(기계로 한국어 번역)

Getting started with Xillybus on a Windows host

Xillybus Ltd.

www.xillybus.com

Version 4.1

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 This document may also be slightly outdated in relation to the original.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

이 안내서는 Windows host에서 Xillybus / XillyUSB를 실행할 목적으로 driver를 설치하는 단계를 안내합니다. IP core의 기본 기능을 시험해 보는 방법도 나와 있습니다.

이 안내서는 Xillybus의 demo bundle을 기반으로 하는 bitstream이 FPGA에 이미 로드되어 있고 FPGA가 host에 의해 peripheral로 인식되었다고 가정합니다(PCI Express 또는 USB 3.x를 통해).

이 단계에 도달하기 위한 단계는 다음 문서 중 하나에 요약되어 있습니다(선택한 FPGA에 따라 다름).

- Getting started with the FPGA demo bundle for Xilinx
- Getting started with the FPGA demo bundle for Intel FPGA

host driver는 named pipes처럼 동작하는 device files를 생성합니다. 이 device files는 여느 파일과 마찬가지로 열고 읽고 쓸 수 있습니다. 그러나 이러한 파일은 프로세스 간에 pipes와 유사한 방식으로 작동합니다. 이 동작은 TCP/IP streams와도 유사합니다. host에서 실행되는 프로그램의 차이점은 stream의 다른 쪽이 다른 process(동일한 컴퓨터 또는 네트워크의 다른 컴퓨터)가 아니라 다른 쪽이 FPGA 내부의 FIFO라는 점입니다. TCP/IP stream와 마찬가지로 Xillybus stream은 고속 데이터 전송으로 효율적으로 작동하도록 설계되었지만 간헐적으로 소량의 데이터를 전송할 때도 stream의 성능이 우수합니다.

host의 단일 driver는 PCIe를 통해 host와 통신하는 모든 Xillybus IP cores와 함께 사용됩니다. 다른 driver는 XillyUSB와 함께 사용됩니다.

FPGA에서 다른 IP core를 사용하는 경우 driver를 변경할 필요가 없습니다. streams와 그 속성은 driver가 host의 운영 체제에 로드될 때 driver에 의해 자동으로 감지됩니다. 그에 따라 device files가 `\\.\xillybus_`*something* 형식의 파일 이름으로 생성됩니다. 마찬가지로 XillyUSB용 driver는 `\\.\xillyusb_00_`*something* 형식으로 device files를 생성합

니다. 이 파일 이름에서 00 부분은 device의 색인입니다. 하나 이상의 XillyUSB device가 동시에 컴퓨터에 연결된 경우 이 부분은 01, 02 등으로 교체됩니다.

host와 관련된 주제에 대한 더 자세한 정보는 Xillybus host application programming guide for Windows에서 찾을 수 있습니다.

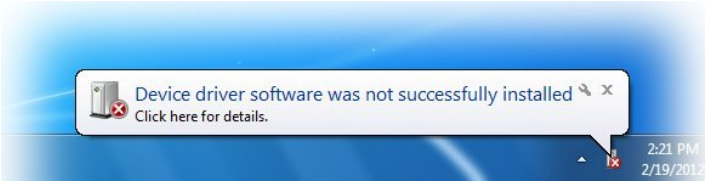
2

host driver 설치

2.1 설치 절차

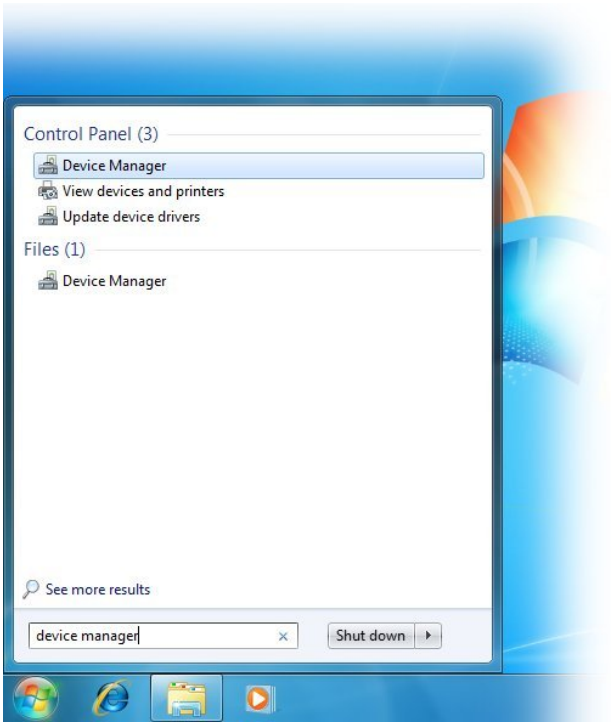
Xillybus 또는 XillyUSB용 Windows driver를 설치하는 데 특별한 것은 없습니다. 아래에 설명된 절차는 디스크의 특정 위치에서 device driver를 설치하는 일반적인 방법입니다.

boot 프로세스 중에 Windows가 PCIe Xillybus IP core를 처음 감지하면 다음과 같은 warning bubble이 나타날 가능성이 높습니다.

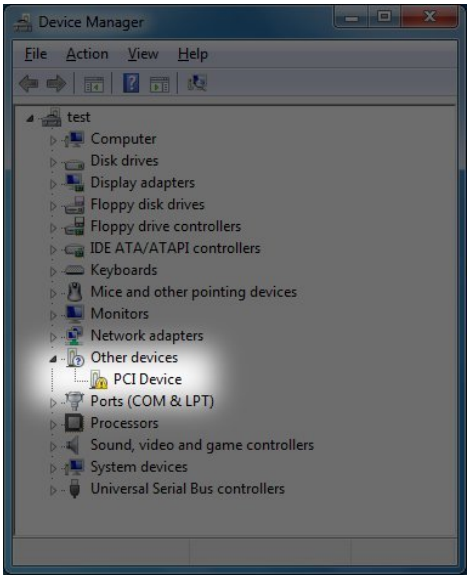


이 알림은 새 하드웨어가 발견되었지만 관련 driver가 설치되지 않았음을 나타냅니다. 이 상황은 정상이며 Windows가 아직 인식하지 못하는 것을 감지했다는 신호입니다. XillyUSB device가 컴퓨터에 처음으로 연결될 때 동일한 결과가 예상됩니다.

이 이벤트에 대응하여 Device Manager 실행부터 시작하십시오. 가장 쉬운 방법은 “Windows start” 버튼을 클릭한 다음 아래 이미지와 같이 “device manager”를 입력하는 것입니다. 그런 다음 상단의 메뉴 항목을 클릭하십시오.



열린 Device Manager는 다음과 같이 보일 것입니다(중요한 부분이 강조 표시됨).



이 스크린샷은 PCIe 시나리오와 관련이 있습니다. XillyUSB의 경우 “Universal Serial Bus

controllers" 그룹에 새 항목이 나타납니다.

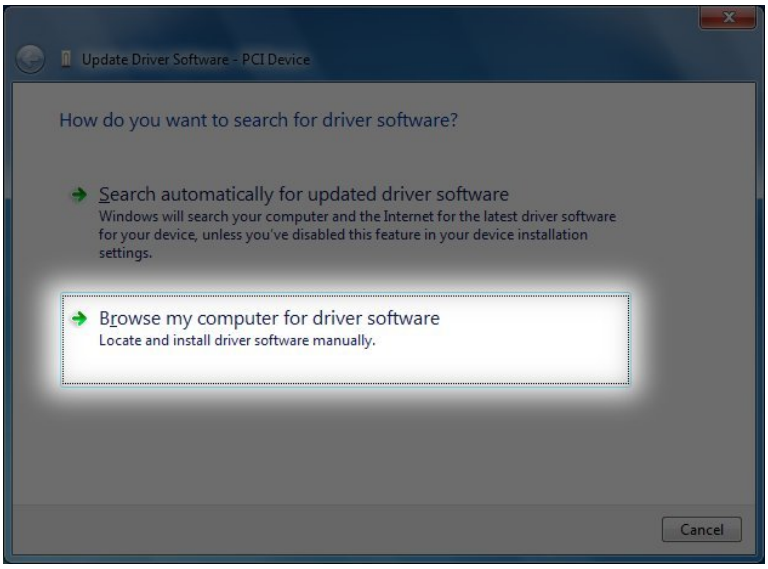
새 항목이 Device Manager에 표시되지 않으면 다음과 같은 몇 가지 원인이 있을 수 있습니다.

- FPGA에 잘못된 bitstream이 로드되었거나 bitstream이 전혀 로드되지 않았습니다.
- FPGA board가 PC에서 전원 공급을 받으면 PC computer의 전원이 켜질 때 bitstream이 FPGA에 로드됩니다. 이 사용 시나리오에서는 FPGA가 bitstream을 너무 느리게 로드했기 때문에 BIOS가 boot 중에 PCIe 인터페이스를 감지하지 못했을 가능성이 있습니다. BIOS가 컴퓨터를 초기화할 때 bitstream은 이미 FPGA 내부에 있어야 합니다.

