(기계로 한국어 번역)

Getting started with Xillybus on a Linux 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 This document may also be slightly outdated in relation to the original.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 **4**

2 host driver 설치 **6**

2.1 Xillybus의 driver 설치 단계 6
2.2 정말 설치해야 할 것이 있습니까? 6
2.2.1 일반적인 6
2.2.2 driver가 사전 설치된 Linux distributions 7
2.2.3 Xillybus의 driver를 포함하는 Linux kernels 7
2.3 전제 조건 확인 8
2.4 다운로드한 파일 압축 해제 9
2.5 kernel module의 compilation 실행 9
2.6 kernel module 설치 10
2.7 udev rule 파일 복사 11
2.8 모듈 로드 및 언로드 12
2.9 공식 Linux kernel의 Xillybus drivers 12

3 “Hello, world” 테스트 **14**

3.1 목표 14
3.2 준비 15
3.3 사소한 loopback 테스트 15

4 host 애플리케이션의 예 **18**

4.1 일반적인 18
4.2 편집 및 compilation 19
4.3 프로그램 실행 21
4.4 메모리 인터페이스 21

5 고대역폭 성능을 위한 지침 **24**

5.1 loopback 하지마 24
5.2 디스크 또는 기타 저장소를 포함하지 마십시오. 25
5.3 많은 부분 읽기 및 쓰기 26

5.4 CPU 소비에 주의 27
5.5 읽기와 쓰기를 상호 의존적으로 만들지 마십시오. 28
5.6 host의 RAM 한계 알아보기 28
5.7 충분히 큰 DMA buffers 29
5.8 데이터 워드에 올바른 너비를 사용하십시오. 29
5.9 cache synchronization로 인한 속도 저하 30
5.10 매개변수 조정 30

6 문제 해결 **32**

A Linux command line에 대한 짧은 생존 가이드 **33**
A.1 일부 키 입력 33
A.2 도움을 받다 34
A.3 파일 표시 및 편집 34
A.4 root 사용자 35
A.5 선택한 명령 36

1

소개

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

단순화를 위해 host는 compilation을 수행할 수 있는 완전한 기능을 갖춘 컴퓨터라고 가정합니다. embedded platform의 절차는 비슷하지만 몇 가지 직접적인 차이점이 있습니다(특히 cross compilation이 필요할 수 있음).

또한 이 안내서는 Xillybus의 demo bundle을 기반으로 하는 bitstream이 이미 FPGA에 로드되었고 FPGA가 host에서 peripheral로 인식되었다고 가정합니다(해당하는 경우 PCI Express, AXI bus 또는 USB 3.*x*를 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
- Getting started with Xillinux for Zynq-7000
- Getting started with Xillinux for Cyclone V SoC (SoCKit)

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 AXI 인터페이스용입니다. XillyUSB용 driver도 있습니다.

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 /dev/xillybus_*something* 형식의 파일 이름으로 생성됩니다. 마찬가지로 XillyUSB용 driver는 /dev/xillyusb_00_*something* 형식으로 device files를 생성합니다. 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 Linux에서 찾을 수 있습니다.

2

host driver 설치

2.1 Xillybus의 driver 설치 단계

Linux kernel driver 설치는 다음 단계로 구성됩니다.

- 설치가 전혀 필요한지 확인합니다. 그렇지 않은 경우 아래의 다른 단계를 건너뜁니다(마지막 단계, 즉 udev 파일 복사를 건너뛰지 않을 수 있음).
- 전제 조건 확인(예: compiler 및 kernel headers가 설치되어 있는지 확인)
- driver를 kernel module로 포함하는 다운로드한 파일의 압축을 풉니다.
- kernel module의 compilation 실행
- kernel module 설치
- udev 파일을 설치하여 Xillybus device files가 모든 사용자(root뿐만 아니라)에 액세스할 수 있도록 합니다.

이러한 단계는 command-line(“Terminal”)를 사용하여 수행됩니다. Appendix A의 짧은 Linux 서바이벌 가이드는 이 인터페이스에 대한 경험이 적은 사람들에게 도움이 될 수 있습니다.

2.2 정말 설치해야 할 것이 있습니까?

2.2.1 일반적인

대부분의 Linux kernels 및 Linux 배포판은 별도의 작업 없이 Xillybus(PCIe 또는 AXI용)를 지원합니다. 이에 대해서는 아래에서 자세히 설명합니다.

즉, Xillybus가 이미 지원되는 경우에도 udev 파일 설치에 관한 2.7 섹션을 살펴볼 가치가 있습니다.

XillyUSB용 driver는 5.14 버전(2021년 8월 출시)의 Linux kernel 일부입니다.

