드할 수 있습니다.

PCIe 프로젝트의 경우 컴퓨터의 전원을 켜기 전에 FPGA에 bitfile을 로드해야 합니다. 컴퓨터는 전원이 켜질 때 PCIe 주변기기가 적절한 상태에 있을 것으로 예상하고 이후에 어떤 놀라움도 용납하지 않을 수 있습니다.

따라서 host가 실행 중인 동안에는 FPGA를 다시 로드하지 **마십시오**. PCIe 사양에는 hotplugging에 대한 지원이 필요하지만 마더보드는 일반적으로 PCIe 카드가 사라졌다가 다시 나타날 것으로 예상하지 않습니다. 따라서 일부 마더보드는 제대로 응답하지 않을 수 있습니다. 그럼에도 불구하고 운영 체제가 실행되는 동안 FPGA를 다시 로드하면 일부 마더보드에서 작동합니다.

Xillybus의 driver는 hotplugging에 제대로 응답하도록 설계되었지만 컴퓨터의 일반적인 안정성을 보장하는 것은 없습니다. 이것은 이 페이지에서 논의됩니다.

<https://xillybus.com/doc/hot-reconfiguration>

FPGA의 전원이 켜지고 컴퓨터의 전원을 켜면서 flash memory에서 로드되는 경우 FPGA가 충분히 빨리 로드되어 BIOS가 bus를 스캔할 때 PCIe 장치가 존재하도록 하는 것이 중요합니다.

4

수정

4.1 맞춤형 logic와 통합

Xillybus demo bundle은 application logic와 쉽게 통합되도록 구성되었습니다. 데이터를 연결하는 위치는 xillydemo.v 또는 xillydemo.vhd 파일입니다(선호하는 언어에 따라 다름). 번들의 다른 모든 HDL 파일은 host(Linux 또는 Windows)와 FPGA 간에 데이터를 전송하기 위해 Xillybus IP core를 사용할 목적으로 무시할 수 있습니다.

사용자 지정 logic designs가 있는 추가 HDL 파일은 단락 3.5 또는 3.3에 설명된 대로 준비된 프로젝트에 추가한 다음 “Generate Programming File” 또는 “Generate Bitstream”을 클릭하여 다시 빌드할 수 있습니다. 초기 배포의 다른 단계를 반복할 필요가 없으므로 logic의 개발 주기는 상당히 빠르고 간단합니다.

Xillybus IP core를 사용자 지정 application logic에 연결할 때 FIFOs를 통해서만 Xillybus IP core와 상호 작용하고 logic로 동작을 모방하려고 시도하지 않는 것이 좋습니다. 최소한 첫 번째 단계에서는 아닙니다.

이에 대한 예외는 메모리 또는 register arrays를 Xillybus에 연결할 때이며, 이 경우 xillydemo 모듈에 표시된 예를 따라야 합니다.

xillydemo 모듈에서 FIFOs는 data loopback을 수행하는 데 사용됩니다. 즉, host에서 도착한 데이터가 다시 host로 전송됩니다. FIFO의 양쪽이 모두 Xillybus IP core에 연결되어 있으므로 core는 데이터 소스이자 데이터 소비자입니다.

실제 사용 시나리오에서는 FIFO의 측면 중 하나만 Xillybus IP core에 연결됩니다. FIFO의 다른 쪽은 데이터를 공급하거나 소비하는 application logic에 연결됩니다.

xillydemo 모듈에 사용되는 FIFOs는 양쪽이 Xillybus의 메인 clock에 의해 구동되기 때문에 양쪽에 대해 하나의 공통 clock에서만 작동합니다. 실제 애플리케이션에서는 읽기와 쓰기를 위한 별도의 clocks가 있는 FIFOs로 교체하는 것이 바람직할 수 있습니다. 이를 통해 bus_clk이 아닌 clock로 데이터 소스와 데이터 소비자를 구동할 수 있습니다. 이렇게 함으

로써 FIFOs는 중재자 역할을 할 뿐만 아니라 적절한 clock domain 교차점 역할도 합니다. Xillybus IP core는 FPGA에서 host로의 streams에 대해 일반 FIFO 인터페이스(First Word Fall Through, FWFT와 반대)를 예상합니다. 다음 문서는 맞춤형 logic 통합과 관련이 있습니다.

- logic design용 API: [Xillybus FPGA designer's guide](#)
- [Getting started with Xillybus on a Linux host](#)
- [Getting started with Xillybus on a Windows host](#)
- [Xillybus host application programming guide for Linux](#)
- [Xillybus host application programming guide for Windows](#)
- [The guide to defining a custom Xillybus IP core](#)

4.2 사용자 지정 프로젝트에 포함

원하는 경우 기존 Vivado / ISE 프로젝트에 Xillybus IP core를 포함하거나 처음부터 새 프로젝트를 생성할 수 있습니다.

프로젝트가 이미 존재하지 않는 경우 새 프로젝트를 시작하고 원하는 HDL 언어 및 의도한 FPGA를 기반으로 설정합니다.

Vivado 프로젝트에 Xillybus IP core를 포함하려면 사용자 정의 프로젝트의 소스 파일 및 설정을 반영하도록 xillydemo-vivado.tcl을 편집하고 이 script를 실행하여 새 프로젝트를 생성하는 것이 좋습니다.

ISE 프로젝트에 Xillybus IP core를 포함하려면:

- Virtex-5/6 또는 series-7 제품군에 속하는 FPGAs로 작업할 때 3.4 단락(Virtex-5 FPGAs의 경우 4.3.5)에 자세히 설명된 대로 PCIe 래퍼 파일을 별도로 생성해야 합니다. 생성된 Verilog 파일은 사용자 정의 프로젝트에 추가해야 합니다(XCO 파일은 제외).
- 언어 기본 설정에 따라 두 개의 src/ 하위 디렉토리 중 하나에 있는 모든 파일을 프로젝트에 추가합니다.
- Macro search path에 디렉토리 추가: process 메뉴의 "Implementation"에서 "Translate"를 마우스 오른쪽 버튼으로 클릭하고 "Process Properties..."를 선택합니다. Macro Search Path 속성에 'core' 하위 디렉토리를 추가합니다(맨 오른쪽에 있는 버튼으로 탐색). 이 속성을 설정하지 않으면 xillybus_core.ngc 파일을 찾을 수 없기 때문에 implementation 동안 Translate 단계가 실패합니다.