  컴퓨터를 끄지 않고 Windows restart를 수행하는 것이 이 문제를 해결하는 안전한 방법입니다. 그러나 Action > Scan for New Hardware를 수행해도 작동할 수 있습니다.
- board의 잘못된 구성(jumpers, DIP switches 등), 잘못된 pin assignment, 잘못된 reference clock 주파수 등

  이러한 종류의 문제는 FPGA의 PCIe block이 감지되지 않기 때문입니다. 이러한 문제는 PCIe bus와의 인터페이스에 이 PCIe block(AMD 또는 Intel에서 제공)를 사용하는 Xillybus와 관련이 없습니다.
- Xillybus / XillyUSB driver는 이미 설치되어 있습니다. 이 시나리오에서 Device Manager는 설치 절차 마지막에 표시된 예와 같아야 합니다.

"PCI Device" 항목을 마우스 오른쪽 버튼으로 클릭하고 "Update Driver Software..."를 선택합니다. 다음 창이 열립니다.

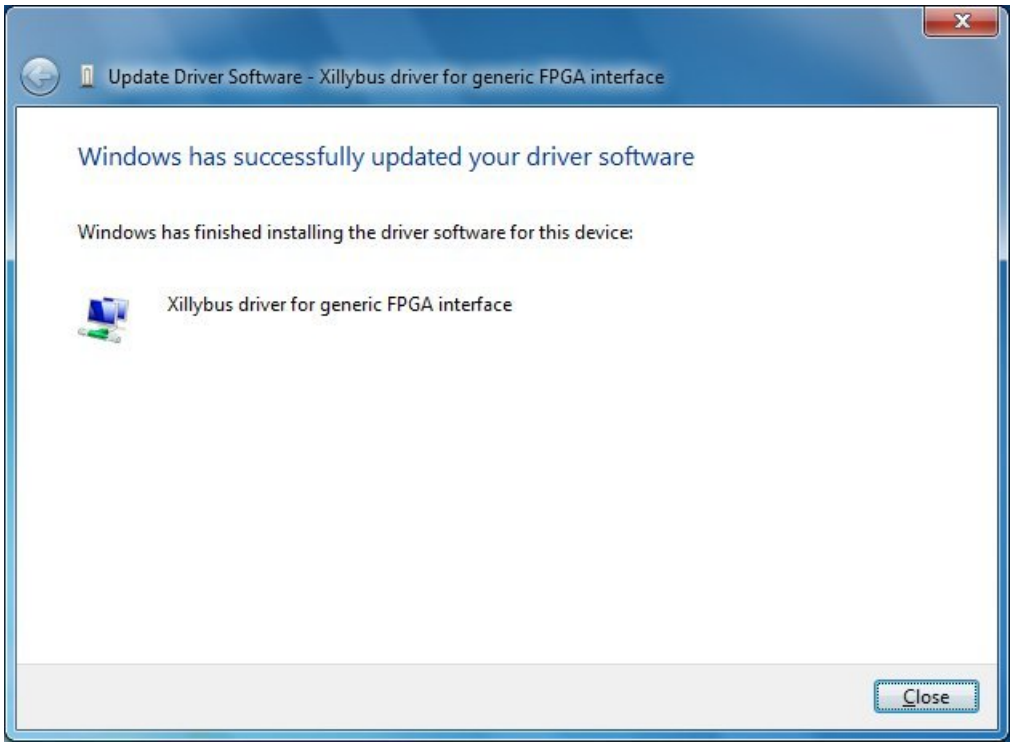


"Browse my computer for driver software"를 선택합니다. 이것은 다음 창입니다.

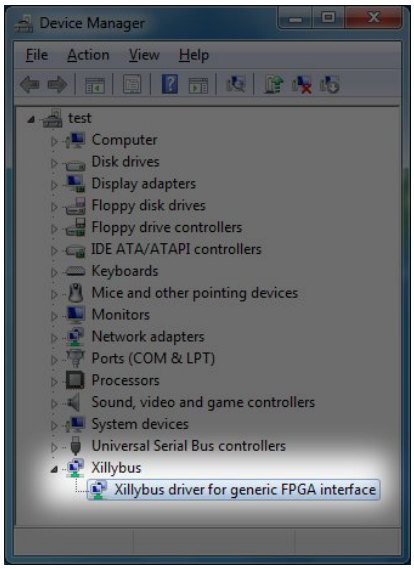
"Browse..." 버튼을 사용하여 driver가 저장된 위치로 이동합니다(driver가 압축 해제된 후). 다음 단계는 설치를 확인하는 것입니다.

"Install"을 클릭합니다. driver를 설치하는 과정은 Windows 7에서 10-20초가 걸립니다. 최신 버전의 Windows는 일반적으로 시간이 훨씬 적게 걸립니다.

다음 창에서 설치가 성공적으로 완료되었음을 알립니다.



이제 Device Manager에 새로 설치된 device가 표시됩니다.



이 시점에서 driver가 설치되고 자동으로 시스템에 로드됩니다. 이 driver는 시스템이 PCIe bus에서 Xillybus IP core로 boot를 수행할 때마다 로드됩니다. XillyUSB의 driver는 해당 장치가 host에 연결될 때마다 로드됩니다.

다음에 설명하는 것처럼 Xillybus의 log messages를 표시하도록 Event Viewer를 설정하는 것이 좋습니다.

위에 표시된 스크린샷은 PCIe용 Xillybus와 관련이 있습니다. 그러나 프로세스는 XillyUSB와 동일하지만 약간의 차이가 있습니다. 특히 Device Manager의 새로운 그룹은 “Xillybus”가 아닌 “XillyUSB”라고 합니다.

2.2 device files

응용 컴퓨터 프로그램은 표준 file I/O API를 사용하여 Xillybus IP core와 통신합니다. 그러나 Xillybus와 연동하기 위해서는 일반 파일에 접근하는 대신 device files를 사용한다.

따라서 Xillybus의 driver의 목적은 이러한 device files를 FPGA와 통신하는 메커니즘으로 운영 체제 내부에 생성하는 것입니다. 이러한 device files는 Microsoft의 용어로 “Windows objects”라고 합니다.

간단한 컴퓨터 프로그램에서 직접 Windows objects에 액세스하는 것은 신뢰할 수 없는 방법으로 보일 수 있습니다. 그러나 Microsoft Windows의 내부 구조에 익숙한 사람들은 하드웨어와 소프트웨어의 인터페이스가 정확히 이와 같이 수행되는 경우가 많다는 것을 알고 있습니다. 이러한 device files가 애플리케이션 소프트웨어에 직접 노출되는 것은 실제로 드문 일입니다. 오히려 하드웨어 제조업체는 종종 프로그램이 API를 통해 하드웨어에 액세

스할 수 있도록 하는 DLL을 제공합니다. 배후에서 이 DLL은 필요한 기능을 수행하기 위해 device files를 사용합니다.

Xillybus의 인터페이스는 매우 간단하기 때문에 이러한 DLL은 필요하지 않습니다. 따라서 user application software는 device files에 직접 액세스합니다. 그러나 불행히도 Windows는 Xillybus의 device files 목록을 얻는 간단한 방법을 제공하지 않습니다. 이는 Windows objects에 액세스하는 것이 고급 기술로 간주되기 때문입니다. 따라서 운영 체제에서 이 정보를 얻으려면 유틸리티 프로그램을 다운로드해야 합니다. 그러나 아래에 설명된 대로 device files 목록은 다른 출처에서도 얻을 수 있습니다.

WinObj 유틸리티(Microsoft 사이트에서 다운로드 가능)를 사용하면 Window의 object tree를 탐색할 수 있습니다. Xillybus / XillyUSB device files는 GLOBAL??라는 이름으로 “subdirectory”에서 symbolic links로 찾을 수 있습니다. 다른 잘 알려진 Windows objects는 같은 위치에서 찾을 수 있습니다(예: C: 및 COM1:).

accesschk라는 이름의 command-line 도구도 있습니다. 이 도구는 Microsoft’s website에서 다운로드할 수 있습니다. Xillybus / XillyUSB device files의 이름을 가져오는 명령은 다음과 같습니다.

```
> accesschk -o \\GLOBAL\?\?
```

다른 많은 글로벌 device files가 이 작업과 함께 나열됩니다.

이 두 가지 도구를 사용하여 device files 목록을 얻을 수 있지만 그렇게 할 필요는 없습니다. device files의 이름은 미리 알려져 있습니다.

Xillybus의 device files에는 PCIe 인터페이스가 사용될 때 `\\.\xillybus_` 형식의 접두어가 있습니다. XillyUSB의 경우 접두사는 `\\.\xillyusb_nn_`입니다.

다음은 demo bundle의 PCIe 변형에 의해 생성된 device files 목록입니다.

- `\\.\xillybus_read_8`
- `\\.\xillybus_write_8`
- `\\.\xillybus_read_32`
- `\\.\xillybus_write_32`
- `\\.\xillybus_mem_8`

host에 연결된 단일 XillyUSB 장치의 경우 다음은 device files입니다.

- `\\.\xillyusb_00_read_8`
- `\\.\xillyusb_00_write_8`

- `\\.\xillyusb_00_read_32`
- `\\.\xillyusb_00_write_32`
- `\\.\xillyusb_00_mem_8`

IP Core Factory에서 생성된 맞춤형 IP core의 경우: device files 목록은 다운로드한 zip 파일에 포함된 README 파일에서 확인할 수 있습니다.