2.2.2 driver가 사전 설치된 Linux distributions

대부분의 Linux 배포판에는 PCIe / AXI Xillybus driver가 이미 설치되어 있습니다(“out of the box”). 예를 들어:

- Ubuntu 14.04 이상
- 최근의 Fedora 배포판
- Xillinux(Zynq 및 Cyclone V SoC 플랫폼 전용)

driver가 설치되어 있는지 빠르게 확인하려면 shell prompt에 다음을 입력하십시오.

```
$ modinfo xillybus_core
```

driver가 설치된 경우 이에 대한 정보가 인쇄됩니다. 그렇지 않으면 “modinfo: ERROR: Module xillybus_core not found”라고 표시됩니다.

마찬가지로 XillyUSB의 driver를 확인하려면 다음 명령을 실행합니다.

```
$ modinfo xillyusb
```

XillyUSB는 Ubuntu 22.04 이상, Fedora 35 이상 및 이 둘에서 파생된 배포판과 함께 설치할 필요 없이 작동합니다.

Linux가 virtual machine 내에서 실행되는 경우 PCIe bus에서 Xillybus를 감지하지 못합니다. driver가 있는 운영 체제는 bare metal에서 실행되어야 합니다. XillyUSB는 virtual machine 내부에서 작동할 수 있습니다.

섹션 2.7는 Xillybus device files의 permissions를 영구적으로 변경하는 방법을 보여줍니다. 이 변경으로 모든 사용자가 이러한 파일에 액세스할 수 있습니다(root 사용자뿐만 아니라). 이 수정은 데스크탑 컴퓨터에서 Xillybus를 사용할 때 종종 필요합니다.

2.2.3 Xillybus의 driver를 포함하는 Linux kernels

Xillybus의 driver(PCIe 및 AXI용)는 버전 3.12부터 공식 Linux kernel에 포함됩니다. 3.12와 3.17 사이 버전의 kernels에서 driver는 “staging driver”로 포함되었으며, 이는 Linux 커뮤니티가 새로운 driver를 완전히 수용하기 전의 예비 단계입니다. Xillybus의

driver는 3.18 버전에서 non-staging로 인정되었습니다. coding style와 관련된 몇 가지 변경 사항에도 불구하고 초기 driver(kernel의 3.12 버전)와 현재 사용 가능한 driver 사이에는 기능적 차이가 거의 없습니다.

staging driver가 로드되면 kernel은 system log에서 경고를 발행합니다. 이 경고는 driver의 품질을 알 수 없다고 말합니다. Xillybus와 관련하여 이 경고는 무시해도 됩니다.

위에서 언급했듯이 XillyUSB용 driver는 5.14 버전부터 Linux kernel에 포함됩니다.

Linux distribution의 일부인 kernels 관련: Xillybus의 drivers가 kernel의 source code의 일부인 경우에도 이러한 drivers는 kernel이 이러한 drivers를 포함하도록 구성된 경우에만 compilation에 포함됩니다. Xillybus의 drivers는 대부분의 주류 Linux distributions에 kernel modules로 포함되어 있지만 각 distribution에는 kernel에 포함할 항목을 선택하는 자체 기준이 있습니다. 따라서 Xillybus는 distribution와 함께 도착하는 kernel에 포함되지 않을 수 있습니다.

이 가이드는 kernel modules의 별도 compilation을 사용하여 drivers를 설치하는 데 중점을 둡니다. 이것은 일반적으로 가장 쉬운 방법입니다. 그러나 어쨌든 kernel의 compilation을 수행하는 사용자는 대신 Xillybus의 drivers를 포함하도록 kernel을 구성하는 것을 선호할 수 있습니다. 이 방법은 섹션 2.9에서 설명합니다.

2.3 전제 조건 확인

Linux 시스템에는 kernel module compilation용 기본 도구가 없을 수 있습니다. 이러한 도구가 있는지 확인하는 가장 간단한 방법은 도구를 실행해 보는 것입니다. 예를 들어 command prompt에 “make coffee”를 입력합니다. 이것이 정답입니다.

```
$ make coffee

make: *** No rule to make target 'coffee'.  Stop.
```

오류이긴 하지만 “make” 유틸리티가 존재함을 알 수 있습니다. 그러나 GNU make이 없어서 설치해야 하는 경우 출력은 다음과 같습니다.

```
$ make coffee
bash: make: command not found
```

C compiler도 필요합니다. compiler가 설치되어 있는지 확인하려면 “gcc”를 입력하십시오.

```
$ gcc
gcc: no input files
```

이 응답은 “gcc”가 설치되었음을 나타냅니다. 다시 오류 메시지가 표시되었지만 “command not found”는 표시되지 않았습니다.

이 두 도구 위에 kernel headers도 설치해야 합니다. 이것은 확인하기가 조금 더 어렵습니다. 이러한 파일이 누락되었는지 확인하는 일반적인 방법은 header file이 누락되었다는 오류와 함께 kernel compilation이 실패하는 경우입니다.

kernel module compilation은 일반적인 작업이므로 compilation용 시스템을 준비하는 방법과 관련하여 각 Linux distribution에 특정한 정보가 인터넷에 많이 있습니다.