- xillydemo 모듈이 프로젝트의 top level module이 아닌 경우 해당 포트를 top level에 연결합니다.
- Xillybus IP core를 맞춤형 application logic에 부착하려면 기존 application logic을 원하는 모듈로 교체하여 xillydemo 모듈을 편집하십시오.

4.3 다른 보드 사용

4.3.1 일반적인

demo bundles 목록에 나타나지 않는 보드로 작업할 때 번들에서 약간의 수정이 필요합니다.

core는 몇 가지 GPIO LED 출력을 생성합니다. 이러한 빈 공간이 있으면 보드의 LEDs에 연결하는 것이 좋습니다.

4.3.2 PCIe에 Xillybus 사용

구입한 대부분의 보드에는 PCIe 인터페이스가 해당 보드에서 어떻게 사용되는지 보여주는 자체 FPGA design 예제가 있습니다. 원하는 보드의 XDC / UCF 파일에서 관련 핀 할당을 찾고 핀 이름을 Xillybus의 XDC / UCF 파일에서 사용되는 이름으로 수정하는 것이 가장 쉬운 경우가 많습니다. 그런 다음 Xillybus의 프로젝트에서 사용되는 XDC / UCF 파일의 관련 행을 교체할 수 있습니다.

핀을 배치하는 방법에 대한 자세한 내용은 아래에 나와 있습니다.

가장 흔한 실수는 PCIe bus의 reference clock에서 발생합니다. 동일한 주파수의 clock만 연결하면 작동하지 않습니다. 마더보드의 clock와 다른 clock 사이의 미세한 주파수 차이로 인해 transceiver가 산발적으로 잠금을 잃게 되어 통신이 불안정해지고 FPGA를 PCIe 장치로 감지하지 못할 수 있습니다.

Xillybus core는 AMD의 PCIe core를 기반으로 하므로 AMD의 사용자 가이드는 PCIe의 physical layer와 관련된 고려 사항에 대한 유효한 소스입니다.

4.3.3 Spartan-6 PCIe 보드 작업

Spartan-6의 경우 Xillybus core는 다음과 같이 7개의 물리적 와이어로 구성된 PCIe bus 포트를 통해 host와 인터페이스합니다.

- 이름이 PCIE_250M_P 및 PCIE_250M_N인 reference clock용 differential wires
쌍: PCIe bus clock에서 파생된 125 MHz(net의 이름에도 불구하고)의 주파수를 가

진 clock이 이 전선에서 예상됩니다. 다른 clock이 적용되는 경우 실제 clock 주파수를 예상하도록 AMD PCIe Coregen core(번들에서 `pcie.xco`로 정의됨)를 재구성해야 합니다. 또한 `TS_PCIE_CLK` 사양이 변경 사항을 반영하도록 timing constraint를 업데이트해야 합니다.

- `PCIE_PERST_B_LS`의 host의 master bus reset
- 직렬 데이터 입력 쌍, `PCIE_RX0_P` 및 `PCIE_RX0_N`
- 직렬 데이터 출력 쌍, `PCIE_TX0_P` 및 `PCIE_TX0_N`

이 핀의 할당은 보드의 배선에 따라 설정됩니다.

4.3.4 Virtex-6 PCIe 보드 작업

Virtex-6의 경우 배선은 유사합니다.

- 이름이 `PCIE_REFCLK_P` 및 `PCIE_REFCLK_N`인 reference clock용 differential wires 쌍: PCIe bus clock에서 파생된 250 MHz의 주파수를 가진 clock이 이러한 전선에 예상됩니다. 다른 clock이 적용되는 경우 AMD PCIe Coregen core(번들에서 `pcie_v6_4x.xco`로 정의됨)는 실제 clock 주파수를 예상하도록 재구성되어야 합니다. 이러한 변경에는 constraints의 변경도 포함될 수 있습니다. Coregen에 의해 생성된 예제 UCF 파일을 참조하십시오.
- `PCIE_PERST_B_LS`의 host의 master bus reset
- 직렬 데이터 입력 벡터 쌍, `PCIE_RX_P` 및 `PCIE_RX_N`(각 4선)
- 직렬 데이터 출력 벡터 쌍, `PCIE_TX_P` 및 `PCIE_TX_N`(각 4선)

핀 할당은 transceiver logic을 배치하여 암시적으로 이루어집니다. UCF 파일에서 GTX 배치를 정의하는 constraints는 특정 pinout를 강제 실행합니다. 마찬가지로 reference clock의 핀 배치는 clock buffer(`pcieclk_ibuf`)의 위치를 제한하여 암시적으로 설정됩니다. Xillybus 번들에 있는 UCF 파일에는 안내 설명이 포함되어 있습니다.

UCF 파일은 이들의 핀 배치가 의도한 보드의 핀 배치와 일치하도록 편집되어야 합니다.

4.3.5 Virtex-5 PCIe 보드 작업

Virtex-5 제품군에는 두 그룹의 장치가 있으며 각각 약간 다른 PCIe 인터페이스가 필요합니다. 이를 간단히 처리하기 위해 'blockplus' 하위 디렉토리에 두 개의 다른 XCO 파일이 있으며 그 중 하나만 사용해야 합니다.

따라서 PCIe core를 빌드하기 전에 해당 하위 디렉토리의 파일 이름을 다음과 같이 변경해야 합니다.