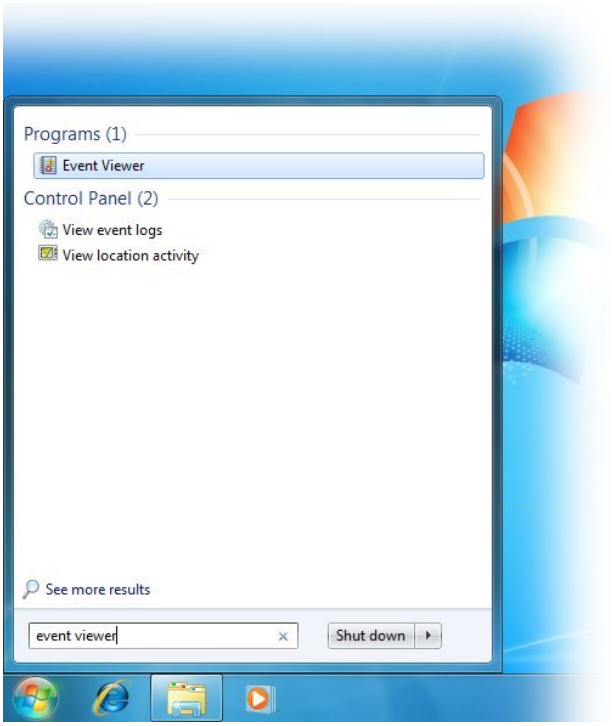
많은 프로그래밍 언어(예: C/C++)에서 파일 이름의 backslashes 앞에 escape character가 필요합니다. 따라서 device file의 이름을 예를 들어 `\\\\.\\xillybus_read_8`로 작성해야 할 수도 있습니다.

2.3 진단 정보 얻기

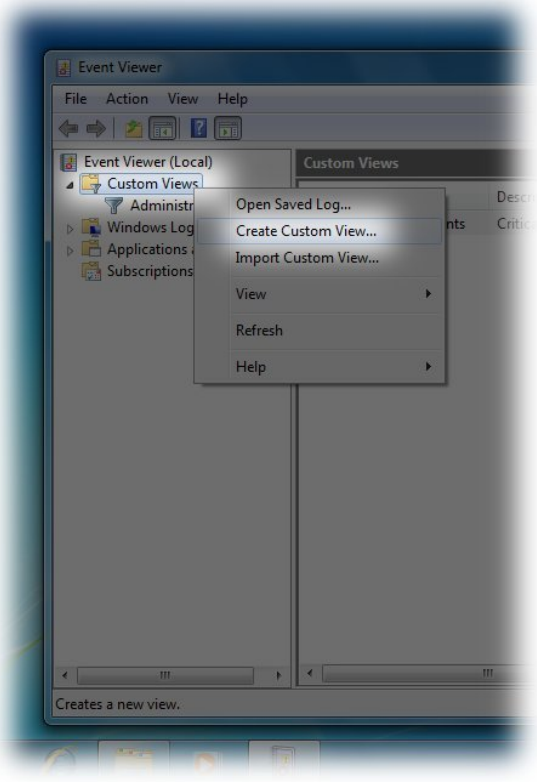
Xillybus / XillyUSB용 driver는 운영 체제의 기본 event logger에 진단 메시지를 보냅니다. 이러한 메시지에는 driver가 초기화에 실패했을 때 무엇이 잘못되었는지에 대한 정보가 포함됩니다(예: DMA buffers에 대한 메모리가 충분하지 않음). 문제 해결에 도움이 될 수 있는 다른 메시지도 전송됩니다.

다음에 설명된 절차는 Event Viewer에서 custom view를 생성하는 방법을 보여줍니다. 이 작업의 목적은 Xillybus 또는 XillyUSB와 관련된 메시지를 표시하는 것입니다.

먼저 Event Viewer를 엽니다. 가장 쉬운 방법은 “Windows start” 버튼을 클릭한 다음 아래와 같이 “event viewer”를 입력하는 것입니다. 상단의 메뉴 항목을 클릭합니다.

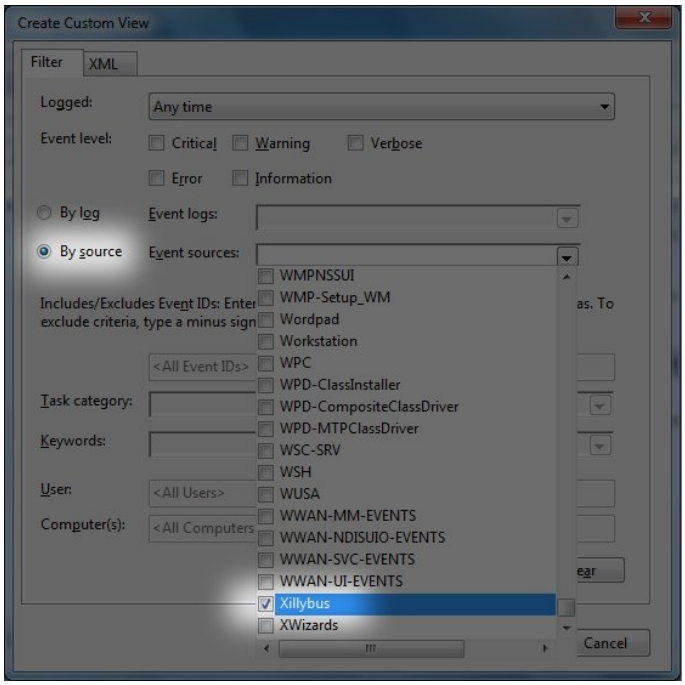


Event Viewer가 열립니다. “Custom Views”를 마우스 오른쪽 버튼으로 클릭하고 메뉴에서 “Create Custom View...”를 선택합니다.



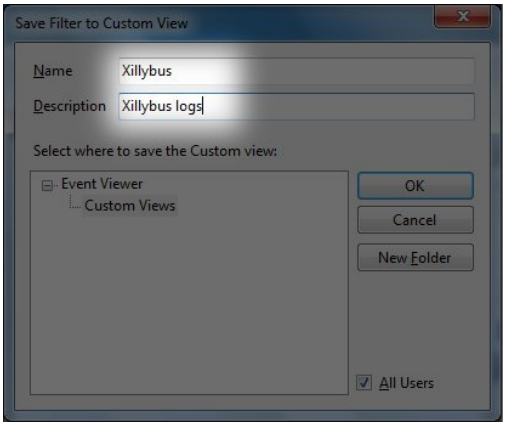
제목이 “Create Custom View”인 창이 열립니다. 이 창의 용도는 표시할 메시지를 선택하는 필터를 정의하는 것입니다. “By source”를 선택합니다. drop-down menu에서 Xillybus를 선택하십시오. 다른 옵션은 기본값을 유지합니다.

대신 XillyUSB에서 메시지를 얻으려면 메뉴에서 XillyUSB를 선택하십시오.

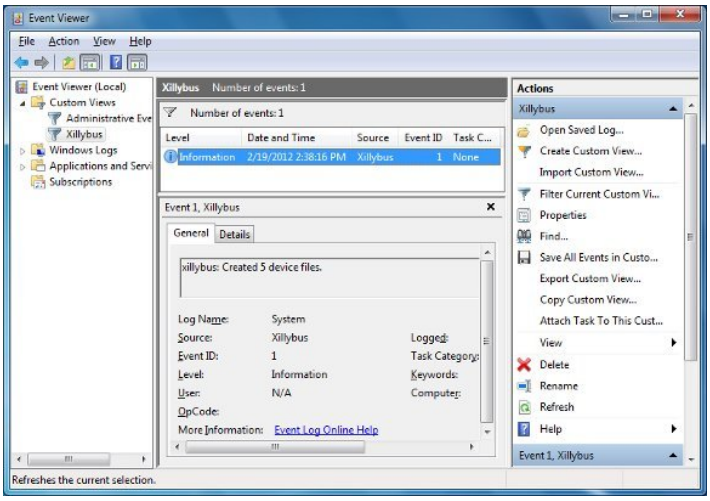


drop-down menu에서 Xillybus / XillyUSB 항목을 찾을 수 없으면 driver가 제대로 설치되었는지 확인하십시오.

“OK”를 클릭하면 이 custom view에 이름과 설명을 할당하기 위한 다른 창이 열립니다. 이것은 개인적인 선택의 문제입니다.



“OK”를 클릭하면 Event Viewer는 다음과 같이 보입니다.



위의 이미지는 Xillybus가 제대로 시작되었고 5개의 device files가 생성되었음을 알리는 하나의 메시지를 보여줍니다. 이것은 FPGA가 demo bundle와 함께 로드될 때 driver를 성공적으로 설치한 직후 예상되는 것입니다.

메시지는 컴퓨터의 reboot에서 삭제되지 않습니다. 따라서 이 custom view는 driver가 설치된 이후의 기록을 보여줍니다(기록을 의도적으로 삭제하지 않는 한).