Fedora, RHEL, CentOS 및 Red Hat의 기타 파생 제품에서 이러한 종류의 명령은 컴퓨터를 준비시킬 수 있습니다.

```
# yum install gcc make kernel-devel-$(uname -r)
```

Ubuntu 및 Debian을 기반으로 하는 기타 배포의 경우:

```
# apt install gcc make linux-headers-$(uname -r)
```

중요한:

이러한 설치 명령은 root로 실행해야 합니다. root 사용자라는 개념에 익숙하지 않은 사람들은 먼저 그것에 대해 배우기를 촉구합니다. 부록’ 섹션 A.4를 참조하십시오.

2.4 다운로드한 파일 압축 해제

Xillybus 사이트에서 driver를 다운로드한 후 다운로드한 파일이 있는 디렉토리로 변경합니다. command prompt에서 다음을 입력합니다($ 기호 제외).

```
$ tar -xzf xillybus.tar.gz
```

XillyUSB driver의 경우:

```
$ tar -xzf xillyusb.tar.gz
```

응답이 없어야 합니다. 새로운 command prompt만 있을 뿐입니다.

2.5 kernel module의 compilation 실행

kernel module의 source code가 있는 디렉토리로 변경합니다. Xillybus driver의 경우:

```
$ cd xillybus/module
```

그리고 XillyUSB driver의 경우:

```
$ cd xillyusb/driver
```

모듈의 compilation을 수행하려면 “make”을 입력하십시오. 성적표는 다음과 같아야 합니다.

```
$ make
make -C /lib/modules/3.10.0/build SUBDIRS=/home/myself/xillybus/module modules
make[1]: Entering directory '/usr/src/kernels/3.10.0'
  CC [M]  /home/myself/xillybus/module/xillybus_core.o
  CC [M]  /home/myself/xillybus/module/xillybus_pcie.o
  Building modules, stage 2.
  MODPOST 2 modules
  CC      /home/myself/xillybus/module/xillybus_core.mod.o
  LD [M]  /home/myself/xillybus/module/xillybus_core.ko
  CC      /home/myself/xillybus/module/xillybus_pcie.mod.o
  LD [M]  /home/myself/xillybus/module/xillybus_pcie.ko
make[1]: Leaving directory '/usr/src/kernels/3.10.0'
```

세부 사항은 약간 다를 수 있지만 오류 또는 warnings가 나타나지 않아야 합니다. XillyUSB의 경우 xillyusb.ko라는 단일 모듈만 생성됩니다.

kernel modules의 compilation은 compilation 중에 실행 중인 kernel에만 해당됩니다.

다른 kernel을 사용하려는 경우 “make TARGET=kernel-version”을 입력합니다. 여기서 “kernel-version”은 필요한 kernel 버전의 이름입니다. /lib/modules/에 나오는 이름입니다. 또는 이름이 “Makefile”인 파일에서 다음 줄을 편집합니다.

```
KDIR := /lib/modules/$(TARGET)/build
```

KDIR의 값을 필요한 kernel headers의 path로 변경합니다.

2.6 kernel module 설치

디렉터리를 변경하지 않고 사용자를 root로 변경합니다(예: “sudo su” 사용). 그런 다음 다음 명령을 입력합니다.

```
# make install
```

이 명령을 완료하는 데 몇 초가 걸릴 수 있지만 오류가 발생해서는 안 됩니다.

이것이 실패하면 compilation에서 생성된 *.ko 파일을 kernel modules의 기존 하위 디렉토리에 복사하십시오. 그런 다음 depmod를 실행합니다. 다음 예는 kernel의 관련 버전이 3.10.0인 경우 PCIe driver에 대해 이 작업을 수행하는 방법을 보여줍니다.

```
# cp xillybus_core.ko /lib/modules/3.10.0/kernel/drivers/char/
# cp xillybus_pcie.ko /lib/modules/3.10.0/kernel/drivers/char/

# depmod -a
```

설치 시 모듈이 kernel에 즉시 로드되지 않습니다. 이것은 Xillybus 주변 장치가 발견되면 시스템의 다음 boot에서 수행됩니다. 모듈을 수동으로 로드하는 방법은 2.8 섹션에 나와 있습니다.

XillyUSB의 경우 reboot가 필요하지 않습니다. 다음에 USB device가 컴퓨터에 연결되면 모듈이 자동으로 로드됩니다.

2.7 udev rule 파일 복사

기본적으로 Xillybus device files는 소유자인 root만 액세스할 수 있습니다. root로 작업하는 것을 피할 수 있도록 모든 사용자가 이러한 파일에 액세스할 수 있도록 하는 것이 좋습니다. udev 메커니즘은 특정 규칙을 준수하여 device files가 생성될 때 file permissions를 변경합니다.

