

(기계로 한국어 번역)

The guide to defining a custom Xillybus IP core

Xillybus Ltd.

www.xillybus.com

Version 3.3

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 This document may also be slightly outdated in relation to the original.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	3
1.1 일반적인	3
1.2 맞춤형 IP core를 통합하는 방법	4
2 사용자 정의 IP cores 정의	6
2.1 개요	6
2.2 device file의 이름	7
2.3 데이터 너비	7
2.4 Use	8
2.5 동기 또는 비동기 stream	9
2.6 버퍼링 시간	10
2.7 DMA buffers의 크기	11
2.8 DMA acceleration	13
3 확장성 및 logic 리소스 소비	14
3.1 일반적인	14
3.2 Block RAMs	14
3.3 logic fabric의 리소스	15
4 개정판 B, XL 및 XXL의 IP cores	18
4.1 일반적인	18
4.2 개정 B/XL/XXL 작업	19
4.3 데이터 워드의 너비	19
4.4 Logic 리소스 소비	20
4.5 host에서 FPGA로 stream의 최적 대역폭을 위한 조정	23

1

소개

1.1 일반적인

Xillybus는 다양한 애플리케이션을 위한 다목적 플랫폼입니다. 따라서 각 사용자는 streams의 수, 방향, 성능과 관련된 속성 및 리소스 소비와 같은 특정 요구 사항을 충족하는 맞춤형 IP core를 쉽게 생성하고 다운로드할 수 있습니다.

맞춤형 IP cores의 정의 및 생성을 단순화하기 위해 온라인 도구인 IP Core Factory(<https://xillybus.com/custom-ip-factory>)를 사용할 수 있습니다.

이 도구는 사용자가 요청된 device files 및 해당 구성을 정의할 수 있는 간단한 웹 애플리케이션으로 구성됩니다. 정의가 완료되면 자동 프로세스가 FPGA 프로젝트에 포함할 파일을 생성합니다. 맞춤형 IP core는 잠시 후(일반적으로 몇 분) zip 파일로 다운로드할 준비가 됩니다.

다운로드된 사용자 지정 IP core는 완전히 작동합니다. 실제 애플리케이션에서 이 IP core를 테스트하고 사용하는 데 기술적인 제한은 없습니다.

웹 애플리케이션은 이 가이드를 읽지 않아도 사용할 수 있지만, 먼저 demo bundle을 실행하여 Xillybus에 익숙해지는 것을 권장합니다. device files의 특성을 더 잘 이해하고 제어하려는 사용자는 이 가이드에서 몇 가지 배경 정보를 찾을 수 있습니다.

이 가이드에서는 Xillyp2p IP core에 대해서는 다루지 **않지만**, IP Core Factory에서도 Xillyp2p IP core를 생성할 수 있습니다. Xillyp2p에 대한 자세한 내용은 <https://xillybus.com/xillyp2p/>를 참조하십시오.

아직 demo bundle에 익숙하지 않은 사용자의 경우 다음 문서 중 일부를 미리 읽을 것을 권장합니다.

- [Getting started with the FPGA demo bundle for AMD](#)
- [Getting started with the FPGA demo bundle for Altera](#)

- [Getting started with Xilinx for Zynq-7000](#)
- [Getting started with Xillybus on a Linux host](#)
- [Getting started with Xillybus on a Windows host](#)

맞춤형 IP core의 필요성이 분명한 경우에도 demo bundle로 시작하는 것이 가장 좋습니다. 이것은 IP core가 application logic와 통합되는 방법과 전체 프로젝트를 설정하는 방법을 명확히 합니다.

IP core의 맞춤형 구성에 대한 모든 정보는 FPGA의 IP core 자체에 저장됩니다. host의 driver는 driver가 초기화될 때 이 정보를 검색합니다. 따라서 IP core를 교체할 때 host에서 아무 것도 변경할 필요가 없습니다.

일반적인 실수는 리소스를 절약하기 위해 구성된 streams의 수를 최소화하는 것입니다. 3 및 4.4 섹션은 Xillybus IP core가 어떻게 확장되는지 보여주고 streams를 관대하게 할당하는 것이 왜 합리적인지 설명합니다.

1.2 맞춤형 IP core를 통합하는 방법

IP Core Factory에서 맞춤형 IP core를 다운로드한 후 이 IP core를 포함하도록 demo bundle을 수정해야 합니다. 여기에는 몇 가지 간단한 단계가 필요합니다. 지침은 IP Core의 zip 파일의 일부인 README.TXT로 작성되었습니다. 이러한 지침은 편의를 위해 아래에 나열되어 있습니다.

이 README 파일에는 다음과 같은 유용한 정보도 포함되어 있습니다.

- 5자리 숫자인 Core ID입니다. 이 번호는 IP core의 고유 식별자입니다. 가격 견적을 요청할 때 Core ID를 언급해야 합니다.
- IP core의 devices files가 나열됩니다. 각 device file에 대한 기술 세부 정보도 표시됩니다. IP core의 실제 특성에 대한 정확한 정보입니다.

custom IP core를 demo bundle에 통합하려면 다음 단계를 따르십시오.

1. demo bundle에 있는 두 파일을 IP Core의 zip에 있는 파일인 xillybus.v 및 xillybus_core.v(또는 xillybus_xl_core.v / xillybus_xxl_core.v)로 바꿉니다.
2. IP core 자체를 교체하십시오. 이 파일은 "core/"라는 이름으로 demo bundle의 subdirectory에 있습니다. 교체할 파일은 xillybus_core.ngc, xillybus_core.edf, xillybus_core.qxp 또는 xillybus_core.vqm와 같은 것입니다.

- 원하는 응용 프로그램을 이 맞춤형 IP core와 통합하려면 xillydemo.v(또는 xillydemo.vhd)를 편집하십시오. 지침을 보려면 IP core의 zip 파일의 일부인 “instantiation templates”라는 디렉터리를 살펴보십시오. template.v(또는 template.vhd)라는 파일에는 따라야 하는 instantiation template가 포함되어 있습니다.

2

사용자 정의 IP cores 정의

2.1 개요

IP Core Factory는 사용자 정의 IP core를 처음부터 정의하거나 demo bundle의 core 구성을 시작점으로 사용하기 위한 wizard와 유사한 웹 애플리케이션입니다.

대부분의 목적을 위해 “Autoset internals” 옵션을 활성화된 상태로 유지하여 IP Core Factory에 의존하여 각 stream의 속성을 설정하는 것이 좋습니다. stream의 매개변수를 조정하기 위해 이 옵션을 끄는 것은 매우 흔한 실수이며, 이는 거의 항상 성능 저하로 이어 집니다.

특히 IP core가 예상 데이터 속도 성능을 충족하지 못한다면 문제가 다른 곳에 있을 가능성이 큼니다. 이 경우 IP core의 전체 성능을 달성하는 방법을 설명하는 다음 두 가이드 중 하나를 참조하는 것이 좋습니다.

- [Getting started with Xillybus on a Linux host](#)의 섹션 5
- [Getting started with Xillybus on a Windows host](#)의 섹션 5

또 다른 일반적인 실수는 DMA buffers의 크기를 조정하기 위해 “Autoset internals”를 꺼서 전송하려는 데이터 패킷의 크기와 일치하도록 하는 것입니다. 이것은 섹션 2.7에서 논의됩니다.