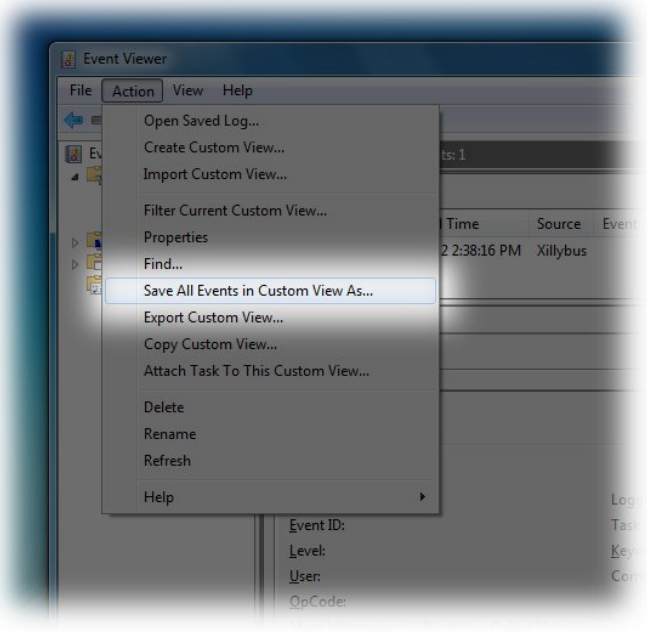
PCIe driver의 메시지 목록과 설명은 다음에서 찾을 수 있습니다.

http://xillybus.com/doc/list-of-kernel-messages

그러나 메시지 텍스트에서 Google을 사용하면 특정 메시지를 찾는 것이 더 쉬울 수 있습니다.

메시지를 파일로 내보낼 수도 있습니다. 지원을 요청할 때 driver의 메시지가 포함된 파일을 보내는 것이 좋습니다. 이러한 메시지에는 종종 귀중한 정보가 포함되어 있습니다.

이러한 파일을 생성하려면 “Action” 메뉴 항목을 선택한 다음 “Save All Events in a Custom View As...”를 선택합니다.



파일 선택 창이 열립니다. 도움을 요청하려면 CSV를 출력 형식으로 선택하십시오.

3

“Hello, world” 테스트

3.1 목표

Xillybus는 logic design의 구성 요소로 사용되는 도구입니다. 따라서 Xillybus의 기능을 배우는 가장 좋은 방법은 user application logic와 통합하는 것입니다. demo bundle의 목적은 Xillybus와 작업하기 위한 출발점이 되는 것입니다.

따라서 가능한 가장 간단한 애플리케이션이 demo bundle에서 구현됩니다. 두 device files 사이의 loopback. 이것은 FIFO의 양쪽을 FPGA의 Xillybus IP Core에 연결함으로써 달성됩니다. 결과적으로 host가 하나의 device file에 데이터를 쓸 때 FPGA는 다른 device file을 통해 동일한 데이터를 host로 반환합니다.

아래의 다음 몇 섹션에서는 이 간단한 기능을 테스트하는 방법을 설명합니다. 이 테스트는 Xillybus가 올바르게 작동하는지 확인하는 간단한 방법입니다. FPGA의 IP Core는 예상대로 작동하고 host는 PCIe 주변 장치를 올바르게 감지하며 driver는 올바르게 설치됩니다. 게다가 이 테스트는 FPGA에서 logic design을 약간 수정하여 Xillybus가 어떻게 작동하는지 배울 수 있는 기회이기도 합니다.

첫 번째 단계로 FPGA의 logic와 device files가 함께 작동하는 방식을 이해하기 위해 demo bundle로 간단한 실험을 하는 것이 좋습니다. 이것만으로도 애플리케이션의 요구 사항에 따라 Xillybus를 사용하는 방법이 명확해지는 경우가 많습니다.

위에서 언급한 loopback 외에도 demo bundle은 RAM와 추가 loopback도 구현합니다. 이 추가 loopback은 아래에서 간략하게 설명합니다. RAM와 관련하여 메모리 어레이 또는 registers에 액세스하는 방법을 보여줍니다. 이에 대한 자세한 내용은 3.4 섹션을 참조하십시오.

3.2 준비

"Hello, world" 테스트는 Command Prompt 창을 사용하여 간단한 command-line 프로그램을 실행하는 것으로 구성됩니다.

첫 번째 단계로 Xillybus package for Windows를 다운로드합니다. driver를 제공하는 동일한 웹 페이지에서 사용할 수 있는 zip 파일입니다. 이 zip 파일에는 이러한 프로그램의 source code와 실행할 준비가 된 executable binaries가 포함되어 있습니다.

가장 쉬운 방법은 "Hello world" 테스트를 수행하기 위해 executable binaries를 사용하는 것입니다. 그러나 4.2 섹션에 자세히 설명된 대로 이러한 프로그램의 compilation을 수행하는 것도 가능합니다.

또 다른 가능성은 4.3 섹션에서 설명한 대로 Linux와 유사한 작업 환경을 만드는 것입니다. 이 가능성을 선택한 경우 Getting started with Xillybus on a Linux host 가이드의 "Hello world" 테스트 지침을 따르십시오.

3.3 사소한 loopback 테스트

다음은 두 개의 Command Prompt 창을 사용한 loopback 테스트의 간단한 예입니다.

일반 Command Prompt 창을 열고 디렉터리를 Xillybus package for Windows의 "precompiled-demoapps" 하위 디렉터리로 변경합니다. 자신의 compilation의 결과를 사용하려면(4.2 섹션 참조) 디렉토리를 대신 XP32_DEBUG로 변경하십시오.

이 Command Prompt 창에 다음을 입력하십시오.

```
> streamread \\.\xillybus_read_8
```

이렇게 하면 "streamread" 프로그램이 xillybus_read_8 device file에서 읽은 모든 내용을 출력합니다. 이 단계에서는 아무 일도 일어나지 않을 것으로 예상됩니다.

backslashes는 중복되지 않습니다. 이 작업이 Cygwin(일반 Command Prompt 창에서가 아님)로 수행된 경우 대신 `\\\\.\\xillybus_read_8`이 됩니다.

이제 다른 Command Prompt 창을 엽니다. 첫 번째 Command Prompt와 동일한 디렉터리로 변경하고 다음을 입력합니다.

```
> streamwrite \\.\xillybus_write_8
```

이 명령은 두 번째 창에 입력된 모든 내용을 device file(예: `\\.\xillybus_write_8`)로 보냅니다.

두 번째 Command Prompt 창에 텍스트를 입력하고 ENTER를 누릅니다. 동일한 텍스트

가 첫 번째 Command Prompt 창에 나타납니다. ENTER를 누르기 전까지는 아무 것도 전송되지 않습니다. 이는 standard input의 예상 동작과 일치합니다.

이 두 명령을 시도하는 동안 오류 메시지가 수신되면 다음 조치가 제안됩니다.

- 오타를 확인하십시오.
- driver가 설치되어 있고 FPGA가 Xillybus 장치로 감지되는지 확인하십시오. Device Manager를 열고 2.1 섹션의 마지막 이미지와 비교합니다.
- 2.3 섹션에 설명된 대로 Event Viewer의 오류를 확인합니다.
- 2.2 섹션에 설명된 대로 device files가 생성되었는지 확인합니다(`xillybus_read_8` 및 `xillybus_write_8` 검색).

FPGA 내부의 FIFOs는 overflow나 underflow에 대해 위험하지 않습니다. core는 FPGA 내부의 'full' 및 'empty' 신호를 준수합니다. 필요한 경우 Xillybus driver는 FIFO가 I/O를 사용할 준비가 될 때까지 컴퓨터 프로그램을 강제로 대기시킵니다. 이를 blocking라고 하며 user space program을 강제로 절전 모드로 전환하는 것을 의미합니다.

또한 streamwrite는 각 행에서 개별적으로 작동하며 ENTER를 누르기 전에는 FPGA로 아무 것도 보내지 않습니다. 이것은 Linux에 대해 같은 이름을 가진 프로그램과 다릅니다.

그들 사이에 loopback이 있는 또 다른 쌍의 device files가 있습니다.xillybus_read_32 및 xillybus_write_32. 이러한 device files는 32비트 워드로 작동하며 이는 FPGA 내부의 FIFO에도 해당됩니다. 따라서 이러한 device files를 사용한 "hello world" 테스트는 다음과 같은 한 가지 차이점을 제외하고 유사한 동작을 나타냅니다. 모든 I/O는 4바이트 그룹으로 수행됩니다. 따라서 입력이 4바이트 경계에 도달하지 않은 경우 입력의 마지막 바이트는 전송되지 않은 상태로 유지됩니다.

3.4 메모리 인터페이스

memread 및 memwrite 프로그램은 FPGA의 메모리에 액세스하는 방법을 보여주기 때문에 더 흥미롭습니다. 이는 device file에서 _lseek()에 대한 함수 호출을 통해 달성됩니다. Xillybus host application programming guide for Windows에는 Xillybus의 device files와 관련하여 이 API를 설명하는 섹션이 있습니다.

demo bundle에서는 xillybus_mem_8만 seeking을 허용합니다. 이 device file은 또한 읽기와 쓰기 모두를 위해 열 수 있는 유일한 제품입니다.

이 섹션에서는 "hexdump"라는 도구를 사용하여 FPGA의 RAM 콘텐츠를 표시합니다. 이 도구는 Xillybus package for Windows의 "unixutils" 하위 디렉토리에서 찾을 수 있습니다. 또는 섹션 4.3는 이 도구를 얻기 위한 다른 옵션을 제안합니다.