이 기능을 활성화하는 방법: 동일한 디렉토리에 남아 있고 root 사용자로 남아 있습니다. udev rule 파일을 해당 파일이 저장된 시스템의 위치(대부분 /etc/udev/rules.d/)에 복사합니다.

예를 들어:

```
# cp 10-xillybus.rules /etc/udev/rules.d/
```

이 파일의 내용은 간단합니다.

```
SUBSYSTEM=="xillybus", MODE="666", OPTIONS="last_rule"
```

즉, Xillybus device driver에서 생성되는 모든 파일에는 permission mode 0666이 지정되어야 합니다. 즉, 모든 사람에게 읽기와 쓰기가 허용됩니다.

XillyUSB의 경우 파일은 10-xillyusb.rules이며 다음을 포함합니다.

```
SUBSYSTEM=="xilly*", KERNEL=="xillyusb_*", MODE="0666"
```

다른 결과를 얻기 위해 udev 파일을 변경할 수 있습니다. 예를 들어 device files의 소유자를 대신 변경하여 특정 사용자만 이 파일에 액세스할 수 있도록 할 수 있습니다.

2.8 모듈 로드 및 언로드

모듈을 로드하고 Xillybus 작업을 시작하려면 root로 입력하십시오.

```
# modprobe xillybus_pcie
```

또는 XillyUSB의 경우:

```
# modprobe xillyusb
```

이렇게 하면 Xillybus device files가 나타납니다(bus에서 Xillybus device가 감지되었다고 가정).

시스템이 boot 프로세스를 수행할 때 Xillybus PCIe / AXI 주변 장치가 감지되고 위에서 설명한 대로 driver가 이미 설치된 경우에는 이 작업이 필요하지 않습니다. driver가 이미 설치된 상태에서 XillyUSB device가 컴퓨터에 연결된 경우에도 이 작업이 필요하지 않습니다.

kernel의 모듈 목록을 보려면 “lsmod”를 입력하십시오. kernel에서 모듈을 제거하려면 (PCIe driver의 경우)를 입력합니다.

```
# rmmod xillybus_pcie xillybus_core
```

이렇게 하면 device files가 사라집니다.

문제가 있는 것 같으면 /var/log/syslog 파일에서 해당하는 경우 “xillybus” 또는 “xillyusb”라는 단어가 포함된 메시지를 확인하십시오. 이 로그 파일에서 종종 귀중한 단서를 찾을 수 있습니다. 동일한 로그 정보는 “dmesg” 명령으로도 액세스할 수 있습니다.

/var/log/syslog 로그 파일이 없으면 대신 /var/log/messages일 수 있습니다. “journalctl -k” 명령을 시도해 보십시오.

2.9 공식 Linux kernel의 Xillybus drivers

앞서 언급했듯이 Xillybus용 driver는 v3.12.0 버전부터 Linux kernel에 포함됩니다. 따라서 전체 kernel의 compilation을 수행할 수 있으므로 이 kernel은 Xillybus를 지원합니다. 이것은 위와 같이 kernel modules를 별도로 설치하는 것의 대안입니다.

기능적 관점에서 kernel compilation을 포함하는 방법은 섹션 2.3에서 2.6에 설명된 단계와 동일한 결과를 생성합니다.

compilation을 대상으로 하는 kernel에 Xillybus의 driver를 포함시키려면 몇 개의 kernel configuration options를 활성화해야 합니다. driver를 포함하는 방법에는 두 가지가 있습니다. kernel modules 또는 kernel image의 일부로.

예를 들어 kernel의 구성 파일(.config)에서 PCIe 인터페이스에 대해 Xillybus의 driver를 활성화하는 부분입니다.

```
CONFIG_XILLYBUS=m
CONFIG_XILLYBUS_PCIE=m
```

“m”은 driver가 kernel module로 포함된다는 의미입니다. “y”는 kernel image에 driver를 포함한다는 의미입니다.

마찬가지로 XillyUSB(kernel v5.14 이상 포함)의 경우:

```
CONFIG_XILLYUSB=m
```

.config을 변경하는 일반적인 방법은 kernel의 구성 도구를 사용하는 것입니다. “make config”, “make xconfig” 또는 “make gconfig”.

xconfig 및 gconfig은 Xillybus의 drivers를 찾기 위해 문자열 “xillybus”를 검색할 수 있기 때문에 사용하기 쉬운 GUI 도구입니다. 확인란을 클릭하면 driver가 활성화됩니다. .config 파일의 텍스트 표현은 올바른 옵션이 설정되었는지 확인하는 데 도움이 됩니다.