다음은 이 도구를 사용할 때 강조할 만한 몇 가지 추가 사항입니다.

- IP core가 netlist로 제공되기 때문에 IP core가 의도된 FPGA 제품군은 올바르게 선택되어야 합니다.
- 각 device file의 “use” 속성을 의도한 목적과 일치하는 설명으로 설정하는 것이 중요합니다. 이렇게 하면 stream의 속성이 올바르게 설정됩니다.

- XillyUSB IP cores의 경우 데이터 속도가 해당 값으로 제한되므로 “Expected bandwidth” 속성은 stream이 최대로 요청한 대역폭으로 정확하게 설정해야 합니다. 다른 변형(PCIe 및 AXI)의 경우 이 속성은 성능 조정에만 영향을 줍니다. 요구 사항을 과장하여 더 나은 결과를 얻으려고 하기보다는 실제 수치를 적용해야 합니다. 이러한 과장은 특정 제한된 리소스가 실제로 필요한 다른 streams의 성능 저하를 초래할 수 있습니다.

이 섹션의 나머지 부분에서는 device files의 속성 중 일부에 대해 설명합니다.

2.2 device file의 이름

각 stream은 host에서 생성된 device file의 이름으로 사용되는 이름으로 지정됩니다.

이름은 항상 xillybus_* 형식을 사용합니다(예: xillybus_mystream). XillyUSB의 경우 이름은 xillyusb_NN_*와 같습니다. 여기서 NN은 인덱스입니다. 일반적으로 하나의 XillyUSB 장치가 host에 연결된 경우 두 개의 0이 있습니다.

Linux 시스템에서 stream은 /dev/xillybus_mystream와 같은 일반 파일로 열립니다. Windows에서는 동일한 stream이 \\.\xillybus_mystream로 나타납니다.

device file은 반대 방향으로 두 개의 streams를 나타낼 수 있습니다. 이는 device file의 이름을 공유하는 두 개의 streams 뿐입니다. 이 두 개의 streams는 어느 방향으로든 별도로 열거나 읽기-쓰기를 위해 열 수 있습니다. 이 기능은 일반적으로 혼동을 방지하기 위해 피해야 하지만 device file이 양방향 pipe을 예상하는 소프트웨어에 전달될 때 유용합니다.

2.3 데이터 너비

데이터 너비는 FPGA의 FIFOs에서 가져오거나 FIFOs에 쓰는 워드의 비트 수입니다. 허용되는 선택은 32, 16 또는 8비트입니다. XillyUSB뿐만 아니라 개정 B/XL/XXL(섹션 4에서 설명)의 Xillybus IP cores에서는 더 넓은 데이터 너비가 허용됩니다.

stream에서 고대역폭 성능이 필요하고 IP core의 개정판이 PCIe용 A 또는 AXI용 IP core인 경우 데이터 너비를 32비트로 설정해야 합니다. 16비트 및 8비트 데이터 너비의 경우 상당한 성능 저하가 발생하므로 기본 전송(예: PCIe bus 전송)의 비효율적인 사용.

그 이유는 단어가 bus clock의 속도로 Xillybus의 내부 데이터 경로를 통해 전송되기 때문입니다. 결과적으로 8비트 워드를 전송하는 데 32비트 워드와 동일한 시간 슬롯이 걸리므로 효과적으로 4배 느려집니다.

이는 데이터 경로가 더 느린 데이터 요소로 채워지기 때문에 주어진 시간에 기본 전송을 위해 경쟁하는 다른 streams에도 영향을 미칩니다.

IP core 및 XillyUSB의 최신 개정판은 내부 데이터 경로 구조가 다르므로 이러한 제한이 없

습니다.

그럼에도 불구하고 데이터 너비와 일치하는 세분성을 사용하여 host application에서 I/O 작업을 수행하는 것이 좋습니다. 예를 들어 데이터 너비가 32비트인 경우 데이터 길이가 4의 배수인 read() 및 write() 함수를 호출합니다.

데이터 너비를 잘못 선택하면 원하지 않는 동작이 발생할 수 있습니다. 예를 들어, host에서 FPGA로의 링크 폭이 32비트인 경우 host에서 3바이트의 데이터를 쓰면 driver가 FPGA로 아무 것도 보내기 전에 네 번째 바이트를 무기한 대기하게 됩니다.

2.4 Use

“Use” 속성은 IP core를 생산하는 도구가 각 stream에 의도한 응용 프로그램에 가장 적합한 속성을 부여하는 데 도움이 됩니다.

최상의 성능을 얻으려면 stream의 목적에 가장 적합한 옵션으로 “Use”를 설정하는 것이 중요합니다.

각 옵션에 대한 간략한 설명은 다음과 같습니다.

- **Frame grabbing / video playback:** stream이 비디오 데이터 애플리케이션용인 경우 이를 선택하세요.
- **Data acquisition / playback:** stream을 지속적으로 데이터를 생성하거나 사용하는 DAC/ADC 또는 다른 장치에 연결하려는 경우 이 옵션을 선택하세요.
- **Data exchange with coprocessor:** stream이 hardware acceleration에 사용되는 경우(즉, 성능 향상을 목적으로 CPU 대신 FPGA를 사용하여 작업을 수행하는 경우) 이 옵션을 선택합니다. 이 선택은 stream에 높은 데이터 속도가 필요하지만 데이터 흐름이 가끔 잠깐 멈출 수 있는 경우에 적합합니다.
- **Bridge to external hardware:** 이 옵션은 FPGA가 stream의 도움으로 외부 하드웨어를 제어할 때 적합합니다. 예를 들어, stream의 데이터에 다른 구성 요소의 펌웨어가 포함된 경우입니다.
- **Data for in-silicon logic verification:** logic의 적절한 기능을 검증하기 위해 stream을 사용하여 logic로 애플리케이션 데이터를 전송하거나 logic에서 애플리케이션 데이터를 전송하는 경우 이 옵션을 선택합니다.
- **Command and status:** stream이 FPGA에 명령을 보내거나 FPGA의 상태에 대한 정보를 수집하도록 의도된 경우 이 옵션을 선택하세요.
- **Short message transport:** 이 옵션은 stream에 짧은 정보 세그먼트가 포함되어 있으며, 메시지를 보내는 목적일 때 적합합니다.

- **Address / data interface:** stream와 함께 lseek()을 사용할 수 있도록 하려면 이것을 선택합니다. 이 옵션을 선택하면 FPGA 측의 인터페이스에 주소 output이 추가됩니다.
이 옵션은 세 가지 형태로 제공되며, 각 형태는 5, 16 또는 32개의 다른 수의 주소선을 제공합니다.
seekable streams에 대한 주제는 [Xillybus FPGA designer's guide](#)에서 더 자세히 설명합니다.
- **General purpose:** 위의 어떤 것도 귀하의 애플리케이션에 맞지 않는 경우 이 옵션을 선택해야 합니다.