- Virtex-5 LX 또는 Virtex-5 SX의 경우: pcie_v5_gtp.xco의 이름을 pcie_v5.xco로 바꿉니다.
- Virtex-5 FX 또는 Virtex-5 TX의 경우: pcie_v5_gtx.xco의 이름을 pcie_v5.xco로 바꿉니다.

중요한:

PCIe Block Plus generator의 버전은 1.14여야 하며 1.15는 절대 아닙니다. ISE 13.1에는 이 목적에 맞는 올바른 버전이 있지만 ISE 13.2와 함께 제공되는 버전은 잘못된 코드를 생성합니다.

전체 implementation에 대해 ISE 13.1 이외의 버전이 필요한 경우 올바른 버전의 PCIe Block Plus(ISE 13.1에 포함됨)로 Verilog 파일을 생성할 수 있습니다. 전체 프로젝트의 implementation은 ISE의 기본 버전에서 수행할 수 있습니다.

UCF 파일에는 핀 설정 방법에 대한 안내 설명이 있습니다. PCIe 핀의 배치는 암시적이며 GTP/GTX 구성 요소의 위치에서 constraint에 의해 강제됩니다.

100 MHz의 주파수를 가진 clock은 PCIE_REFCLK 와이어 쌍에서 예상됩니다. 다른 clock이 적용되는 경우 실제 clock 주파수를 예상하도록 AMD PCIe Coregen core(pcie_v5.xco로 정의됨)를 재구성해야 합니다. 또한 TS_MGTCLK 사양이 변경 사항을 반영하도록 timing constraint를 업데이트해야 합니다.

4.3.6 Kintex-7, Virtex-7 및 Artix-7 보드 작업(PCIe)

series-7 제품군의 모든 FPGAs에는 동일한 PCIe 인터페이스가 있습니다.

- 이름이 PCIE_REFCLK_P 및 PCIE_REFCLK_N인 reference clock용 differential wires 쌍: PCIe bus clock에서 파생된(또는 직접 연결됨) 100 MHz의 주파수를 가진 clock이 이 전선에서 예상됩니다.

다른 clock이 적용되는 경우 실제 clock 주파수를 예상하도록 PCIe 블록(pcie_k7_vivado.xci 또는 demo bundle에서 이와 유사한 것으로 정의됨)을 재구성해야 합니다. 이 파일은 프로젝트의 소스 목록에 나타납니다. 이러한 변경에는 timing constraints의 변경도 포함될 수 있습니다. AMD의 도구로 생성된 예제 XCF 파일을 참조하십시오.

ISE를 사용하는 경우 AMD의 PCIe core는 예를 들어 pcie_k7_8x.xco로 정의됩니다. XDC가 아닌 UCF 파일을 조정해야 할 수 있습니다.

- PCIE_PERST_B_LS의 host의 master bus reset
- 직렬 데이터 입력 벡터 쌍, PCIE_RX_P 및 PCIE_RX_N(각각 8선 또는 4선)
- 직렬 데이터 출력 벡터 쌍, PCIE_TX_P 및 PCIE_TX_N(각각 8선 또는 4선)

핀 할당은 transceiver logic을 배치하여 암시적으로 이루어집니다. UCF/XDC 파일에서 GTX 배치를 정의하는 constraints는 특정 pinout를 강제 실행합니다. 마찬가지로 reference clock의 핀 배치는 clock buffer(pcieclk_ibuf)의 위치를 제한하여 암시적으로 설정됩니다. Xillybus의 demo bundle에 있는 UCF / XDC 파일에는 안내 설명이 포함되어 있습니다.

UCF/ XDC 파일은 이들의 핀 배치가 의도한 보드의 핀 배치와 일치하도록 편집되어야 합니다.

4.3.7 Ultrascale 및 Ultrascale+ 보드 작업(PCIe)

이 모든 FPGAs에는 동일한 PCIe 인터페이스가 있습니다.

- 이름이 PCIE_REFCLK_P 및 PCIE_REFCLK_N인 reference clock용 differential wires 쌍: PCIe bus의 clock에 직접 연결된 100 MHz 주파수의 clock.

다른 clock이 적용되는 경우 PCIe 블록(pcie_ku_vivado.xci 또는 demo bundle에서 이와 유사한 것으로 정의됨)은 실제 clock 주파수를 예상하도록 재구성되어야 하며 이 clock에 대한 timing constraint는 xillydemo.xdc에서 업데이트되어야 합니다. 이 파일은 프로젝트의 소스 목록에 나타납니다.

- PCIE_PERST_B_LS의 host의 master bus reset
- 직렬 데이터 입력 벡터 쌍, PCIE_RX_P 및 PCIE_RX_N(각각 8선 또는 4선)
- 직렬 데이터 출력 벡터 쌍, PCIE_TX_P 및 PCIE_TX_N(각각 8선 또는 4선)

핀 할당은 transceiver logic을 배치하여 암시적으로 이루어집니다. XDC 파일에서 GTX 배치를 정의하는 constraints는 특정 pinout를 강제 실행합니다. 마찬가지로 reference clock의 핀 배치는 clock buffer(pcieclk_ibuf)의 위치를 제한하여 암시적으로 설정됩니다. Xillybus의 demo bundle에 있는 XDC 파일에는 안내 설명이 포함되어 있습니다.

XDC 파일은 이들의 핀 배치가 의도한 보드의 핀 배치와 일치하도록 편집되어야 합니다.

4.3.8 Versal ACAP 보드 작업(PCIe)

이 FPGAs에는 다음과 같은 PCIe 인터페이스가 있습니다.