3.18 미만 버전의 kernels에서 Xillybus를 활성화하기 전에 staging drivers를 활성화해야 할 수 있습니다. 그 결과 .config 파일에 다음 줄이 생깁니다.

```
CONFIG_STAGING=y
```

.config 파일에서 Xillybus의 driver를 활성화한 후 평소처럼 kernel compilation을 실행합니다.

kernel 5.14부터 Xillybus 또는 XillyUSB용 driver가 활성화되면 이름이 CONFIG_XILLYBUS_CLASS인 옵션이 자동으로 활성화됩니다. 이는 구성 시스템의 종속성 규칙의 결과입니다. 따라서 이 옵션을 수동으로 변경하는 것은 불필요합니다(종종 불가능함).

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 4.4 섹션을 참조하십시오.

3.2 준비

"Hello world" 테스트를 수행하려면 몇 가지 준비 작업이 필요합니다.

- 2 섹션에 설명된 대로 Xillybus의 driver가 컴퓨터에 설치됩니다.
- FPGA는 수정 없이 demo bundle에서 생성된 bitstream와 함께 로드되어야 합니다. 이를 달성하는 방법은 Getting started with the FPGA demo bundle for Xilinx 또는 Getting started with the FPGA demo bundle for Intel FPGA에 설명되어 있습니다.

  Xillinux(Zynq 또는 Cyclone V SoC 포함)를 사용하는 사용자는 Getting started with Xillinux for Zynq-7000 또는 Getting started with Xillinux for Cyclone V SoC (SoCKit)를 참조하십시오. demo bundle은 기본적으로 이 시스템에 이미 포함되어 있습니다.
- PCIe에만 해당: FPGA는 컴퓨터가 boot를 수행할 때 PCIe bus에서 감지되었습니다. 이는 "lspci" 명령을 사용하여 확인할 수 있습니다.
- USB에만 해당: FPGA는 USB port를 통해 컴퓨터에 연결되어 있고 컴퓨터는 FPGA를 USB device로 감지했습니다. 이는 "lsusb" 명령을 사용하여 확인할 수 있습니다.
- Linux command-line을 사용하는 것이 편해야 합니다. 부록 A이 도움이 될 수 있습니다.

이러한 준비가 올바르게 완료되면 Xillybus의 device files를 사용할 수 있습니다. 예를 들어 /dev/xillybus_read_8라는 파일이 있어야 합니다.

3.3 사소한 loopback 테스트

이 테스트를 수행하는 가장 쉬운 방법은 "cat"라는 이름의 Linux command-line utility를 사용하는 것입니다.

두 개의 terminal windows를 엽니다. 일부 컴퓨터에서는 이름이 "Terminal"인 아이콘을 두 번 클릭하여 이 작업을 수행할 수 있습니다. 해당 아이콘이 없으면 바탕 화면 메뉴에서 검색하십시오.

첫 번째 terminal window에서 command prompt에 다음 명령을 입력합니다(달러 기호는 입력하지 마십시오. prompt입니다).

```
$ cat /dev/xillybus_read_8
```

이렇게 하면 “cat” 프로그램이 xillybus_read_8 device file에서 읽은 모든 내용을 출력합니다. 이 단계에서는 아무 일도 일어나지 않을 것으로 예상됩니다.

XillyUSB를 사용하는 사용자는 대신 device file을 xillyusb_00_read_8로 찾을 수 있습니다. “xillyusb” 접두사는 명백하며 “00” 인덱스는 여러 USB devices를 동일한 host에 연결할 수 있도록 하기 위한 것입니다. 이 가이드에서는 PCIe 및 AXI의 명명 규칙을 사용합니다.

두 번째 terminal 창에서 다음을 입력합니다.

```
$ cat > /dev/xillybus_write_8
```

> 문자에 유의하십시오. console에 입력된 모든 것을 xillybus_write_8(redirection)로 보내도록 “cat”에 지시합니다.

이제 두 번째 terminal에 텍스트를 입력하고 ENTER를 누릅니다. 동일한 텍스트가 첫 번째 terminal에 나타납니다. ENTER를 누를 때까지 아무 것도 xillybus_write_8로 전송되지 않습니다. 이것은 Linux 컴퓨터의 일반적인 규칙입니다.

이 두 “cat” 명령은 모두 CTRL-C로 중지할 수 있습니다.

이 두 “cat” 명령을 시도하는 동안 오류 메시지가 나타나면 먼저 device files가 생성되었는지 확인하십시오(즉, /dev/xillybus_read_8 및 /dev/xillybus_write_8이 존재하는지). 또한 오타가 없는지 확인하십시오.

오류가 “permission denied”인 경우 섹션 2.7에 표시된 대로 수정할 수 있습니다. 그러나 udev 파일은 kernel modules가 kernel에 로드된 경우에만 유효하다는 점에 유의하십시오. kernel modules를 다시 로드하는 방법은 2.8 섹션을 참조하십시오. 또는 컴퓨터에서 reboot를 수행합니다.