“Autoset internals”를 사용하면 도구는 위의 옵션 중 어느 것을 선택했는지에 따라 stream이 동기식인지 비동기식인지 판별합니다. stream은 동기식이며 다음 옵션 중 하나를 선택하면 Command and status, Short message transport, Address / data interface 또는 Bridge to external hardware가 됩니다. stream은 다른 모든 옵션에 대해 비동기식입니다.

2.5 동기 또는 비동기 stream

이 속성은 “use” 설정 선택에 따라 “Autoset internals” 옵션이 선택될 때 자동으로 설정됩니다.

대부분의 경우, 비동기 streams는 연속적인 데이터 흐름에 적합하고, 동기식 streams는 명령, 제어 데이터 및 상태 정보 수집에 적합합니다.

동기식 streams의 경우 모든 I/O(FPGA의 데이터 흐름 포함)는 read() 또는 write()에 대한 함수 호출의 호출과 반환 사이에만 발생합니다. 이것은 어떤 일이 발생하는지에 대한 완전한 제어를 제공하지만 CPU가 다른 작업을 수행하는 동안 데이터 전송 리소스를 사용하지 않은 상태로 둡니다. 다음 두 문서 중 하나의 섹션 2에서 이 주제에 대한 자세한 설명을 읽는 것이 좋습니다.

- [Xillybus host application programming guide for Linux](#)
- [Xillybus host application programming guide for Windows](#)

동기식 streams로 작업하면 소프트웨어 프로그래밍이 더 직관적이지만 대역폭 활용에 부정적인 영향을 미칩니다. 비동기식 streams를 사용하면 운영 체제가 특정 기간 동안 user space에서 실행되는 프로세스에서 CPU를 제거하더라도 지속적인 데이터 흐름을 유지할 수 있습니다.

이 주제를 요약하면 다음과 같은 안내 질문이 있습니다.

- downstreams의 경우(host FPGA): 데이터가 FPGA에 도달하기 전에 write() 작업이 반환되어도 괜찮습니까?
- upstreams의 경우(FPGA host): host의 read() 작업이 요청하기 전에 Xillybus IP core가 FPGA의 user application logic에서 데이터 가져오기를 시작해도 괜찮습니까?

해당 질문에 대한 대답이 아니오인 경우 동기식 stream이 필요합니다. 그렇지 않으면 데이터 흐름에 대한 제어가 덜하고 직관적이지 않다는 이해와 함께 비동기식 옵션이 일반적으로 선호되는 선택입니다.

2.6 버퍼링 시간

Xillybus는 FPGA와 host 사이에 데이터의 연속적인 stream의 환상을 유지합니다. DMA buffers의 존재는 FPGA의 user application logic와 host의 애플리케이션 소프트웨어에 투명합니다. 특히 높은 데이터 속도에서 데이터 흐름의 효율성과 연속성을 유지하는 기능을 제어하는 데만 관심이 있습니다.

data acquisition 및 data playback와 같은 응용 프로그램은 FPGA에서 데이터의 지속적인 흐름이 필요합니다. 그렇지 않으면 데이터가 손실됩니다. 이 흐름을 유지하기 위해 user space application은 FPGA의 활동으로 인해 DMA buffers가 (각각) 가득 차거나 비어 있는 것을 방지할 수 있을 만큼 충분히 자주 read() 또는 write()에 대한 함수 호출을 수행해야 합니다.

그러나 이러한 함수 호출이 충분히 자주 이루어지도록 하는 데 문제가 있습니다. Linux 및 Windows와 같은 일반 운영 체제는 이론적으로 임의의 시간 동안 user-space application에서 CPU를 박탈할 수 있습니다. FPGA는 관계없이 driver의 buffers를 계속 채우거나 비웁니다. 따라서 DMA buffers는 CPU의 일시적인 부족에도 불구하고 지속적인 데이터 흐름을 유지할 수 있을 만큼 충분히 커야 합니다.

여기에서 논의를 위해 *버퍼링 시간*은 stream이 모든 DMA buffers가 비어 있는 상태에서 모든 DMA buffers가 가득 찬 상태로 데이터가 buffers를 다음 비율로 채울 때 stream이 변경되는 데 걸리는 시간입니다. stream이 의도된 것입니다(그리고 그 시간 동안 배수되지 않습니다).

“Autoset internals”가 활성화된(권장) Xillybus stream을 설정할 때 “Buffering”라는 제목의 선택 상자가 웹 응용 프로그램에 나타납니다. 여기서 원하는 버퍼링 시간이 선택됩니다.

연속성을 유지해야 하는 비동기식 stream의 경우 선택한 시간은 CPU가 user space application에서 제거될 수 있는 예상 최대 시간을 반영해야 합니다.

“Maximum”을 선택하면 buffers를 할당하는 알고리즘이 다른 streams에 대해 약간만 고려하면서 가능한 한 많은 RAM을 할당하도록 시도합니다.

원하는 버퍼링 시간 t 와 예상 대역폭 W 가 주어지면 알고리즘은 다음 공식을 기반으로 driver의 DMA buffers에 대해 RAM, M 의 총량을 할당하려고 시도합니다.

$$M = t \times W$$

그러나 실제 buffer 크기는 항상 2의 거듭제곱(2^N)입니다. 또한 원하는 버퍼링 시간을 충족하기에 충분한 메모리를 할당하는 것이 불가능할 수도 있습니다.

따라서 IP core의 README 파일에서 할당된 buffer 크기를 조회하고 작업하기에 적합한지 확인하는 것이 중요합니다. buffer 크기를 수동으로 설정(즉, "Autoset internals" 끄기)하면 streams 간에 RAM이 더 잘 분포되도록 할 수 있으며, 이는 의도한 애플리케이션에 더 적합합니다.

2.7 DMA buffers의 크기

웹 애플리케이션에서 "Autoset internals"를 활성화하여 도구가 DMA buffers의 매개변수를 자동으로 설정하도록 하는 것이 좋습니다(위의 2.6 섹션 참조). 일부 시나리오에서는 자동 설정이 애플리케이션에 적합하지 않을 수 있으며, 이 경우 DMA buffers의 크기와 번호를 수동으로 설정할 수 있습니다.

비동기식 streams의 경우 buffers의 매개변수는 다음 두 문서의 "Continuous I/O at high rate" 섹션에서 설명하는 상당한 영향을 미칩니다.

- [Xillybus host application programming guide for Linux](#)
- [Xillybus host application programming guide for Windows](#)

DMA buffers의 크기를 `read()` 및 `write()`의 의도된 함수 호출 크기에 맞게 조정할 필요가 없습니다. 이 두 가이드에서 설명했듯이 DMA buffers의 크기는 `read()` 및 `write()`에 대한 함수 호출과 관련이 없으며 투명합니다. 특히 `read()`에 대한 함수 호출은 충분한 데이터가 FPGA의 IP core에 도달하면 즉시 반환됩니다(DMA buffer의 채우기 수준에 관계없이). 이는 FPGA가 부분적으로 채워진 DMA buffer를 제출할 수 있도록 하는 FPGA와 host 간의 메커니즘 덕분입니다. 이 메커니즘은 `read()`에 대한 함수 호출을 즉시 완료하는 데 도움이 될 때 사용됩니다.