- 이름이 PCIE_REFCLK_P 및 PCIE_REFCLK_N인 reference clock용 differential wires 쌍: PCIe bus의 clock에 직접 연결된 100 MHz 주파수의 clock.
다른 clock이 적용되면 CPM block의 PCIe controller(pcie_versal block design의 CIPS IP 내부에 있음)는 실제 clock 주파수를 예상하도록 재구성해야 합니다.
- 직렬 데이터 입력 벡터 쌍, PCIE_RX_P 및 PCIE_RX_N(각 8선)
- 직렬 데이터 출력 벡터 쌍, PCIE_TX_P 및 PCIE_TX_N(각각 8선)

host의 master bus reset은 PMC MIO 38에 직접 연결됩니다. 다른 MIO pin이 이 신호에 사용되는 경우 CIPS IP를 그에 따라 구성해야 합니다(자세한 내용은 Xillybus의 [tutorial page](#) 참조).

XDC에는 PCIe 블록과 관련된 constraints가 포함되어 있지 않습니다. timing constraints와 pin placement constraints는 모두 CIPS IP에서 암시적으로 제공됩니다. CPM은 PCIe 인터페이스에 사용되므로 pin placements는 이동할 수 없습니다.

PCIe block의 매개변수를 변경하는 절차는 다른 FPGAs와 다릅니다. PCIe 부분은 block design로 구현됩니다. 이 block design에는 pcie_block_support라는 블록이 있습니다. 이 블록에는 PCIe 블록에서 사용되는 transceivers 및 clock resources가 포함되어 있습니다. 따라서 lanes나 link speed의 번호를 변경한 후에는 pcie_block_support를 업데이트해야 합니다. pcie_block만 업데이트하는 것만으로는 충분하지 않습니다.

pcie_block_support를 업데이트하는 방법은 이 블록을 삭제하고 Vivado가 다시 생성하도록 하는 것입니다. 이런 방식으로 PCIe 블록의 업데이트된 매개변수를 사용하여 블록이 생성됩니다.

이 절차를 시작하기 전에 block design의 시각적 복사본을 만드는 것이 좋습니다. 업데이트된 pcie_block_support 블록으로 동일한 block design을 재구성해야 하기 때문에 이는 도움이 될 것입니다.

이 절차의 단계는 다음과 같습니다.

- pcie_block_support 블록과 모든 external ports를 삭제합니다. 이는 아무 것도 연결되지 않은 ports(예: "pcie_mgt")를 삭제하고, 블록에 연결된 ports(예: "m_axis_cq_0")도 삭제한다는 의미입니다.
- versal_cips_0와 pcie_block 사이의 reset signal 연결을 삭제합니다.
- 이전 단계에 대한 응답으로 Vivado는 "Run Block Automation"을 제안해야 합니다. 해당 제안을 클릭하세요. Vivado는 이에 대한 응답으로 pop-up windows를 엽니다. PCIe 매개변수가 올바른지 확인하고 "OK"를 클릭합니다. Vivado는 "Run Connection Automation"도 제안하지만 이 옵션으로는 충분하지 않습니다.

- Vivado는 새로운 `pcie_block_support` 블록을 추가하고 여러 연결을 만듭니다.
- “`sys_reset`”와 관련된 `external port`를 제거합니다. 대신 `versal_cips_0`의 `pl_pcie0_resetrn`을 `pcie_block`와 `pcie_block_support` 두 블록의 `sys_reset inputs`에 연결하세요. 이렇게 하면 “`sys_reset`”이 이전과 같이 연결됩니다.
- PCIe 블록 중 아무 것도 연결되지 않은 pins를 모두 선택합니다(이런 목적으로 CTRL-click을 사용할 수도 있습니다). `right-click > Make External`을 사용하여 ports를 외부로 만듭니다. Vivado는 모든 pins에 대해 ports를 생성합니다. 각 `external port`의 이름은 `net`의 이름과 같으며 “_0” 접미사가 추가됩니다.

4.3.9 XillyUSB로 작업하기

XillyUSB는 SFP+ 인터페이스가 있는 다른 보드에서 사용할 수 있습니다. 이 경우 SFP+ 커넥터에 연결된 MGT를 사용하도록 design의 constraints를 설정하기만 하면 됩니다.

보드는 또한 MGT에 대해 낮은 jitter와 함께 125 MHz reference clock을 공급해야 합니다. USB 사양의 요구 사항에도 불구하고 Spread Spectrum Clocking (SSC)는 활성화되어서는 안 됩니다(해당 옵션이 있는 경우): SSC reference clock이 사용되는 경우 MGT는 수신된 신호에 대해 제대로 잠기지 않습니다.

맞춤형 보드의 경우 SFP+ 커넥터의 핀이 FPGA의 MGT에 직접 연결되므로 `sfp2usb` 모듈의 회로도를 참조하는 것이 좋습니다. `sfp2usb` 모듈에서와 같이 SSRX 와이어를 교체하는 것은 선택 사항입니다. 이는 PCB design을 단순화하는 경우에만 권장됩니다.

원하는 경우 SSTX 와이어를 교체하는 것도 가능합니다. 이를 위해서는 전송된 비트의 극성이 반전되도록 `*_frontend.v` 파일을 편집해야 하므로 전선 쌍 교환을 보상합니다. USB 사양은 극성 교체로도 link partners가 제대로 작동해야 하므로 이 편집 없이도 USB 연결이 제대로 작동할 가능성이 높습니다. 그러나 이것에 의존하지 않는 것이 좋습니다.

4.4 PCIe lanes의 수를 나타내는 PRSNT 핀

PCIe 사양에 따르면 PCIe 커넥터에는 하나 이상의 핀이 있으며, 이는 PCIe slot의 주변 장치와 lanes의 수를 나타냅니다. PRSNT 핀입니다. 대부분의 개발 보드에는 이러한 핀 덕분에 host에 정보를 제공하는 lanes 수를 조정하기 위한 DIP switches가 있습니다.