“permission denied” 오류를 극복할 수 있는 또 다른 가능성은 root 사용자로 Xillybus device files를 사용하는 것입니다. 데스크톱 컴퓨터에서는 권장되지 않습니다(하지만 일반적으로 embedded platforms에서 수행됨). 이에 대한 자세한 내용은 부록 섹션 A.4를 참조하십시오.

다른 오류의 경우 /var/log/syslog에서 더 많은 정보를 찾거나 “dmesg” 명령(또는 kernel log을 얻기 위한 유사한 방법)을 사용하여 섹션 2.8의 지침을 따르십시오.

사소한 파일 작업을 수행하는 것도 가능합니다. 예를 들어 첫 번째 terminal에서 “cat” 명령을 중지하지 않고 두 번째 terminal에 다음을 입력합니다.

```
$ date > /dev/xillybus_write_8
```

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.

그들 사이에 loopback이 있는 또 다른 쌍의 device files가 있습니다./dev/xillybus_read_32 및 /dev/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.

4

host 애플리케이션의 예

4.1 일반적인

Xillybus의 device files에 액세스하는 방법을 보여주는 간단한 C 프로그램이 4 5개 있습니다. 이러한 프로그램은 Xillybus / XillyUSB용 host driver가 포함된 압축 파일에서 찾을 수 있습니다(웹 사이트에서 다운로드 가능). 다음 파일로 구성된 “demoapps” 디렉토리를 참조하십시오.

- Makefile– 이 파일에는 프로그램의 compilation을 위해 “make” 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.

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 간단하며 Linux 컴퓨터에서 파일에 액세스하는 표준 방법을 보여줄 뿐입니다. 이러한 방법은 Xillybus host application programming guide for Linux에서 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.

PCIe용 driver를 다운로드하는 사용자는 “demoapps” 디렉토리에서 다섯 번째 프로그램을 찾을 수 있습니다.fifo.c. 이 프로그램은 userspace RAM FIFO의 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.

이 프로그램은 XillyUSB의 driver와 함께 포함되지 않습니다. fifo.c가 필요할 수 있는 데이터 속도가 XillyUSB에서는 불가능하기 때문입니다.

4.2 편집 및 compilation

Linux에서 compilation 프로그램을 사용해 본 경험이 있다면 다음 섹션으로 건너뛸 수 있습니다. Xillybus의 예제 프로그램 중 compilation은 일반적인 방식으로 “make”와 함께 수행됩니다.

가장 먼저 C 파일이 있는 디렉토리로 변경하십시오.

```
$ cd demoapps
```

5개 프로그램 모두의 compilation을 실행하려면 shell prompt에 “make”을 입력하면 됩니다. 다음 성적표가 예상됩니다.

```
$ make
gcc  -g -Wall -O3 memwrite.c -o memwrite
gcc  -g -Wall -O3 memread.c -o memread
gcc  -g -Wall -O3 streamread.c -o streamread
gcc  -g -Wall -O3 streamwrite.c -o streamwrite
gcc  -g -Wall -O3 -pthread fifo.c -o fifo
```

“gcc”로 시작하는 5개의 행은 compiler를 사용하기 위해 “make”에서 요청하는 명령입니다. 이 명령은 프로그램의 compilation에 별도로 사용할 수 있습니다. 그러나 그렇게 할 이유가 없습니다. 그냥 “make”을 사용하세요.

일부 시스템에서는 POSIX threads library가 설치되지 않은 경우(예: Cygwin의 일부 설치에서) 다섯 번째 compilation(fifo.c 의)가 실패할 수 있습니다. fifo.c를 사용할 의도가 없다면 이 오류는 무시해도 됩니다.

"make" 유틸리티는 필요한 경우에만 compilation을 실행합니다. 하나의 파일만 변경되면 "make"은 해당 파일의 compilation만 요청합니다. 따라서 정상적인 작업 방법은 편집하려는 파일을 편집한 다음 recompilation용 "make"을 사용하는 것입니다. 불필요한 compilation이 발생하지 않습니다.

이전 compilation에서 생성된 executables를 제거하려면 "make 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 "make"도 이 새 파일의 compilation을 실행할 수 있습니다. 예를 들어 memwrite.c가 mygames.c라는 새 파일에 복사되었다고 가정합니다. 이는 GUI 인터페이스 또는 command line을 사용하여 수행할 수 있습니다.

```
$ cp memwrite.c mygames.c
```

다음 단계는 Makefile을 편집하는 것입니다. 많은 텍스트 편집기와 각 편집기를 실행하는 다양한 방법이 있습니다. 대부분의 시스템에서 "gedit" 또는 "xed"를 입력하여 shell prompt에서 GUI editor를 시작할 수 있습니다. 그러나 컴퓨터 바탕 화면의 메뉴에서 GUI text editor를 찾는 것이 더 쉽습니다. 또한 vim, emacs, nano 및 pico와 같이 terminal window 내에서 작동하는 많은 텍스트 편집기가 있습니다.