마찬가지로 host에서 FPGA로의 데이터는 명시적 요청 덕분에 FPGA에 즉시 도달하도록 보장할 수 있습니다.

DMA buffers의 크기와 의도한 데이터 교환 패턴을 연결하는 것은 일반적인 실수입니다. Xillybus를 사용하면 그럴 필요가 없으므로 "Autoset internals"가 DMA buffers의 크기를 설정하는 데 선호되는 선택입니다.

연속성을 위해 RAM이 많을수록 더 좋습니다. DMA buffers의 총 공간은 CPU가 애플리케이션에서 박탈된 경우에도 데이터 흐름을 연속적으로 유지하기 때문입니다. 올바른 결정을 내리기 위해서는 위에서 언급한 프로그래밍 가이드에 자세히 설명되어 있는 다른 요소가 필요합니다.

그러나 DMA buffers의 **전체 크기**가 지나치게 크면 대용량 데이터를 저장할 수 있는 기능의 결과인 buffering delay의 위험이 있습니다. 결과적으로 한쪽이 buffers를 채우는 속도가 다른 쪽이 비우는 속도보다 빠르면 상당한 시간이 지난 후 데이터가 다른 쪽 끝에 도착할 수 있습니다. 이것은 “Monitoring the amount of buffered data”라는 섹션에서 [Xillybus FPGA designer's guide](#)에 언급된 기술로 제어할 수 있습니다.

XillyUSB IP cores의 경우 각 stream에 대해 하나의 buffer가 있으며 이는 driver에서 관리하는 대형 FIFO로 작동합니다. 다른 IP cores(PCIe 및 AXI)는 각 stream에 대해 여러 DMA buffers를 유지하므로 크기와 개수가 모두 정의됩니다. 따라서 DMA buffers의 유효 크기는 각 DMA buffer의 크기에 숫자를 곱한 것입니다.

따라서 PCIe / AXI를 기반으로 하는 IP cores에 대해 “Autoset internals”가 꺼져 있는 경우에는 DMA buffers의 개수와 각각의 크기를 지정해야 합니다. 다음 사항을 고려해야 합니다.

- 각 DMA buffer의 크기는 host에서 FPGA까지 streams에서 자체적으로 의미가 있습니다. 이러한 buffers가 가득 차면 데이터가 FPGA로 전송됩니다(flush가 소프트웨어에서 명시적으로 요청되거나 stream이 10밀리초 동안 유휴 상태인 경우 제외).). 따라서 각 DMA buffer의 크기는 흐르는 데이터의 일반적인 latency에 영향을 미칩니다.
- 느린 streams(10 MBytes/s 미만)의 경우 권장되는 DMA buffers 수는 4입니다. 더 높은 대역폭이 필요한 경우 buffers의 수는 적절한 전체 DMA buffer 할당을 달성하기 위해 선택됩니다. 고대역폭 streams의 경우 적절한 DMA buffers 수는 16 64 사이이며, 각 buffer가 128 kBytes 이하가 될 수 있는 경우입니다.
- 항상된 driver가 host에서 사용되지 않는 한 모든 streams에 대한 DMA buffers의 총 할당량은 512 MBytes를 초과해서는 안 됩니다. 그렇지 않으면 운영 체제가 이 이상 할당을 거부하여 driver의 초기화에 실패할 수 있습니다.
- buffer가 채워질 때마다 hardware interrupt가 host로 전송됩니다. 예상 데이터 속도가 주어지면 interrupts의 속도를 계산하고 processor에 정상적인 수준으로 유지해야 합니다(초당 수천 개 이하).
- 각 DMA Buffers의 크기는 숫자를 늘려 전체 크기에 도달할 수 있는 경우 128 kBytes를 초과하지 않아야 합니다.

driver의 DMA buffers 문제는 동기식 streams의 경우 덜 중요합니다. 이를 위해, stream을 대신하여 buffers에 할당된 총 RAM은 read() 및 write()의 의도된 함수 호출의 데이터 길이 크기 정도여야 합니다. 위에서 이미 말했듯이 buffers의 크기를 이러한 함수 호출에 맞게 조정할 필요는 없지만 kernel RAM을 그렇게 크게 만들어서 낭비하는 경우는 거의 없습니다.

2.8 DMA acceleration

PCIe를 기반으로 하는 IP cores의 경우 host에서 FPGA로의 streams는 DMA 데이터 전송의 가속화가 필요할 수 있습니다.

이 방향으로 데이터 교환을 수행하기 위해 PCIe bus protocol은 FPGA가 host에서 데이터 요청을 발행하고 데이터가 도착할 때까지 기다려야 한다고 명시합니다. 요청이 bus에서 이동하고 host가 대기열에 넣어 처리하고 데이터가 다시 이동하면 고유한 지연이 발생합니다. 이러한 처리 시간 간격으로 인해 bus의 효율성이 약간 저하되고 때로는 단일 stream의 대역폭이 40%만큼 낮아집니다.

이 문제를 해결하기 위해 여러 데이터 요청이 전송되어 host는 연속 전송 중에 항상 대기열에 요청을 갖습니다. 다른 요청의 데이터가 임의의 순서로 도착할 수 있으므로 application logic에 데이터의 순서화된 흐름을 제공하려면 FPGA의 RAM buffers에 데이터를 저장해야 합니다.

FPGA의 각 buffer는 요청된 데이터 세그먼트를 저장하는 데 사용됩니다. DMA accelerations에 대한 현재 가능한 설정은 다음과 같습니다.

- 없음. FPGA에는 데이터가 저장되지 않습니다. 각 데이터 요청은 이전 요청에서 모든 데이터가 도착한 경우에만 전송됩니다.
- 각각 512바이트의 4개 세그먼트. block RAM의 2048바이트가 FPGA에 할당됩니다. 주어진 순간에 최대 4개의 데이터 요청이 활성화될 수 있습니다.
- 각각 512바이트의 8개 세그먼트. block RAM의 4096바이트가 FPGA에 할당됩니다. 주어진 순간에 최대 8개의 데이터 요청이 활성화될 수 있습니다.
- 개정 B 및 이후 IP cores에는 각각 512바이트의 16개 세그먼트 옵션도 있습니다.

요청과 데이터 도착 사이의 처리 시간은 host의 하드웨어에 따라 다릅니다. 따라서 실제 대역폭 성능은 다를 수 있습니다.

IP Core Factory에서 "Autoset internals"를 사용할 때 가속 리소스의 자동 할당은 일반적인 PC 컴퓨터 하드웨어에서 측정된 결과를 기반으로 하며 드문 경우지만 수동 조정이 필요할 수 있습니다.

3

확장성 및 logic 리소스 소비

3.1 일반적인

Xillybus는 확장성을 염두에 두고 설계되었습니다. 단일 stream에 대해 맞춤형 IP core를 구성하는 것이 완벽하지만, 많은 수의 streams로 확장하는 것은 Xillybus core가 소비하는 logic의 양에 상대적으로 적은 영향을 미칩니다.

logic의 소비량을 측정하기 위해 Xillybus IP core(기준선, 개정판 A)의 연속 빌드가 streams의 수를 늘리면서 만들어졌습니다. 모든 테스트에서 FPGA에서 host까지 streams의 수는 반대 방향과 동일했습니다. streams의 수는 2개(각 방향에 1개)에서 64개(각 방향에 32개) 범위였습니다.