메모리에 쓰기 전에 hexdump을 사용하여 기존 상황을 관찰할 수 있습니다.

```
> hexdump -C -v -n 32 \\.\xillybus_mem_8
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020
```

이 출력은 메모리 배열의 처음 32바이트입니다.hexdump은 xillybus_mem_8을 열고 이 device file에서 32바이트를 읽습니다. _lseek()을 허용하는 파일을 열 때 초기 위치는 항상 0입니다. 따라서 출력은 위치 0에서 위치 31까지 메모리 배열의 데이터로 구성됩니다.

출력이 다를 수 있습니다. 이 출력은 다른 값을 포함할 수 있는 FPGA의 RAM을 반영합니다. 특히 이러한 값은 RAM을 사용한 이전 실험의 결과로 0이 아닐 수 있습니다.

hexdump의 flags에 대한 몇 마디: 위에 표시된 출력 형식은 “-C” 및 “-v”의 결과입니다. “-n 32”는 처음 32바이트만 표시한다는 의미입니다. 메모리 배열의 길이는 32바이트에 불과하므로 그 이상을 읽는 것은 의미가 없습니다.

memwrite는 어레이의 값을 변경하는 데 사용할 수 있습니다. 예를 들어 주소 3의 값은 다음 명령을 사용하여 170(hex 형식의 0xaa)로 변경됩니다.

```
> memwrite \\.\xillybus_mem_8 3 170
```

명령이 작동하는지 확인하기 위해 위에서 hexdump 명령을 반복할 수 있습니다.

```
> hexdump -C -v -n 32 \\.\xillybus_mem_8
00000000  00 00 00 aa 00 00 00 00  00 00 00 00 00 00 00 00  |...Ãª............|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020
```

분명히 명령이 작동했습니다.

memwrite.c에서 중요한 부분은 “_lseek(fd, address, SEEK_SET)”라고 적힌 부분입니다. 이 함수 호출은 device file의 위치를 변경합니다. 결과적으로 이것은 FPGA 내부에서 액세스되는 어레이 요소의 주소를 변경합니다. 후속 읽기 작업 또는 쓰기 작업은 이 위치에서 시작됩니다. 이러한 각 액세스는 전송된 바이트 수에 따라 위치를 증가시킵니다.

seeking을 허용하는 device file은 FPGA에 구성 명령을 쉽게 보내는 데도 유용합니다. 이는 FPGA에 memory array 대신 registers를 배치함으로써 이루어집니다. 다음은 2 주소에 대한 쓰기 작업에 대한 응답으로 새 값을 수신하는 register의 예입니다.

```
reg [7:0] my_register;

always @(posedge bus_clk)
  if (user_w_mem_8_wren && (user_mem_8_addr == 2))
    my_register <= user_w_mem_8_data;
```

마찬가지로 FPGA의 logic에서 “case” 문을 사용하여 registers의 값을 다시 읽는 것이 가능합니다.

4

host 애플리케이션의 예

4.1 일반적인

Xillybus의 device files에 액세스하는 방법을 보여주는 6개의 C 프로그램이 있습니다. 이러한 프로그램은 driver를 제공하는 동일한 웹 페이지에서 다운로드할 수 있는 zip 파일인 Xillybus package for Windows에서 찾을 수 있습니다.

이 파일의 “precompiled-demoapps” 하위 디렉토리에는 precompiled executables가 있습니다.

source code는 “demoapps” 하위 디렉토리에서 찾을 수 있습니다. 이 C 프로그램은 Microsoft의 SDK의 일부로 무료로 다운로드할 수 있는 Microsoft’s Visual C++ compiler용입니다. 이 프로그램은 Visual Studio에서도 사용할 수 있습니다.

Cygwin 내에서 샘플 프로그램을 실행할 수도 있습니다(4.3 섹션 참조). MinGW도 이 용도로 사용할 수 있습니다. 이 두 도구 중 하나를 선택하면 Linux용 source code를 사용해야 하며 Getting started with Xillybus on a Linux host의 지침을 따라야 합니다. 또한 /dev/ 접두사를 `\\\\.\\`로 대체하는 것과 관련된 4.4 섹션을 참조하십시오.

“demoapps” 하위 디렉토리는 다음 파일로 구성됩니다.

- Makefile– 이 파일에는 프로그램의 compilation을 위해 “nmake” 유틸리티에서 사용하는 규칙이 포함되어 있습니다.
- streamread.c– 파일에서 읽고 데이터를 standard output로 보냅니다.
- streamwrite.c– standard input에서 데이터를 읽고 파일로 보냅니다.
- memread.c– seek을 수행한 후 데이터를 읽습니다. FPGA에서 메모리 인터페이스에 액세스하는 방법을 보여줍니다.

- memwrite.c– seek을 수행한 후 데이터를 씁니다. FPGA에서 메모리 인터페이스에 액세스하는 방법을 보여줍니다.
- fifo.c– userspace RAM FIFO 구현을 보여줍니다. device file의 RAM buffers는 거의 모든 시나리오에 충분하도록 구성할 수 있기 때문에 이 프로그램은 거의 유용하지 않습니다. 따라서 fifo.c는 매우 높은 데이터 전송률과 RAM buffer가 매우 커야 하는 경우(예: 여러 gigabytes)에만 유용합니다.
- winstreamread.c– 파일에서 읽고 데이터를 standard output로 보냅니다. 이 프로그램은 streamread.c와 동일하지만 winstreamread.c는 표준 API 대신 Microsoft의 file I/O API를 사용하여 시연합니다.

winstreamread.c를 제외한 모든 프로그램은 Microsoft의 compiler와 함께 compilation용으로 만들어졌음에도 불구하고 고전적인 Linux 스타일로 작성되었습니다.

이 프로그램의 목적은 올바른 coding style을 표시하는 것입니다. 또한 자신의 프로그램을 작성하기 위한 기초로 사용할 수도 있습니다. 그러나 이러한 프로그램은 특히 이러한 프로그램이 높은 데이터 속도에서 잘 수행되지 않기 때문에 실제 응용 프로그램에서 사용하기 위한 것이 아닙니다. 고대역폭 성능 달성에 대한 지침은 5 장을 참조하십시오.

이러한 프로그램은 매우 단순하며 파일에 액세스하는 표준 방법을 보여줄 뿐입니다. 이러한 방법은 Xillybus host application programming guide for Windows에서 자세히 설명합니다. 이러한 이유로 여기에는 이러한 프로그램에 대한 자세한 설명이 없습니다.

이러한 프로그램은 하위 수준 API(예: _open(), _read() 및 _write())를 사용합니다. 더 잘 알려진 API(fopen(), fread(), fwrite() 등)는 C runtime library에 의해 유지되는 data buffers에 의존하기 때문에 피합니다. 이러한 data buffers는 특히 FPGA와의 통신이 종종 runtime library에 의해 지연되기 때문에 혼동을 일으킬 수 있습니다.

4.2 Compilation

이미 언급했듯이 예제 프로그램을 시험하기 위해 compilation이 필요하지 않습니다. Xillybus package for Windows에는 Windows 컴퓨터에서 실행할 준비가 된 파일이 포함되어 있습니다. 그러나 분명히 이러한 프로그램의 compilation은 변경이 필요합니다.

Microsoft Visual Studio로 작업하는 데 익숙한 사용자는 아마도 이 compiler를 선호하고 이 도구를 사용하는 방법을 알고 있을 것입니다. 예제 프로그램은 간단한 Command Prompt 애플리케이션입니다.

그러나 아래 지침은 Microsoft의 software development kit (SDK) 7.1을 기준으로 합니다. 이것은 무료로 다운로드할 수 있는 오래되고 단순한 개발 키트입니다. 아래 지침은 주로 작업을 수행하는 데 필요한 단계가 적기 때문에 이 소프트웨어를 기반으로 합니다.

Windows SDK 7.1을 다운로드하여 설치합니다. “Start menu”에서 Program Files를 열고 Microsoft Windows SDK v7.1 > Windows SDK 7.1 Command Prompt를 선택합니다. 그러면 compilation용으로 구성된 여러 environment variables가 있는 Command Prompt 창이 열립니다. 이 창의 텍스트는 노란색 글꼴로 표시됩니다.