어떤 editor를 사용할지는 취향과 개인적인 경험의 문제입니다. 예를 들어 다음 명령을 사용하여 Makefile 편집을 시작할 수 있습니다.

```
$ xed Makefile &
```

명령 끝에 있는 '&'는 프로그램이 완료될 때까지 기다리지 않도록 shell에 지시합니다. 다음 shell prompt가 즉시 나타납니다. 이는 무엇보다도 GUI 애플리케이션을 시작하는 데 적합합니다.

Makefile에서 변경해야 하는 행은 다음과 같습니다.

```
APPLICATIONS=memwrite memread streamread streamwrite
```

이 행은 다음으로 변경됩니다.

```
APPLICATIONS=memwrite memread streamread streamwrite mygames
```

다음에 "make"을 입력하면 mygames.c의 compilation이 실행됩니다.

4.3 프로그램 실행

섹션 3.3에 표시된 간단한 loopback 예제는 두 가지 예제 프로그램으로 수행할 수 있습니다.

"demoapps"가 이미 current directory이고 compilation이 이미 "make'로 완료되었다고 가정해 봅시다.

첫 번째 terminal에 다음을 입력합니다.

```
$ ./streamread /dev/xillybus_read_8
```

이것은 device file에서 읽는 프로그램입니다.

명령은 "./"로 시작합니다. executable의 디렉토리를 명시적으로 지정해야 합니다. 이 예에서 "./" 표현식은 current directory를 요청하는 데 사용됩니다.

그런 다음 두 번째 terminal window에서:

```
$ ./streamwrite /dev/xillybus_write_8
```

이것은 "cat"의 예와 거의 비슷하게 작동합니다. 차이점은 "streamwrite"가 device file로 데이터를 보내기 전에 ENTER를 기다리지 않는다는 것입니다. 대신 이 프로그램은 각 character에서 개별적으로 작동하려고 시도합니다. 이를 달성하기 위해 프로그램은 config_console()라는 기능을 사용합니다. 이 기능은 키보드 입력에 대한 즉각적인 응답의 목적으로만 사용됩니다. 이것은 Xillybus와 아무 관련이 없습니다.

위의 예는 PCIe / AXI용 Xillybus와 관련이 있습니다. XillyUSB에서 device files의 이름은 접두사가 약간 다릅니다. 예를 들어, xillybus_read_8 대신 xillyusb_00_read_8입니다.

중요한:

streamread 및 streamwrite에서 수행되는 I/O 작업은 비효율적입니다. 이러한 프로그램을 더 간단하게 만들기 위해 I/O buffer의 크기는 128바이트에 불과합니다. 높은 데이터 속도가 필요한 경우 더 큰 buffers를 사용해야 합니다. 섹션 5.3를 참조하십시오.

4.4 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 Linux에는 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.

메모리에 쓰기 전에 hexdump 유틸리티를 사용하여 기존 상황을 관찰할 수 있습니다.

```
$ hexdump -C -v -n 32 /dev/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 /dev/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 /dev/xillybus_mem_8 3 170
```

명령이 작동하는지 확인하기 위해 위에서 hexdump 명령을 반복할 수 있습니다.

```
$ hexdump -C -v -n 32 /dev/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. 이미 언급했듯이 lseek()을 허용하는 파일을 열면 초기 위치는 항상 0이다. 이는 다음 예와 같은 명령에도 적용됩니다.

```
$ echo -n 010111 > /dev/xillybus_mem_8
```

"echo" 명령에서 "-n" 부분을 참고하세요. 이는 "echo"가 출력 끝에 newline character를 추가하는 것을 방지합니다.

이 명령은 "0"의 ASCII code(값 0x30)를 주소 0에 씁니다. 마찬가지로 0x31 값은 1 주소에 기록됩니다. 따라서 이 간단한 "echo" 명령을 사용하여 여러 registers의 값을 한 번에 설정할 수 있습니다.

FPGA의 구현이 간단하기 때문에 이는 편리한 방법입니다. 예를 들어 "0" 및 "1" 문자만 "echo" 명령과 함께 사용한다고 가정합니다. 따라서 bit 0만이 중요합니다. 다음은 "echo" 명령의 세 번째 바이트에서 해당 값을 얻는 register의 예입니다.