이 섹션에서는 도구에서 보고한 대로 FPGAs의 세 가지 제품군에서 IP core 자체의 logic 소비에 대해 간략히 설명합니다. 이러한 FPGAs(AMD 기준)는 상당히 구식이지만, AMD 및 Altera가 유사하게 최신 FPGAs에서 유사한 결과를 얻었습니다.

수정 버전 B, XL 및 XXL이 있는 IP cores에 대한 유사한 분석은 4 섹션을 참조하십시오.

XillyUSB는 이러한 분석에서 다루지 않습니다.

3.2 Block RAMs

Xillybus core에서 사용하는 block RAMs의 수는 0에서 몇 개까지 다양합니다(64 streams의 경우 3 block RAMs). 각 stream의 Xillybus core 내부에는 buffers가 없습니다. 오히려 Xillybus core는 연결된 FIFOs에 의존하여 데이터를 수집합니다. 내부적으로 core에는 모든 streams에서 사용하는 단일 메모리 풀이 있습니다.

streams의 수가 증가함에 따라 block RAMs는 DMA buffers의 주소를 저장하는 데 사용됩니다.

추가 block RAMs는 core의 README 파일에 설명된 대로 host에서 FPGA까지 streams용

DMA acceleration에 할당됩니다.

3.3 logic fabric의 리소스

아래 그래프는 streams의 수가 2에서 64로 증가함에 따라 LUTs 및 registers(flip-flops)의 소비를 보여줍니다. 해당 그래프의 각 점은 synthesis 보고서에 표시된 대로 사실상 사용입니다. 이 그래프에서 분명한 것은 logic 소비가 거의 선형으로 증가한다는 것입니다. FPGA 아키텍처에 관계없이 각 stream은 평균적으로 약 110개의 LUTs 및 82개의 registers를 추가합니다.

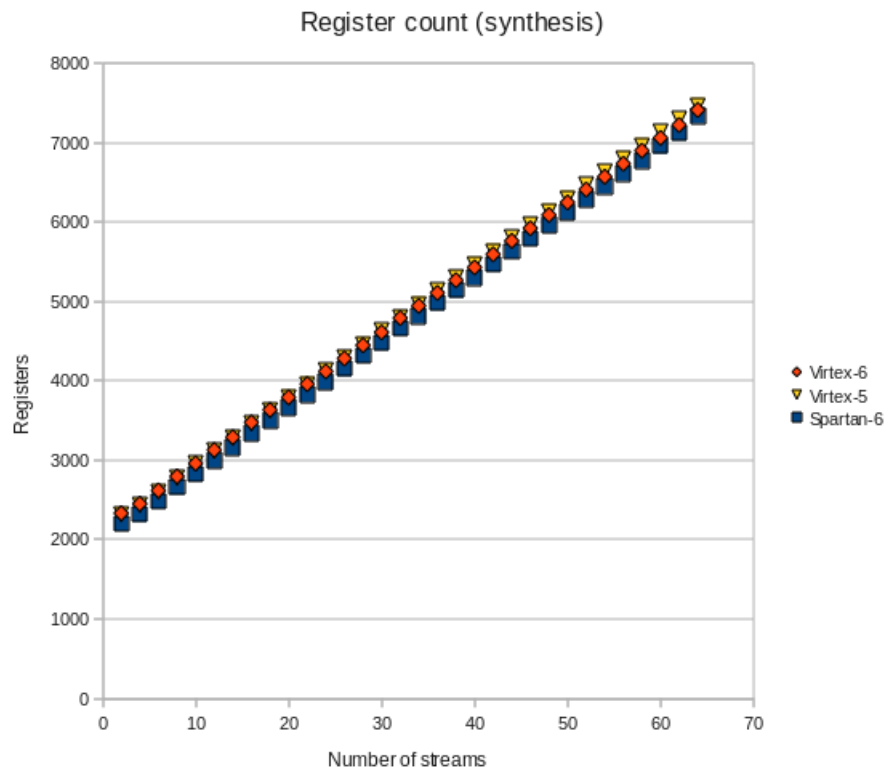
FPGA에서 실제로 소비되는 slices의 수는 logic의 요소가 어떻게 구성되어 있는지에 따라 다릅니다. Spartan-6 또는 Virtex-6 제품군에서 각 slice는 최대 8개의 LUTs 및 8개의 registers를 포함할 수 있습니다. 따라서 매우 낙관적인 접근 방식은 registers가 완벽하게 포장되어 각 stream이 소비된 리소스에 $110/8 = 14$ slices만 추가한다고 가정하는 것입니다. 반면에 그 효율성의 절반으로 포장하는 것은 상당한 노력 없이 달성할 수 있는 것입니다. 따라서 stream에 대한 slices의 예상 비용은 14-28 slices 범위에서 추정할 수 있습니다.

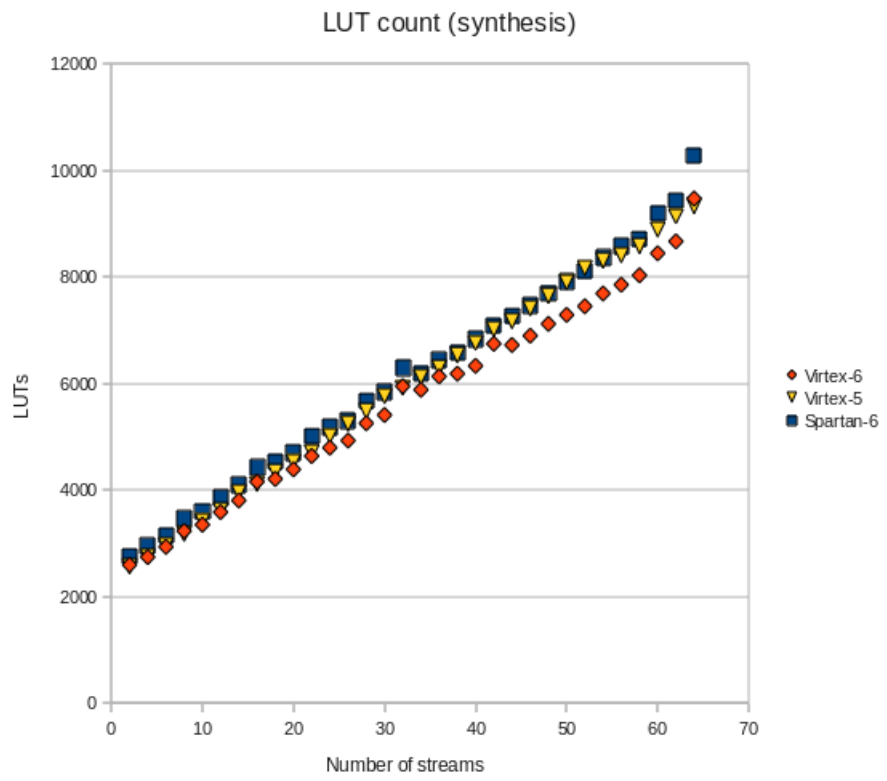
FPGA가 거의 꽉 찼을 때 implementation을 수행하는 도구는 logic을 slices에 효율적으로 포장하지 않으므로 slices 수의 증가가 훨씬 더 가파를 수 있다는 점에 유의하는 것이 중요합니다. 이 경우 도구는 자원이 많기 때문에 자원을 낭비합니다.