이 핀의 일반적인 기본 설정은 보드에서 가능한 최대 lanes 수입니다. 이 설정은 실제로 사용되는 lanes가 더 적은 경우에도 일반적으로 작동합니다. 이는 host와 주변 장치(PCIe 사양에서 요구됨) 간의 초기 협상이 lanes의 실제 수를 올바르게 감지하도록 보장하기 때문입니다.

이러한 DIP switches를 설정하는 방법은 보드의 참조 설명서를 참조하십시오. 일부 hosts는 잘못된 설정의 결과로 lanes를 무시할 수 있으므로 이러한 DIP switches를 실제

로 사용되는 것보다 적은 lanes로 설정하지 않는 것이 중요합니다.

4.5 PCIe lanes 및/또는 link speed의 번호 변경

4.5.1 소개

중요한:

*link*의 매개변수를 변경하려면 *timing constraints*에서 조정이 필요할 수 있습니다. 이 문제에 주의를 기울이지 않으면 작동하지만 신뢰할 수 없는 방식으로 *PCIe link*이 발생할 수 있습니다.

필요한 경우 변경한 후에는 항상 *timing constraints*를 적절하게 조정했는지 확인하십시오. 이 주제는 아래에 자세히 설명되어 있습니다.

Xillybus의 FPGA demo bundles는 일반적으로 의도한 보드에서 사용 가능한 최대 lanes 수와 2.5 GT/s의 link speed(Gen1)로 설정됩니다.

그 근거는 FPGA 보드가 마더보드의 PCIe 커넥터에 맞으면 모든 lanes가 host와 연결하여 사용될 것이라고 기대할 수 있다는 것입니다. 반면에, 거의 모든 경우에 이러한 lanes에 의해 달성되는 대역폭은 2.5 GT/s를 사용하는 경우에도 Xillybus IP core가 활용할 수 있는 것보다 높으므로 더 높은 link speed를 설정하는 것은 무의미합니다.

PCIe 사양에는 관련된 모든 bus 구성 요소의 속도를 낮추기 위한 폴백 기능이 필요하므로 2.5 GT/s를 선택하면 모든 마더보드에서 균일한 동작을 보장합니다.

그러나 특히 사용자 정의 보드에서 Xillybus IP core를 사용할 때 lanes 및 해당 link speed의 수를 변경하는 것이 종종 바람직합니다. 더 적은 lanes와 더 높은 link speed는 일반적인 요구 사항입니다.

Xillybus IP core는 PCIe bus와의 저수준 인터페이스를 위해 AMD의 PCIe 블록에 의존합니다. 따라서 AMD의 PCIe 블록이 제대로 작동하는 한 IP core는 lanes 또는 link speed의 수에 관계없이 제대로 작동합니다.

PCIe 블록이 link speed와 결합된 적은 수의 lanes로 구성되어 대역폭 기능이 Xillybus IP core보다 낮더라도 여전히 제대로 작동합니다. 이 경우 Xillybus의 streams가 제공하는 총 대역폭은 PCIe 블록의 설정에 의해 부과된 대역폭 제한과 거의 같습니다. 무엇이 병목이 되는가의 문제입니다.

4.5.2 작업 절차

Versal ACAP FGAs의 경우 4.3.8 섹션을 참조하세요. 아래 설명은 다른 모든 FGAs에 적용됩니다.

원칙적으로 lanes 및/또는 link speed의 수를 변경하는 것은 PCIe 블록의 구성을 원하는 대로 변경하는 것으로 구성됩니다. 그러나 주의해야 할 몇 가지 문제가 있습니다.

- 수정은 PCIe 블록의 다른 매개변수에 영향을 주어 올바르게 작동하지 못하게 할 수 있습니다. 무엇보다도 AMD의 GUI 도구에는 아래에 자세히 설명된 대로 주의해야 할 버그가 있습니다.
- 수정은 PCIe 블록(PIPE clocks)을 구동하는 clocks의 주파수를 변경할 수 있으므로 timing constraints의 변경이 필요합니다.
- clock 주파수를 변경하려면 PCIe 블록(해당되는 경우 pipe_clock 모듈의 instantiation)을 지원하는 Verilog 코드를 변경해야 할 수도 있습니다. 그렇지 않으면 PCIe 블록이 작동하지 않습니다.

따라서 단계는 다음과 같습니다.

1. 활성 프로젝트에서 XCO 또는 XCI 파일의 복사본을 만드십시오(즉, Vivado 또는 ISE가 필요에 따라 IP를 업그레이드한 후). 이렇게 하면 이후에 변경 사항을 diff 도구와 비교하고 원하지 않는 변경 사항이 발생하는 경우 발견할 수 있습니다.
2. 구성을 위해 Vivado(또는 ISE)에서 PCIe 블록의 IP를 엽니다(Versal FPGAs를 사용하면 pcie_versal block design의 유일한 블록인 CIPS IP에서 CPM unit을 엽니다).
3. (AXI) interface width를 변경하지 않도록 주의하면서 lanes 및/또는 Maximum Link Speed의 번호를 원하는 대로 변경하십시오. 작업에 적합한 lanes와 link speed의 조합을 선택하여 (AXI) Interface Frequency의 변경을 피할 수 있다면 그것이 바람직합니다.
4. 원하는 변경을 수행한 후 Vendor ID 및 Device ID가 변경되지 않았는지 확인하십시오(Subsystem에 해당하지 않음). Vivado의 일부 개정판은 관련 없는 수정의 결과로 일부 매개변수를 기본값으로 재설정할 수 있습니다(버그입니다).
5. 변경 사항을 확인합니다(일반적으로 대화 상자 하단에서 "OK" 클릭). 확인 후 제안되는 경우 출력 제품을 생성할 필요가 없습니다.
6. 업데이트된 XCO 또는 XCI 파일을 텍스트 diff 도구와 비교하고 관련 매개변수만 변경되었는지 확인합니다. 이에 대한 자세한 내용은 아래에 있습니다.
7. xillybus.v 및 xillydemo.v/.vhd에서 PCIE_*의 신호 벡터 너비를 조정하여 lanes의 새 번호를 반영하도록 합니다.
8. 필요한 경우 아래 4.5.3 단락에 설명된 대로 PIPE clock 모듈의 instantiation을 조정합니다.