C 파일이 있는 디렉토리로 변경:

```
> cd \path\to\demoapps
```

모든 프로그램에서 compilation을 실행하려면 “nmake”을 입력합니다. 다음 성적표가 예상됩니다.

```
> nmake

Microsoft (R) Program Maintenance Utility Version 10.00.30319.01
Copyright (C) Microsoft Corporation.  All rights reserved.

    if not exist "XP32_DEBUG/" mkdir XP32_DEBUG
    cl -D_CRT_SECURE_NO_WARNINGS -c -DCRTAPI1=_cdecl -DCRTAPI2=_cdecl -nologo  [ ..
    link  /INCREMENTAL:NO /NOLOGO -subsystem:console,5.01 -out:XP32_DEBUG\ [ ... ]
    cl -D_CRT_SECURE_NO_WARNINGS -c -DCRTAPI1=_cdecl -DCRTAPI2=_cdecl -nologo  [ ..
    link  /INCREMENTAL:NO /NOLOGO -subsystem:console,5.01 -out:XP32_DEBUG\ [ ... ]
    cl -D_CRT_SECURE_NO_WARNINGS -c -DCRTAPI1=_cdecl -DCRTAPI2=_cdecl -nologo  [ ..
    link  /INCREMENTAL:NO /NOLOGO -subsystem:console,5.01 -out:XP32_DEBUG\ [ ... ]
    cl -D_CRT_SECURE_NO_WARNINGS -c -DCRTAPI1=_cdecl -DCRTAPI2=_cdecl -nologo  [ ..
    link  /INCREMENTAL:NO /NOLOGO -subsystem:console,5.01 -out:XP32_DEBUG\ [ ... ]
    cl -D_CRT_SECURE_NO_WARNINGS -c -DCRTAPI1=_cdecl -DCRTAPI2=_cdecl -nologo  [ ..
    link  /INCREMENTAL:NO /NOLOGO -subsystem:console,5.01 -out:XP32_DEBUG\ [ ... ]
    cl -D_CRT_SECURE_NO_WARNINGS -c -DCRTAPI1=_cdecl -DCRTAPI2=_cdecl -nologo  [ ..
    link  /INCREMENTAL:NO /NOLOGO -subsystem:console,5.01 -out:XP32_DEBUG\ [ ... ]
```

“cl”로 시작하는 6개의 행은 compiler를 사용하기 위해 “nmake”에서 요청하는 명령입니다. 이 명령은 프로그램의 compilation에 별도로 사용할 수 있습니다. 그러나 그렇게 할 이유가 없습니다. 그냥 “nmake” 쓰세요. object files 및 libraries에서 linking을 수행하여 executables를 생성하는 “link” 명령도 마찬가지입니다.

executables(및 object files)는 XP32_DEBUG 하위 디렉토리에서 찾을 수 있습니다. 이 하위 디렉토리는 필요한 경우 compilation 프로세스 중에 생성됩니다. 디렉토리 이름에서 알 수 있듯이 이러한 파일은 32-bit Windows XP용입니다. 그러나 executables는 64비트 버전을 포함하여 이후 버전의 Windows에서 실행됩니다.

“nmake” 유틸리티는 필요한 경우에만 compilation을 실행합니다. 하나의 파일만 변경되면 “nmake”은 해당 파일의 compilation만 요청합니다. 따라서 정상적인 작업 방법은 편

집하려는 파일을 편집한 다음 recompilation용 “nmake”을 사용하는 것입니다. 불필요한 compilation이 발생하지 않습니다.

이전 compilation에서 생성된 executables를 제거하려면 “nmake clean”을 사용하십시오.

위에서 언급했듯이 Makefile에는 compilation의 규칙이 포함되어 있습니다. 이 파일의 구문은 간단하지 않지만 다행스럽게도 상식적으로만 사용하여 이 파일을 변경할 수 있는 경우가 많습니다.

Makefile은 Makefile 자체와 동일한 디렉토리에 있는 파일과 관련이 있습니다. 따라서 전체 디렉터리의 복사본을 만들고 이 복제본 내부에 있는 파일에 대해 작업할 수 있습니다. 디렉토리의 두 사본은 서로 간섭하지 않습니다.

또한 C 파일을 추가하고 Makefile을 쉽게 변경하여 “nmake”도 이 새 파일의 compilation을 실행할 수 있습니다.

4.3 Windows와 함께 Linux의 도구 사용

Linux로 작업하는 사람들은 간단한 작업을 수행하기 위해 표준 command line 도구를 사용하는 경향이 있습니다. 이러한 도구는 주로 Windows를 사용하는 사람들 사이에서 덜 알려져 있습니다. 주된 이유는 Windows용 command-line 도구가 불행히도 모든 Linux 컴퓨터에 있는 도구만큼 유용하지 않기 때문입니다.

이미 언급했듯이 이 안내서의 지침 대신 Getting started with Xillybus on a Linux host에 설명된 대로 “Hello world” 테스트를 수행할 수 있습니다. Windows에서 이를 수행하려면 컴퓨터에서 몇 가지 도구를 사용할 수 있도록 해야 합니다. 이를 달성하기 위한 몇 가지 대안이 있습니다.

- Gnuwin32 프로젝트에서 두 개의 패키지를 다운로드하고 설치합니다. Coreutils 및 Util-Linux-NG. 이 두 패키지는 “Hello world” 테스트의 요구 사항을 충족합니다(또한 이 작업에 필요하지 않은 프로그램도 제공). 이러한 패키지가 Gnuwin32의 setup 도구와 함께 설치되더라도 Command Prompt의 execution path에는 변경 사항이 없습니다.
- Xillybus에서 제공하는 Windows용 도구를 사용하십시오. 이들은 Xillybus package for Windows의 “unixutils” 하위 디렉토리에서 찾을 수 있습니다. 이 방법으로 얻은 도구는 “Hello world” 테스트에 충분하도록 Gnuwin32 패키지에서 선택되었습니다.
- Cygwin을 설치합니다. 이 방법을 선택한다는 것은 Linux와 유사한 command-line 인터페이스를 제공하는 전체 시스템을 설치하는 것을 의미합니다. 이러한 종류의 설치에는 GNU C compiler 및 기타 소프트웨어 개발 도구가 포함될 수 있습니다. 이것은 Linux의 command-line로 작업하는 데 익숙한 사용자에게 권장되는 선택입니다.

4.4 Linux와의 차이점

Windows 컴퓨터에서 Getting started with Xillybus on a Linux host에 설명된 대로 “Hello world” 테스트를 수행할 때 알아야 할 몇 가지 차이점이 있습니다.

가장 중요한 차이점은 path와 device files가 `/dev/`가 아닌 `\\.\`라는 것입니다. 예를 들어 Linux에 대한 가이드에서 `/dev/xillybus_read_8`을 언급할 때 Windows의 올바른 파일 이름은 `\\.\xillybus_read_8`입니다.

device file의 이름에 backslashes가 포함되어 있기 때문에 일부 상황에서는 escape characters가 필요합니다. backslash 자체는 종종 escape character로 취급됩니다. 따라서 파일 이름의 각 backslash에 대해 두 개의 backslashes가 필요합니다. 즉, `\\.\`는 `\\\\.\\`로 작성해야 합니다. 예를 들어 Linux에 대한 가이드에서 `/dev/xillybus_read_8`을 언급할 때 일부 상황에서는 파일 이름 `\\\\.\\xillybus_read_8`을 사용해야 합니다.

그러나 항상 그런 것은 아닙니다. Command Prompt에서 프로그램을 실행할 때 escape characters가 필요하지 않습니다. Command Prompt는 backslash를 다른 character와 마찬가지로 취급합니다.

대부분의 프로그래밍 언어에는 추가 backslashes가 필요합니다. scripts 내부에는 추가 backslashes가 필요할 수 있습니다. 이는 arguments가 script 내부에서 처리되는 방식에 따라 다릅니다.

4.5 Cygwin’ 경고 메시지

Cygwin의 command-line 인터페이스에는 추가 백슬래시가 필요합니다. 그러나 `\\\\.\\` 접두사를 처음 사용하는 경우 Cygwin은 다음과 같은 경고를 표시할 수 있습니다.

```
$ cat \\\\.\\xillybus_read_8
cygwin warning:
  MS-DOS style path detected: \\.\xillybus_read_8
  Preferred POSIX equivalent is: //./xillybus_read_8
  CYGWIN environment variable option "nodosfilewarning" turns off this warning.
  Consult the user's guide for more details about POSIX paths:
    \url{http://cygwin.com/cygwin-ug-net/using.html#using-pathnames}
```

이 경고는 무시해야 합니다.

Xillybus는 Cygwin로 광범위하게 테스트되었으며 device files에 액세스하기 위해 위에 표시된 방법이 정확합니다. 일반 파일 이름의 경우 슬래시를 사용하는 것이 실제로 더 좋습니다. 그러나 Cygwin은 `//./`를 `\\.\`로 변환하지 않습니다. 따라서 backslashes의 사용은 필수입니다.

이 경고를 피하려면 environment variable에 관한 경고 메시지의 제안을 따를 수 있습니다.

5

고대역폭 성능을 위한 지침

Xillybus의 IP cores 사용자는 광고된 데이터 전송 속도가 실제로 충족되는지 확인하기 위해 종종 데이터 대역폭 테스트를 수행합니다. 이러한 목표를 달성하려면 데이터 흐름을 상당히 저하시킬 수 있는 병목 현상을 피해야 합니다.

이 섹션은 가장 일반적인 실수를 기반으로 하는 지침 모음입니다. 이 가이드라인을 따르면 게시된 것과 같거나 약간 더 나은 대역폭 측정 결과가 나와야 합니다.

이 프로젝트가 IP core의 전체 기능을 활용하도록 Xillybus를 기반으로 하는 프로젝트를 구현할 때 이러한 지침을 따르는 것이 물론 중요합니다.

종종 문제는 host가 데이터를 충분히 빠르게 처리하지 않는다는 것입니다. 데이터 속도를 잘못 측정하는 것은 게시된 숫자를 얻을 수 없다는 불만이 발생하는 가장 일반적인 이유입니다. 권장되는 방법은 아래 섹션 5.3에 표시된 대로 Linux의 “dd” 명령을 사용하는 것입니다. 이 도구는 Xillybus package for Windows에서 executable로 사용할 수 있습니다.