벤치마크 테스트를 위해 선택한 설정은 upstreams의 경우 50%이고 downstreams의 경우에도 동일합니다. 실제 IP cores는 일반적으로 방향 중 하나에 중점을 두지만 아래 결과는 무엇을 기대해야 하는지에 대한 아이디어를 제공합니다.

그래프는 다음과 같습니다. 기울기가 가파르게 보일 수 있지만 streams의 수는 최소 IP core(2 streams)에서 다소 무거운 것(64 streams)으로 이동합니다.

결론은 slices 수에 대한 기여도가 상당히 낮기 때문에 가장 사소한 작업에도 IP core에 추가 streams를 할당하는 것이 합리적이라는 것입니다.





4

개정판 B, XL 및 XXL의 IP cores

4.1 일반적인

지금까지 이 문서는 2010년부터 사용 가능한 IP cores의 기준 개정판(개정판 A)과 관련되어 있습니다. 개정판 B 및 XL은 Xillybus 사용자의 데이터 대역폭 요구 사항에 맞게 2015년에 도입되었습니다. 이 cores는 점진적으로 개정 A를 대체합니다.

개정판 XXL은 2019년에 도입되었습니다.

새 개정판(B, XL 및 XXL)은 개정판 A와 비교하여 기능의 상위 집합을 제공하지만 동일한 속성으로 정의할 때 기능적으로 동일합니다(일부 가능한 성능 향상 포함).

가장 눈에 띄는 차이점은 다음과 같습니다.

- 데이터 대역폭 증가: 모든 FPGA의 경우 IP cores 개정판 B, XL 및 XXL은 각각 개정판 A 대역폭의 약 2배, 4배 및 8배의 총 대역폭을 허용합니다.
Xillybus IP cores의 대역폭 기능을 얻는 방법에 대해서는 [Getting started with Xillybus on a Linux host](#) 또는 [Getting started with Xillybus on a Windows host](#)의 섹션 5를 참조하십시오.
- 64, 128 및 256비트의 사용자 인터페이스 데이터 너비는 이미 존재하는 8, 16 및 32비트 옵션에 추가로 허용됩니다. 이러한 너비는 사용 중인 Xillybus의 IP core와 PCIe 블록 사이의 데이터 경로 너비에 관계없이 허용됩니다.
- logic design은 더 빠르며(timing constraints에 도달하기 더 쉬움) 가장 느린 timing path에서 지연이 약 1 ns 더 적습니다.
- logic의 소비는 가장 일반적인 경우에 더 낮습니다(섹션 4.4 참조).
- PCIe 블록의 대역폭은 IP core와 application logic 사이의 신호 데이터 폭에 관계없이 효율적으로 활용됩니다. 이것은 개정판 A에서 8비트 및 16비트 워드가 있는

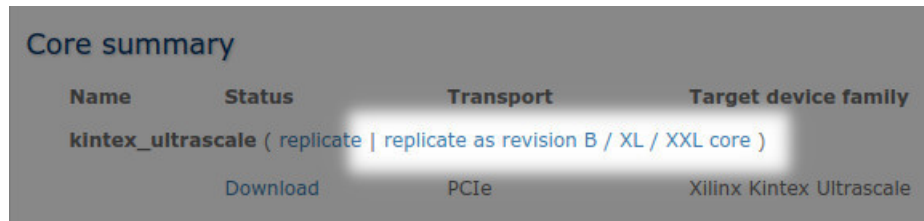
streams의 효율성이 낮은 것과 반대입니다.

- AMD 플랫폼에서 개정판 B, XL 및 XXL은 Vivado에서만 사용할 수 있습니다.

4.2 개정 B/XL/XXL 작업

수정 버전 B/XL/XXL의 IP core는 수정 버전 A의 IP core를 복제하여 IP Core Factory에서 생성됩니다.

이것은 IP Core Factory의 이 스크린샷에서와 같이 “replicate as revision B / XL / XXL core”를 클릭하여 수행됩니다.



버전 A로 다시 다운그레이드할 수 없습니다.

B/XL/XXL로의 업그레이드 가능성은 이러한 고급 IP cores에 대한 액세스를 요청한 사용자만 사용할 수 있습니다. 이러한 요청은 웹사이트에 광고된 연락처 정보를 사용하여 일반 이메일로 이루어집니다.

이 액세스 권한을 얻기 위한 특별한 요구 사항은 없습니다. 이 요청의 목적은 고급 사용자와 더 긴밀하게 접촉하기 위한 것입니다.

개정판 B의 IP cores는 개정판 A용 drop-in replacements입니다. 따라서 원하는 FPGA의 기준선 demo bundle을 시작점으로 사용해야 합니다. 이 demo bundle은 개정판 A의 IP core와 함께 제공되므로 개정판 B로 작업하려는 사람들은 IP Core Factory에서 구성하고 다운로드해야 합니다.

반면에 개정판 XL 및 XXL을 사용하려면 전용 demo bundle이 필요합니다. 이 demo bundles는 이메일을 통해 요청해야 합니다.

4.3 데이터 워드의 너비

개정판 A의 IP cores는 8, 16 및 32비트의 애플리케이션 데이터 너비만 허용하지만 개정판 B/XL/XXL은 64, 128 및 256비트 폭 인터페이스도 허용합니다. 주요 동기는 단일 stream로 전체 대역폭 용량을 활용할 수 있도록 하는 것입니다.

그럼에도 불구하고 여러 streams(8, 16 또는 32비트 폭)를 사용하여 대역폭을 분할하여 전체 대역폭이 전체 대역폭 기능을 활용하도록 할 수도 있습니다(5-10% 저하 가능성 있음).

데이터 너비는 application logic에서 자연스럽게 작동하도록 선택해야 합니다.

대역폭 기능을 높이는 데 도움이 되는지 여부에 관계없이 더 넓은 워드 인터페이스가 허용됩니다. 예를 들어, 64비트가 IP core의 대역폭을 활용하기에 충분하다면 개정 B의 IP cores에서는 256비트의 워드 너비가 허용됩니다. 이러한 데이터 폭은 PCIe 블록과의 인터페이스 신호와 관련이 없습니다.

32비트 이상의 워드 너비를 사용할 때 PCIe bus의 기본 데이터 요소는 32비트이므로 streams의 잘못된 사용을 방지하는 driver의 일부 안전 장치는 데이터 너비가 32비트 이상일 때 적용되지 않습니다. 32비트. 예를 들어, 워드 너비가 64비트인 stream에 대해 read() 또는 write()에 대한 함수 호출은 길이가 8의 배수여야 합니다. 마찬가지로 64비트 너비의 stream에서 seek 작업이 요청한 위치는 의미 있는 결과를 얻기 위해 8의 배수여야 합니다. 그러나 소프트웨어는 4의 배수만 적용합니다.