9. 필요한 경우 아래 4.5.4 단락에 설명된 대로 timing constraints를 조정합니다.
10. 4.5.5 단락에 설명된 대로 PIPE clock 모듈을 업데이트합니다.

Ultrascale 이상으로 작업할 때는 마지막 세 단계가 필요하지 않습니다.

새로운 XCI 파일과 기존 XCI 파일을 비교할 때 PARAM_VALUE.Device_ID는 종종 실수로 변경되기 때문에 특별한 주의를 기울여야 합니다.

XCI 파일의 매개변수 차이는 원하는 것과 일치해야 합니다. 이것은 Vivado의 변경 사항에 따라 변경이 허용되는 가능한 매개변수의 짧은 목록입니다. Vivado의 다른 개정판(따라서 PCIe 블록의 다른 개정판)이 다른 XML 매개변수를 사용하여 PCIe 블록의 속성을 나타낼 수 있으므로 매개변수의 이름을 약간의 의미로 사용해야 합니다.

- lanes의 수와 관련:

- PARAM_VALUE.Maximum_Link_Width
- MODELPARAM_VALUE.max_lnk_wdt

- link speed 관련:

- PARAM_VALUE.Link_Speed
- PARAM_VALUE.Trgt_Link_Speed
- MODELPARAM_VALUE.c_gen1
- MODELPARAM_VALUE.max_lnk_spd

- 인터페이스 주파수와 관련이 있습니다. 이러한 매개변수의 변경은 4.5.3 및 4.5.4 단락의 단계가 필요하다는 강력한 표시입니다.

- PARAM_VALUE.User_Clk_Freq
- MODELPARAM_VALUE.pci_exp_int_freq

4.5.3 PIPE 주파수가 변경되었습니까?

Ultrascale FPGAs 이상으로 작업할 때 PCIe 블록이 timing constraints를 IP 자체의 필수 부분으로 제공하기 때문에 아래의 고려 사항 및 조치가 필요하지 않습니다. PIPE 모듈도 마찬가지입니다.

다른 FPGA 제품군의 경우 다음과 같이 PIPE clock 설정이 올바른지 확인하는 것이 중요합니다.

변경 후 PCIe 블록에 대한 예제 프로젝트를 생성하고 해당 프로젝트의 synthesis를 실행합니다. Vivado에서 이는 일반적으로 프로젝트의 소스 계층에서 PCIe 블록을 마우스 오른쪽

버튼으로 클릭하고 “Open IP Example Design...”을 선택하여 수행됩니다. design의 위치를 선택하고 생성된 후 왼쪽 열에서 “Run Synthesis”를 클릭하여 synthesis를 시작합니다. 다음으로, synthesis report에서 PIPE clock 모듈의 instantiation parameters를 얻습니다(Vivado에서는 pcie_example/pcie_example.runs/synth_1/runme.log와 같은 것으로 발견됨). 이 보고서에서 다음과 같은 세그먼트를 검색합니다.

```
INFO: [Synth 8-638] synthesizing module 'example_pipe_clock' [...]
Parameter PCIE_ASYNC_EN bound to: FALSE - type: string
Parameter PCIE_TXBUF_EN bound to: FALSE - type: string
Parameter PCIE_CLK_SHARING_EN bound to: FALSE - type: string
Parameter PCIE_LANE bound to: 4 - type: integer
Parameter PCIE_LINK_SPEED bound to: 3 - type: integer
Parameter PCIE_REFCLK_FREQ bound to: 0 - type: integer
Parameter PCIE_USERCLK1_FREQ bound to: 4 - type: integer
Parameter PCIE_USERCLK2_FREQ bound to: 4 - type: integer
Parameter PCIE_OOBCLK_MODE bound to: 1 - type: integer
Parameter PCIE_DEBUG_MODE bound to: 0 - type: integer
```

이 보고서의 매개변수는 다음 형식의 xillybus.v에 표시되므로 pipe_clock의 instantiation에 있는 매개변수와 일치해야 합니다.

```
pcie_[...]_pipe_clock #
(
    .PCIE_ASYNC_EN          ( "FALSE" ),
    .PCIE_TXBUF_EN          ( "FALSE" ),
    .PCIE_LANE              ( 6'h08 ),
    .PCIE_LINK_SPEED        ( 3 ),
    .PCIE_REFCLK_FREQ        ( 0 ),
    .PCIE_USERCLK1_FREQ      ( 4 ),
    .PCIE_USERCLK2_FREQ      ( 4 ),
    .PCIE_DEBUG_MODE        ( 0 )
)
pipe_clock
(
    [ ... ]
);
```

비교할 세 가지 매개변수는 PCIE_LINK_SPEED, PCIE_USERCLK1_FREQ 및 PCIE_USERCLK2_FREQ이며 일치해야 합니다. 그렇게 하면(예제에 표시된 대로) timing constraints를 포함하여 모든 것이 올바르게 설정됩니다. 그렇지 않은 경우 두 가지 조치를 취해야 합니다.

- xillybus.v의 instantiation parameters는 예제 프로젝트의 synthesis report와 일치

하도록 업데이트해야 합니다.

- timing constraints는 예제 프로젝트에 맞게 조정되어야 합니다. 이 작업을 올바르게 수행하지 않으면 즉시 문제가 발생하지는 않지만 design의 안정성에 영향을 미칠 수 있기 때문에 이것은 더 어렵습니다.

xillybus.v의 PCIE_LANE 매개변수가 예제 프로젝트의 것보다 크면 그대로 두어도 문제가 없으며 종종 그렇게 하는 것이 더 쉽습니다.