이 섹션의 정보는 “Getting Started” 가이드에 대해 상대적으로 고급입니다. 이 토론에서는 다른 문서에서 설명하는 고급 항목도 참조합니다. 그럼에도 불구하고 많은 사용자가 IP core에 익숙해지는 초기 단계에서 성능 테스트를 수행하기 때문에 이 가이드에 이러한 지침이 제공됩니다.

5.1 loopback 하지마

demo bundle(FPGA 내부)에는 두 쌍의 streams 사이에 loopback이 있습니다. 이렇게 하면 “Hello, world” 테스트가 가능하지만(3 섹션 참조) 성능 테스트에는 좋지 않습니다.

문제는 Xillybus IP core가 데이터 전송 버스트로 매우 빠르게 FPGA 내부의 FIFO를 채운다는 것입니다. 이 FIFO가 가득 차서 데이터 흐름이 일시적으로 중지됩니다.

loopback은 이 FIFO로 구현되었으므로 이 FIFO의 양쪽이 IP core에 연결됩니다. FIFO에 데이터가 있으면 IP core는 FIFO에서 이 데이터를 가져와 host로 다시 보냅니다. 이 역시

매우 빠르게 발생하므로 FIFO가 비게 됩니다. 다시 한 번 데이터 흐름이 일시적으로 중지됩니다.

이러한 데이터 흐름의 일시적인 일시 중지로 인해 측정된 데이터 전송 속도가 예상보다 낮습니다. 이는 FIFO가 너무 얕고 IP core가 FIFO를 채우고 비우는 일을 모두 담당하기 때문에 발생합니다.

실제 시나리오에는 loopback이 없습니다. 오히려 FIFO의 반대편에는 application logic이 있습니다. 최대 데이터 전송 속도를 달성하는 사용 시나리오를 고려해 보겠습니다. 이 시나리오에서 application logic은 IP core가 이 FIFO를 채우는 속도만큼 빠르게 FIFO의 데이터를 사용합니다. 따라서 FIFO는 가득 차지 않습니다.

반대 방향도 마찬가지입니다. application logic은 IP core가 데이터를 소비하는 속도만큼 빠르게 FIFO를 채웁니다. 따라서 FIFO는 결코 비어 있지 않습니다.

기능적인 관점에서 볼 때 FIFO가 때때로 가득 차거나 비어 있는 것은 문제가 되지 않습니다. 이로 인해 데이터 흐름이 일시적으로 중단될 뿐입니다. 최대 속도가 아닌 모든 것이 올바르게 작동합니다.

demo bundle은 성능 테스트를 위해 쉽게 수정할 수 있습니다. 예를 들어 `\\.\xillybus_read_32`를 테스트하려면 FPGA 내부의 FIFO에서 user_r_read_32_empty를 분리합니다. 대신 이 signal을 상수 0에 연결하십시오. 결과적으로 IP core는 FIFO가 결코 비어 있지 않다고 생각할 것입니다. 따라서 데이터 전송은 최대 속도로 수행됩니다.

이것은 IP core가 때때로 빈 FIFO에서 읽을 수 있음을 의미합니다. 결과적으로 host에 도착하는 데이터가 항상 유효한 것은 아닙니다(underflow로 인해). 그러나 속도 테스트의 경우 이것은 중요하지 않습니다. 데이터의 내용이 중요한 경우 가능한 해결책은 application logic이 FIFO를 가능한 한 빨리 채우는 것입니다(예: counter의 출력으로).

마찬가지로 `\\.\xillybus_write_32`를 테스트할 때도 마찬가지입니다. FIFO에서 user_w_write_32_full을 분리하고 이 signal을 상수 0에 연결합니다. IP core는 FIFO가 절대 가득 차지 않는다고 생각하므로 데이터 전송이 최대 속도로 수행됩니다. FIFO로 전송되는 데이터는 overflow로 인해 부분적으로 손실됩니다.

loopback을 분리하면 각 방향을 개별적으로 테스트할 수 있습니다. 그러나 이것은 양방향을 동시에 테스트하는 올바른 방법이기도 합니다.

5.2 디스크 또는 기타 저장소를 포함하지 마십시오.

디스크, solid-state drives 및 기타 종류의 컴퓨터 스토리지는 종종 대역폭 기대치를 충족하지 못하는 이유입니다. 저장 매체의 속도를 과대평가하는 것은 흔한 실수입니다.

운영 체제의 cache 메커니즘은 혼란을 가중시킵니다. 데이터가 디스크에 기록될 때 물리적 저장 매체가 항상 관련되는 것은 아닙니다. 대신 데이터가 RAM에 기록됩니다. 나중에

야 이 데이터가 디스크 자체에 기록됩니다. 디스크에서 읽기 작업이 물리적 매체와 관련되지 않을 수도 있습니다. 동일한 데이터를 최근에 이미 읽은 경우에 발생합니다.

cache는 최신 컴퓨터에서 매우 클 수 있습니다. 따라서 디스크의 실제 속도 제한이 표시되기 전에 여러 Gigabytes 데이터가 흐를 수 있습니다. 이로 인해 사용자는 종종 Xillybus의 데이터 전송에 문제가 있다고 생각하게 됩니다. 데이터 전송 속도의 갑작스러운 변화에 대한 다른 설명은 없습니다.

solid-state drives(flash)를 사용하면 특히 길고 연속적인 쓰기 작업 중에 추가적인 혼란의 원인이 있습니다. flash drive의 저수준 구현에서 메모리의 사용되지 않는 세그먼트(blocks)는 flash에 쓰기 위한 준비로 지워야 합니다. flash memory에 대한 데이터 쓰기는 지워진 blocks에만 허용되기 때문입니다.

시작점으로 flash drive에는 일반적으로 이미 지워진 blocks가 많이 있습니다. 이렇게 하면 쓰기 작업이 빨라집니다. 데이터를 쓸 수 있는 공간이 많습니다. 그러나 지워진 blocks가 더 이상 없으면 flash drive는 강제로 blocks를 지우고 데이터의 defragmentation을 수행할 수 있습니다. 이로 인해 명백한 설명이 없는 상당한 속도 저하가 발생할 수 있습니다.

이러한 이유로 Xillybus의 대역폭 테스트에는 저장 매체가 포함되어서는 안 됩니다. 짧은 테스트 동안 저장 매체가 충분히 빠른 것처럼 보이더라도 이는 잘못된 것일 수 있습니다.

Xillybus device file에서 디스크의 큰 파일로 데이터를 복사하는 데 걸리는 시간을 측정하여 성능을 추정하는 것은 일반적인 실수입니다. 이 작업이 기능적으로 올바르더라도 이러한 방식으로 성능을 측정하면 완전히 잘못된 것으로 판명될 수 있습니다.

저장소가 응용 프로그램(예: data acquisition)의 일부로 의도된 경우 이 저장소 매체를 철저히 테스트하는 것이 좋습니다. 저장 매체에 대한 광범위하고 장기적인 테스트를 통해 기대치를 충족하는지 확인해야 합니다. 짧은 benchmark test는 매우 오해의 소지가 있습니다.

5.3 많은 부분 읽기 및 쓰기

_read() 및 _write()에 대한 각 함수 호출은 운영 체제에 대한 system call로 이어집니다. 따라서 이러한 함수 호출을 수행하려면 많은 CPU cycles가 필요합니다. 따라서 buffer의 크기가 충분히 커서 더 적은 수의 system calls가 수행되는 것이 중요합니다. 이는 대역폭 테스트와 고성능 애플리케이션에 해당됩니다.

일반적으로 128 kB는 각 함수 호출의 buffer에 적합한 크기입니다. 이는 각 함수 호출이 최대 128 kB로 제한됨을 의미합니다. 그러나 이러한 함수 호출은 더 적은 데이터를 전송할 수 있습니다.

섹션 4.1 및 3.3(streamread 및 streamwrite)에서 언급한 예제 프로그램은 성능 측정에 적합하지 않다는 점에 유의해야 합니다. 이러한 프로그램의 buffer 크기는 128바이트입니

다(kB 아님). 이것은 예제를 단순화하지만 성능 테스트를 하기에는 프로그램을 너무 느리게 만듭니다.

다음 shell 명령은 빠른 속도 확인을 위해 Cygwin 내에서 사용할 수 있습니다(필요한 경우 `\\\\.\\xillybus\_*` 이름 교체).

```
dd if=/dev/zero of=\\\\.\\xillybus_sink bs=128k
dd if=\\\\.\\xillybus_source of=/dev/null bs=128k
```

이러한 명령은 CTRL-C로 중지될 때까지 실행됩니다. 고정된 양의 데이터에 대한 테스트를 수행하려면 “count=”를 추가하십시오.

테스트를 위한 Cygwin 사용에 대한 자세한 내용은 섹션 4.3를 참조하십시오.

5.4 CPU 소비에 주의

데이터 전송률이 높은 응용 프로그램에서 컴퓨터 프로그램은 종종 병목 지점이며 반드시 데이터 전송은 아닙니다.

일반적인 실수는 CPU의 기능을 과대평가하는 것입니다. 일반적인 믿음과 달리 데이터 속도가 100-200 MB/s를 초과하면 가장 빠른 CPUs도 데이터로 의미 있는 작업을 수행하는 데 어려움을 겪습니다. multi-threading로 성능을 향상시킬 수 있지만 이것이 필요하다는 사실에 놀랄 수도 있습니다.