결론적으로, 데이터 폭이 32비트 이상일 때, 워드의 폭에 맞춰 정렬된 I/O를 수행하는 것은 응용 소프트웨어가 더 책임이 있습니다. 워드 정렬 규칙은 모든 워드 너비에 대해 동일하지만 32비트 및 16비트 워드 너비의 streams와 달리 driver는 이러한 규칙을 반드시 적용하지는 않습니다.

4.4 Logic 리소스 소비

수정 버전 B/XL/XXL의 IP cores는 streams의 수가 증가함에 따라 logic의 소비가 약간 더 가파르게 오르는 대신 logic의 속도와 약간 낮은 소비에 최적화되어 있습니다.

logic 자원의 사용을 수량화하기 위해 streams의 수가 증가하는 Kintex-7용 cores가 생성되었습니다. cores는 synthesis를 겪었고 logic의 요소는 계산되었습니다. 3.3 섹션에서와 같이 벤치마크 테스트는 upstreams의 경우 50%이고 downstreams의 경우에도 동일합니다.

다음 세 개의 차트는 logic의 소비를 보여주고 A, B 및 XL 개정판의 IP cores를 동일한 설정으로 비교합니다. 테스트된 모든 streams는 32비트 너비였습니다.

registers와 LUTs의 수를 비교하면 B가 streams의 수가 적으면 A보다 성능이 우수하지만 streams의 수가 증가하면 이러한 이점이 사라집니다.

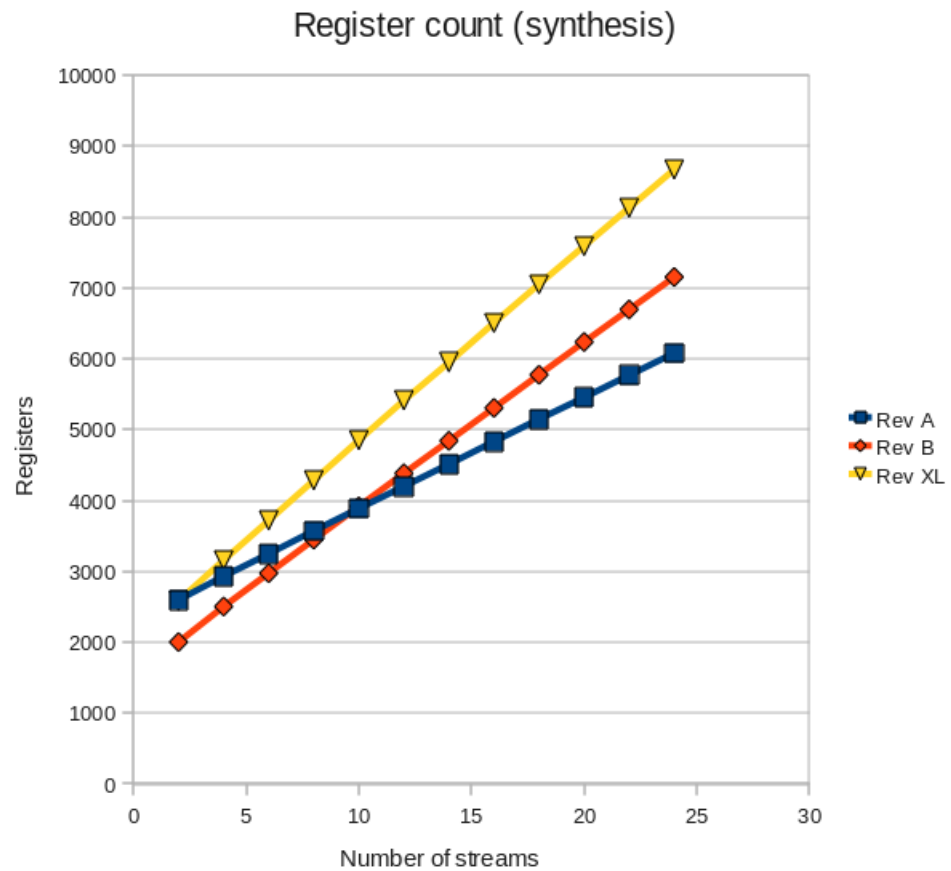
개정판 XL 및 XXL은 모든 시나리오에서 다른 두 개정판보다 더 많은 logic을 사용합니다.

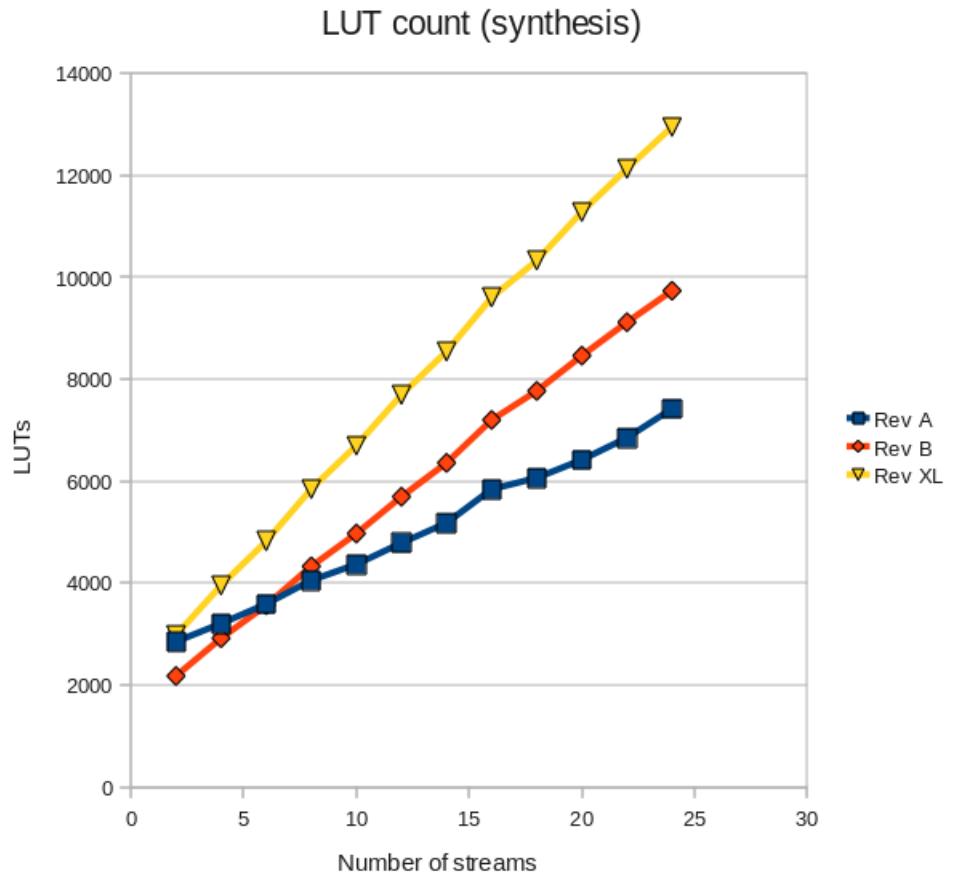
block RAMs에 대한 차트는 개정 B와 XL이 개정 A에 비해 두 배의 block RAMs를 소비한다는 것을 보여줍니다.