4.5.4 timing constraints 적용

PCIe 블록의 clocks 변경 사항이 발생한 경우 이를 반영하도록 timing constraints를 조정하는 것이 필수입니다.

constraints는 선택한 FPGA와 Vivado의 개정판에 의존하므로 이를 올바르게 수행하는 것이 다소 어려울 수 있습니다. 이 조정을 피하는 것이 link speed와 lanes 수(가능한 경우)의 조합을 선택하여 PIPE clock의 주파수를 변경하지 않고 유지하려는 주된 동기입니다. 그러나 PIPE clock의 주파수가 변경되지 않은 경우에도 constraints를 업데이트해야 할 수 있습니다.

다시 한 번, Ultrascale 제품군(이상)의 FPGA를 사용할 때 PCIe 블록의 IPs가 내부적으로 처리하기 때문에 timing constraints를 처리할 필요가 없습니다.

timing constraints를 조정하려면 먼저 예제 프로젝트의 constraints를 찾습니다. Vivado의 경우 일반적으로

example.srcs/constrs_1/imports/example_design/xilinx_*.xdc 형식의 파일입니다.

lanes 및/또는 link speed의 수를 변경하기 전에 PCIe 블록의 설정으로 하나의 예제 프로젝트를 생성하고 이러한 변경 후에 하나의 예제 프로젝트를 생성하는 것이 좋습니다. 두 예제 프로젝트의 constraint 파일 사이에 있는 간단한 diff는 constraints의 적용이 필요한지, 그리고 필요한 경우 어떤 방식으로 적용되는지에 대한 확실한 답을 제공합니다.

constraint 파일의 “Timing constraints” 섹션을 xillydemo.xdc의 섹션과 비교하십시오. 예제 프로젝트는 logic 계층 구조에서 절대 위치에 따라 logic 요소를 선택하므로 일부 편집이 필요합니다. 예를 들어 예제 프로젝트에서 다음과 같은 timing constraint가 있다고 가정합니다.

```
set_false_path -to [get_pins {pcie_vivado_support_i/pipe_clock_i/
pclk_il_bufgctrl.pclk_il/S0}]
```

xillydemo.xdc에서는 다음과 같이 작성해야 합니다.

```
set_false_path -to [get_pins -match_style ucf */pipe_clock/
pclk_i1_bufgctrl.pclk_i1/S0]
```

주요 차이점은 Xillybus의 constraints에서 사용되는 상대 경로입니다. AMD 도구의 이전 버전에서는 일부 constraints가 필요하고 이후 도구에서는 불필요해지기 때문에 다른 약간의 차이점도 있을 수 있습니다.

timing constraints에서 변경한 후 design의 implementation 이후 timing report를 검토하여 logic에 적용되었는지 확인하는 것이 중요합니다.

마지막으로 일부 demo bundles의 xillydemo.xdc에 있는 다음 두 constraints를 설명할 가치가 있습니다.

```
set_case_analysis 1 [get_pins -match_style ucf */pipe_clock/
pclk_i1_bufgctrl.pclk_i1/S0]
set_case_analysis 0 [get_pins -match_style ucf */pipe_clock/
pclk_i1_bufgctrl.pclk_i1/S1]
```

이 constraints는 [Xilinx AR #62296](#)에 설명된 대로 Vivado의 아주 오래된 개정판을 사용할 때 Gen1 PCIe 블록에 필요합니다. 따라서 최근 AMD 도구에서는 생략될 수 있습니다.

4.5.5 PIPE clock 모듈 업데이트

위에서 언급했듯이 이 단계는 Ultrascale FPGAs 이상에서는 필요하지 않습니다.

어떤 경우에는 특히 link speed를 2.5 GT/s (Gen1)에서 또는 2.5 GT/s (Gen1)로 늘릴 때 vivado-essentials/ 디렉토리에 있는 pcie_*_vivado_pipe_clock.v 파일을 업데이트해야 합니다.

이 파일은 xillydemo-vivado.tcl을 실행하여 초기 프로젝트 설정의 일부로 자동으로 생성됩니다. PCIe 블록의 구성, 특히 2.5 GT/s로 제한되는지 여부에 따라 약간 변경될 수 있습니다.

권장되는 방법은 xillydemo-vivado.tcl script로 Vivado 프로젝트를 재생성하는 것입니다. 즉, 새로운 demo bundle에서 시작하고 이전 단계에서 변경된 파일을 복사합니다.

- PCIe 블록의 XCI 파일
- xillydemo.xdc
- lanes 및 link speed의 새로운 수에 맞게 편집된 Verilog / VHDL 파일입니다.

이 파일을 새 demo bundle에 복사한 상태에서 xillydemo-vivado.tcl script로 프로젝트를 생성하면 PIPE clock module이 PCIe 블록의 설정을 따르고 프로젝트가 변경 전의 남은 파일에 의존하지 않도록 합니다.

또는 4.5.3 단락에서 생성된 예제 프로젝트에서 `pcie_*_vivado_pipe_clock.v` 파일을 업데이트합니다. 사용할 파일은 `vivado-essentials/`에 있는 것과 정확히 같은 이름을 가지며 일반적으로 예제 프로젝트의 파일 계층 구조 깊은 곳에서 찾을 수 있습니다. 이 파일을 `vivado-essentials/`에 복사합니다(기존 파일 덮어쓰기).

4.6 FPGA 부품 번호 변경

한 FPGA 제품군에서 다른 제품군으로 마이그레이션할 때 다른 demo bundle에서 시작해야 합니다. PCIe 블록(및 XillyUSB가 있는 MGT 블록)과 관련된 차이점(때로는 미묘함)이 있습니다. 이러한 차이점에는 다른 Xillybus IP core와 다른 wrapper modules가 필요합니다.