때때로 buffers의 크기가 부적절하면(위에서 언급한 대로) CPU가 과도하게 소모될 수도 있습니다.

따라서 CPU 소비량을 주시하는 것이 중요합니다. 예를 들어 Task Manager를 이 용도로 사용할 수 있습니다. 그러나 이 프로그램이 표시하는 정보는 여러 processor cores가 있는 컴퓨터(즉, 오늘날 거의 모든 컴퓨터)에서 오해의 소지가 있을 수 있습니다. 예를 들어 processor cores가 4개 있다면 25% CPU는 무엇을 의미할까요? 낮은 CPU 소비입니까, 아니면 특정 thread의 100%입니까? system monitoring의 다른 도구는 이 정보를 다른 방식으로 표시합니다.

5.5 읽기와 쓰기를 상호 의존적으로 만들지 마십시오.

양방향 통신이 필요할 때 단 하나의 thread로 컴퓨터 프로그램을 작성하는 것은 일반적인 실수입니다. 이 프로그램은 일반적으로 읽기와 쓰기를 수행하는 하나의 루프를 가지고 있습니다. 반복할 때마다 데이터가 FPGA 쪽으로 쓰여진 다음 반대 방향으로 데이터가 읽힙니다.

예를 들어 두 개의 streams가 기능적으로 독립적인 경우 이와 같은 프로그램에는 문제가 없는 경우가 있습니다. 그러나 이와 같은 프로그램의 의도는 종종 FPGA가 coprocessing을

수행해야 한다는 것입니다. 이 프로그래밍 스타일은 프로그램이 처리를 위해 데이터의 일부를 보낸 다음 결과를 다시 읽어야 한다는 오해에 기반합니다. 따라서 반복은 데이터의 각 부분 처리를 구성합니다.

이 방법은 비효율적일 뿐만 아니라 프로그램이 중단되는 경우가 많습니다. Xillybus host application programming guide for Windows 의 섹션 6.5에서는 이 주제에 대해 더 자세히 설명하고 보다 적절한 프로그래밍 기술을 제안합니다.

5.6 host의 RAM 한계 알아보기

이것은 주로 revision XL / XXL IP core를 사용할 때 관련이 있습니다. 마더보드(또는 embedded processor)와 DDR RAM 사이에는 제한된 데이터 대역폭이 있습니다. 이 제한은 컴퓨터의 일반적인 사용에서 거의 발견되지 않습니다. 그러나 Xillybus를 사용하는 매우 까다로운 애플리케이션의 경우 이 제한이 병목 현상이 될 수 있습니다.

FPGA에서 user space program로 데이터를 전송할 때마다 RAM에서 두 가지 작업이 필요합니다. 첫 번째 작업은 FPGA가 데이터를 DMA buffer에 쓸 때입니다. 두 번째 작업은 driver가 이 데이터를 user space program에서 액세스할 수 있는 buffer로 복사하는 것입니다. 유사한 이유로 데이터가 반대 방향으로 전송될 때도 RAM에서 두 가지 작업이 필요합니다.

운영 체제에서 DMA buffers와 user space buffers를 분리해야 합니다. _read() 및 _write()(또는 유사한 함수 호출)를 사용하는 모든 I/O는 이러한 방식으로 수행되어야 합니다.

예를 들어, XL IP core의 테스트는 각 방향에서 3.5 GB/s, 즉 총 7 GB/s가 될 것으로 예상됩니다. 그러나 RAM은 두 배로 액세스됩니다. 따라서 RAM의 대역폭 요구 사항은 14 GB/s입니다. 모든 마더보드에 이 기능이 있는 것은 아닙니다. 또한 host는 동시에 다른 작업에 RAM을 사용합니다.

리비전 XXL을 사용하면 같은 이유로 한 방향의 간단한 테스트라도 RAM의 대역폭 용량을 초과할 수 있습니다.

5.7 충분히 큰 DMA buffers

이는 거의 문제가 되지 않지만 여전히 언급할 가치가 있습니다. DMA buffers용 host에 RAM이 너무 적게 할당되면 데이터 전송 속도가 느려질 수 있습니다. 그 이유는 host가 data stream을 작은 세그먼트로 나눌 수밖에 없기 때문입니다. 이로 인해 CPU cycles가 낭비됩니다.

모든 demo bundles에는 성능 테스트를 위한 충분한 DMA 메모리가 있습니다. 이는 IP Core Factory에서 올바르게 생성된 IP cores의 경우에도 마찬가지입니다. “Autoset Inter-

nals"가 활성화되고 "Expected BW"는 필요한 데이터 대역폭을 반영합니다. "Buffering"은 10 ms로 선택해야 합니다. 어떤 옵션이든 괜찮을 것입니다.

일반적으로 이것은 대역폭 테스트에 충분합니다. 10 ms 동안 데이터 전송에 해당하는 RAM의 총량이 있는 최소 4개의 DMA buffers. 물론 필요한 데이터 전송 속도를 고려해야 합니다.

5.8 데이터 워드에 올바른 너비를 사용하십시오.

확실히 application logic은 FPGA 내부의 각 clock cycle에 대해 IP core로 한 단어의 데이터만 전송할 수 있습니다. 따라서 데이터 워드의 폭과 bus_clk의 주파수로 인해 데이터 전송 속도에 제한이 있습니다.

또한 기본 개정판(revision A IP cores)이 있는 IP cores와 관련된 제한 사항이 있습니다. 워드 폭이 8비트 또는 16비트일 때 PCIe의 기능은 워드 폭이 32비트일 때만큼 효율적으로 사용되지 않습니다. 따라서 고성능이 필요한 응용 프로그램 및 테스트에서는 32비트만 사용해야 합니다. 이것은 revision B IP cores 및 이후 버전에는 적용되지 않습니다.

워드 너비는 리비전 B부터 최대 256비트가 될 수 있습니다. 단어의 너비는 최소한 PCIe block의 너비만큼 넓어야 합니다. 따라서 데이터 대역폭 테스트의 경우 다음 데이터 워드 너비가 필요합니다.

- 기본 개정판(Revision A): 32비트.
- Revision B: 최소 64비트.
- Revision XL: 최소 128비트.
- Revision XXL: 256비트.

데이터 단어가 위에서 요구한 것보다 더 넓은 경우(가능한 경우) 일반적으로 약간 더 나은 결과를 얻습니다. 그 이유는 application logic와 IP core 간의 데이터 전송이 개선되었기 때문입니다.

5.9 매개변수 조정

광고된 데이터 전송 속도를 지원하기 위해 demo bundles의 PCIe block 매개변수가 선택됩니다. 성능은 x86 제품군에 속하는 CPU가 있는 일반 컴퓨터에서 테스트되었습니다.

또한 IP Core Factory에서 생성되는 IP cores는 일반적으로 미세 조정이 필요하지 않습니다. "Autoset Internals"가 활성화되면 streams는 FPGA 리소스의 성능과 활용 사이에서 최적의 균형을 유지할 수 있습니다. 따라서 각 stream에 대해 요청된 데이터 전송 속도가 보장됩니다.

따라서 PCIe block 또는 IP core의 매개변수를 미세 조정하는 것은 거의 항상 무의미합니다. IP cores(revision A)의 기본 개정판에서는 이러한 조정이 항상 의미가 없습니다. 이러한 튜닝으로 성능이 향상된다면 문제는 application logic 또는 user application software의 결함일 가능성이 큽니다. 이 상황에서 이 결함을 수정하면 얻을 수 있는 것이 훨씬 더 많습니다.

그러나 예외적인 성능이 필요한 드문 시나리오에서는 요청된 데이터 속도를 얻기 위해 PCIe block의 매개변수를 약간 조정해야 할 수도 있습니다. 이것은 특히 host에서 FPGA까지의 streams와 관련이 있습니다. The guide to defining a custom Xillybus IP core의 섹션 4.5에서는 이 조정을 수행하는 방법에 대해 설명합니다.

이 미세 조정이 유익한 경우에도 수정되는 것은 Xillybus IP core의 매개변수가 아닙니다. PCIe block만 조정됩니다. IP core의 매개변수를 조정하여 데이터 전송 속도를 개선하려고 시도하는 것은 일반적인 실수입니다. 오히려 문제는 거의 항상 이 장에서 위에서 언급한 문제 중 하나입니다.

6

문제 해결

Xillybus / XillyUSB용 drivers는 시스템의 event log에서 의미 있는 log messages를 생성하도록 설계되었습니다. 따라서 뭔가 잘못된 것으로 보일 때 관련된 메시지를 검색하는 것이 좋습니다. 이는 2.3 섹션에 설명된 대로 event log에 필터를 적용하여 수행됩니다.

또한 모든 것이 잘 작동하는 것처럼 보이더라도 때때로 event log을 살펴보는 것이 좋습니다.

PCIe driver의 메시지 목록과 설명은 다음에서 찾을 수 있습니다.

http://xillybus.com/doc/list-of-kernel-messages

그러나 메시지 텍스트에서 Google을 사용하면 특정 메시지를 찾는 것이 더 쉬울 수 있습니다.