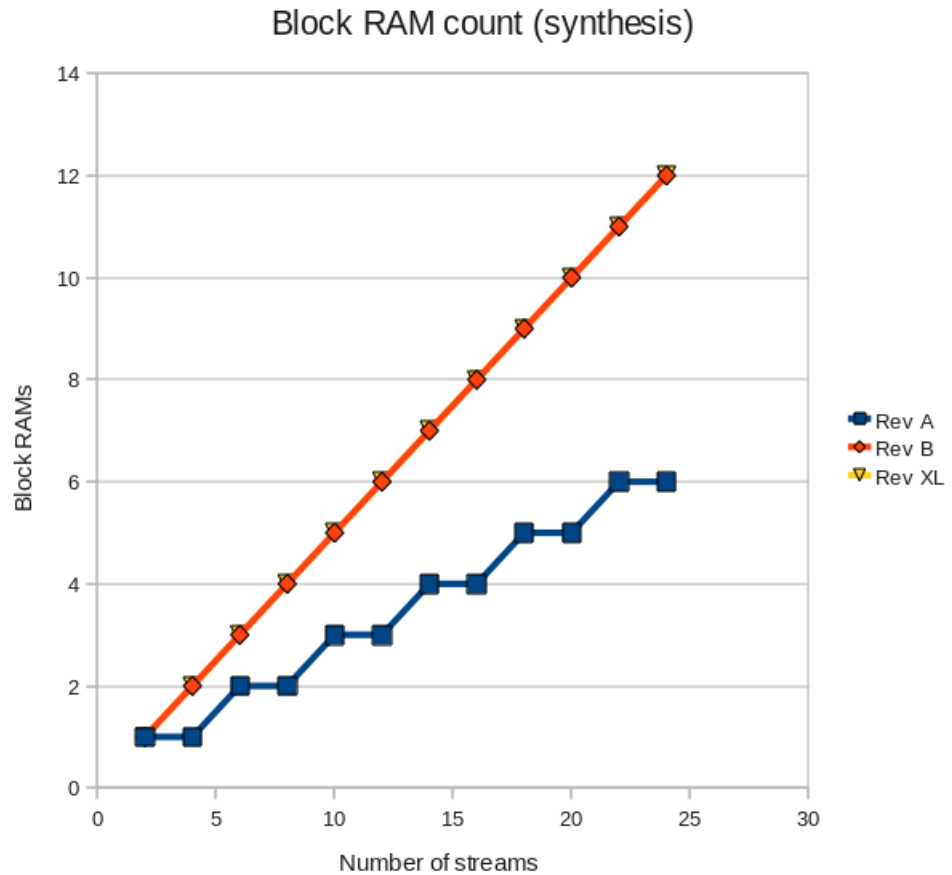
제안된 결론은 개정판 B가 거의 항상 개정판 A보다 선호되어야 한다는 것입니다. logic의 소비가 반대라고 말하더라도 개정판 B의 개선된 timing은 이 차이를 능가합니다. 이것은

IP core의 logic 소비가 FPGA의 용량에 비해 무시할 수 있는 대부분의 실제 시나리오에서 사실입니다.

반면 XL 및 XXL 버전은 logic을 더 많이 소비하고 timing constraints와 관련하여 더 어렵기 때문에 대역폭 용량이 필요한 애플리케이션에만 선택해야 합니다.







4.5 host에서 FPGA로 stream의 최적 대역폭을 위한 조정

수정 버전 B, XL 및 XXL이 있는 IP cores는 더 높은 데이터 속도를 허용하므로 PCIe 블록 매개변수의 적절한 조정에 점점 더 민감해집니다.

demo bundles의 PCIe 블록은 이미 최적의 성능을 위해 설정되어 있습니다. 그러나 PCIe bus(host 하드웨어의 일부)가 정상보다 긴 latency로 패킷을 릴레이하는 경우 약간의 조정이 필요할 수 있습니다.

FPGAs by AMD 중 이것은 Gen2로 제한되는 PCIe 블록이 있는 Kintex-7 또는 Virtex-7와 함께 사용되는 개정 B 또는 XL의 IP cores에만 적용됩니다. 개정판 A는 개선 사항이 확인되는 데이터 속도에 도달하지 않습니다.

이 제한된 경우에 host에서 FPGA로 streams에서 의도한 대역폭을 달성하기 위해 PCIe 블록의 매개변수를 조정해야 할 수 있습니다.

이는 FPGA에서 발행한 DMA 전송 요청으로 인해 데이터가 host에서 FPGA 방향으로 흐르기 때문에 필요할 수 있습니다. host는 데이터를 전송하여 이러한 요청을 수행합니다. 요청과 이를 수행하는 데이터 전송 사이의 지연(completions)은 이러한 요청에 대한 host의 응답성에 따라 달라지며 PCIe bus마다 다릅니다.

PCIe bus가 제공하는 대역폭을 효과적으로 사용하기 위해 여러 DMA 요청이 FPGA에 의해 병렬로 발행되어 host가 항상 처리할 요청을 갖도록 합니다. 그러나 PCIe protocol의 흐름 제어에 의해 부과되는 활성 요청 수에는 제한이 있습니다. 흐름 제어 메커니즘에 의해 할당되는 제한된 리소스를 *completion credits*라고 하며 PCIe endpoint에 대해 구성됩니다. 일반적으로 이러한 항목이 많을수록 더 많은 활성 요청이 허용되며 PCIe 블록을 구현하기 위해 FPGA에서 더 많은 리소스가 필요합니다.

FPGA가 DMA 요청과 함께 데이터를 보내기 때문에 FPGA에서 host 방향은 credits 할당에 의해 영향을 거의 받지 않습니다. 따라서 영향이 거의 없기 때문에 credits의 설정을 수정하여 개선할 가능성이 적습니다.

demo bundles의 PCIe 블록은 x86 아키텍처를 기반으로 하는 processor와 함께 일반 데스크탑 컴퓨터에서 최적의 대역폭 활용을 위해 구성됩니다. 아주 드물긴 하지만 host에서 FPGA 방향으로 알려진 대역폭을 얻기 위해 구성을 변경해야 할 수도 있습니다.

completion credits(header 및 data 모두)의 수를 늘리면 개선할 수 있습니다. 이는 PCIe block IP의 구성을 호출하고 해당 매개변수를 수정하여 Vivado 또는 Quartus에서 수행됩니다. 예를 들어 Vivado에 구성된 Kintex-7의 경우 “Core Capabilities” 탭을 선택하고 “BRAM Configuration Options”를 설정하면 됩니다. “Perf level”은 이미 가능한 최고로 설정되어 있으므로 개선의 여지가 없습니다. 그러나 “Buffering Optimized for Bus Mastering Application”을 활성화하면 다른 유형의 credits를 희생하여 completion credits가 증가합니다. 이것은 반대 방향에 대한 역효과 없이 host에서 FPGA 방향으로의 대역폭 성능을 향상시킬 수 있습니다.