Vivado / ISE에서 프로젝트의 일부만 변경하려고 하면 implementation 중에 프로젝트에 오류가 발생할 수 있습니다. implementation이 성공하더라도 logic이 작동하지 않거나 불안정하게 작동할 수 있습니다.

그러나 동일한 FPGA 제품군에 남아 있는 경우 부품 번호를 변경하는 것으로 충분합니다(핀 배치 및 constraints와 관련하여 위에서 언급한 고려 사항 포함).

일부 FPGA 제품군(특히 Ultrascale)의 경우 logic fabric에서 PCIe 블록의 위치(site)는 PCIe 블록 자체의 속성이므로 수정이 필요할 수 있습니다. 또한 각 특정 FPGA 및 각 특정 package에는 고유한 유효한 sites 세트가 있습니다. 따라서 PCIe 블록에 대해 선택된 site가 새 FPGA에 존재하지 않는 경우 FPGA를 변경하면 PCIe IP의 속성이 재설정될 수 있습니다.

PCIe 블록의 위치(site)에서 유효하지 않은 사이트에 대한 Vivado의 반응은 매우 파괴적입니다. 이 경우 PCIe 블록의 “upgrade”(FPGA를 변경한 후 IP를 잠금 해제하는 데 항상 필요한 작업)로 인해 PCIe 블록의 여러 속성이 임의의 값으로 재설정됩니다. 그렇게 하는 동안 다음과 같은 Critical Warning이 생성됩니다.

```
CRITICAL WARNING: [IP_Flow 19-3419] Update of 'pcie_ku' to current project options has resulted in an incomplete parameterization. Please review the message log, and recustomize this instance before continuing with your design.
```

이 문제에 주의를 기울이지 않고 프로젝트의 implementation을 시도하는 것은 완전히 무의미하며, 오해의 소지가 있는 warnings, Critical Warnings 및 가능한 오류가 많이 발생할 뿐만 아니라 결과가 작동하는 것과는 거리가 멉니다.

솔루션은 프로젝트의 부품 번호 속성을 변경하기 전에 새 FPGA에서 유효한 PCIe 블록 사이트를 할당하는 것입니다. 변경 전후에 FPGA 부품 사이에 공통 사이트가 없는 경우 XCI 파일을 수동으로 편집해야 할 수 있습니다(이 경우 GUI에서 이 변경을 수행할 수 없으므로 설정이 허용된 사이트로 제한됩니다. 현재 FPGA 부품에서).

5

문제 해결

5.1 implementation 중 오류

AMD 도구 릴리스 간의 약간의 차이로 인해 bitfile 생성을 위해 implementation을 실행하지 못하는 경우가 있습니다.

문제가 빠르게 해결되지 않으면 Xillybus 웹 사이트에 제공된 이메일 주소를 통해 도움을 요청하십시오. 특히 도구에서 보고한 첫 번째 오류 주변에 실패한 프로세스의 출력 로그를 첨부하십시오. 또한 design에서 사용자 지정 변경이 이루어진 경우(즉, demo bundle에서 전환) 이러한 변경 사항을 자세히 설명하십시오. 또한 어떤 버전의 Vivado(또는 ISE)가 사용되었는지 알려 주십시오.

5.2 PCIe 하드웨어 문제

일반적으로 PCIe 카드는 host의 BIOS 및/또는 운영 체제에서 올바르게 감지되고 host의 driver가 성공적으로 실행됩니다.

대부분의 PC 컴퓨터에서 BIOS는 boot 프로세스 초기에 감지된 주변 장치 목록을 간략하게 표시합니다. Xillybus 인터페이스가 성공적으로 감지되면 vendor ID 10EE 및 device ID EBEB가 있는 주변 장치가 목록에 나타납니다.

운영 체제의 카드 감지에 대해서는 다음 두 문서 중 해당하는 문서를 참조하십시오.

- [Getting started with Xillybus on a Linux host](#)
- [Getting started with Xillybus on a Windows host](#)

카드 감지 실패(또는 컴퓨터의 boot 프로세스 실패)는 Xillybus IP core와 관련이 없으며 AMD의 PCIe IP core를 사용하여 bus와 연결합니다.

처음에는 다음을 확인하는 것이 좋습니다.

- bitstream은 컴퓨터의 전원이 켜졌을 때(또는 PCI-SIG 사양 측면에서 전원이 켜진 직후) 이미 FPGA에 로드되었습니다.
- reference clock을 포함하여 PCIe 와이어의 pinouts가 정확합니다(이는 placement report에서 확인할 수 있음).
- 보드는 올바른 reference clock을 FPGA에 공급합니다.

문제가 즉시 발견되지 않으면 보드와 함께 제공된 PCIe용 샘플 프로젝트를 시도하는 것이 좋습니다. 잘못된 점퍼 설정과 하드웨어 결함이 나타날 수 있습니다.

이 샘플에서는 카드가 감지되지만 Xillybus에서는 감지되지 않는 경우 두 designs의 pinouts를 비교하는 것이 도움이 될 수 있습니다. 동일한 경우 다음 단계는 각각에 대해 IP GUI를 호출하여 AMD의 PCIe cores 속성을 비교하는 것입니다(각 프로젝트가 열린 상태에서 Project Manager에서 XCI / XCO 요소를 두 번 클릭).

다음 구성 요소를 조정해야 할 수 있습니다.

- reference clock의 주파수(GUI에서 "Interface Frequency"로 나타날 수 있음).
- base class 및 sub class(가능성은 없지만 class가 알 수 없는 것으로 간주되는 경우 상대적으로 오래된 일부 PC 컴퓨터에서 boot 프로세스가 실패했습니다).
- 기본 주소 register 설정, vendor ID, device ID 및 인터럽트 설정을 제외하고 다르게 구성된 기타 속성은 변경되어서는 안 됩니다.

문제가 지속되면 Xillybus 웹 사이트에 제공된 이메일 주소를 통해 도움을 요청하십시오.