```
reg my_register;

always @(posedge bus_clk)
  if (user_w_mem_8_wren && (user_mem_8_addr == 2))
    my_register <= user_w_mem_8_data[0];
```

이 방법은 특히 FPGA의 logic을 개발하는 동안 테스트를 실행하는 데 적합합니다.

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 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 /dev/xillybus_read_32를 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 /dev/xillybus_write_32를 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 4.3(streamread 및 streamwrite)에서 언급한 예제 프로그램은 성능 측정에 적합하지 않다는 점에 유의해야 합니다. 이러한 프로그램의 buffer 크기는 128바이트입니다(kB

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

속도 확인을 위해 다음 shell 명령을 사용할 수 있습니다(필요에 따라 /dev/xillybus_* names 교체).

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

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

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 소비를 주시하는 것이 중요합니다. “top”와 같은 유틸리티 프로그램을 이 용도로 사용할 수 있습니다. 그러나 이 프로그램의 출력(및 유사한 대안)은 여러 processor cores가 있는 컴퓨터(즉, 오늘날 거의 모든 컴퓨터)에서 오해의 소지가 있을 수 있습니다. 예를 들어 processor cores가 4개 있는 경우 25% CPU는 무엇을 의미합니까? 낮은 CPU 소비입니까, 아니면 특정 thread의 100%입니까? “top”을 사용하는 경우 프로그램 버전에 따라 다릅니다.

주목해야 할 또 다른 사항은 system calls의 처리 시간을 측정하고 표시하는 방법입니다. 운영 체제의 overhead가 데이터 흐름을 늦추는 경우 이를 어떻게 측정합니까?

이것을 검사하는 간단한 방법은 “time” 유틸리티를 사용하는 것입니다. 예를 들어,

```
$ time dd if=/dev/zero of=/dev/null bs=128k count=100k
102400+0 records in
102400+0 records out
13421772800 bytes (13 GB) copied, 1.07802 s, 12.5 GB/s

real 0m1.080s
```

```
user 0m0.005s
sys 0m1.074s
```

하단의 “time” 출력은 “dd”가 완료되는 데 걸린 시간이 1.080초임을 나타냅니다. 이 시간 중 processor는 5 ms 동안 user space program을 수행했으며, system calls로 1.074초 동안 Busy했습니다. 따라서 이 특정 예에서는 processor가 거의 항상 system calls를 수행하느라 바빴다는 것이 분명합니다. “dd”는 여기에서 아무 것도 하지 않기 때문에 이것은 놀라운 일이 아닙니다.

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 자주 중단됩니다. Xillybus host application programming guide for Linux 의 섹션 6.6은 이 주제에 대해 자세히 설명하고 보다 적절한 프로그래밍 기술을 제안합니다.

5.6 host의 RAM 한계 알아보기

이는 주로 embedded systems 및/또는 revision XL / XXL IP core를 사용할 때와 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 Internals”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 cache synchronization로 인한 속도 저하

이 문제는 x86 제품군(32비트 및 64비트)에 속하는 CPUs 기반 컴퓨터에는 적용되지 않습니다. Zynq processor의 AXI bus와 함께 Xillybus를 사용하는 사용자(예: Xillinux)는 이 주제를 무시해도 됩니다.

그러나 여러 embedded processors는 DMA buffers를 사용할 때 cache의 명시적 동기화가 필요합니다. 이로 인해 CPU의 주변 장치와의 데이터 전송 속도가 상당히 느려집니다.

이 문제는 Xillybus에만 국한되지 않습니다. 유사한 동작이 DMA를 기반으로 하는 모든 I/O(예: Ethernet, USB 및 기타 주변 장치)에서 관찰됩니다.

cache로 인한 둔화는 CPU 소비량을 보면 알 수 있습니다. CPU가 system call 상태("time" 유틸리티의 "sys" 행 출력)에서 비합리적인 시간을 보내는 경우 cache에 문제가 있음을 나타낼 수 있습니다. 이는 CPU가 cache synchronization를 수행하는 데 많은 시간을 소비하기 때문에 발생합니다.

그러나 먼저 작은 buffers의 가능성을 배제하는 것이 중요합니다(위의 5.3 및 5.7 섹션에서 언급).

이러한 CPUs에는 coherent cache가 있기 때문에 x86 제품군에서는 이 문제가 발생하지 않습니다. 따라서 cache synchronization가 필요하지 않습니다. Xillinux도 마찬가지입니다. IP core는 ACP port를 통해 CPU에 연결되기 때문입니다.

그러나 Zynq processor가 PCIe bus와 함께 Xillybus를 사용하면 이 문제가 발생합니다. 몇몇 다른 embedded processors, 특히 ARM processors도 영향을 받습니다.

5.10 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 의미 있는 log messages를 생산하도록 설계되었습니다. 이를 얻기 위한 몇 가지 대안은 다음과 같습니다.

- “dmesg” 명령의 출력.
- “journalctl -k” 명령의 출력.
- log file에서. 이는 운영 체제(예: /var/log/syslog 또는 /var/log/messages)에 따라 다릅니다.

뭔가 잘못된 것 같으면 “xillybus” 또는 “xillyusb”라는 단어가 포함된 메시지를 검색하는 것이 좋습니다. 또한 모든 것이 잘 작동하는 것처럼 보이더라도 때때로 system log을 검사하는 것이 좋습니다.

PCIe / AXI 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.

A

Linux command line에 대한 짧은 생존 가이드

command line 인터페이스에 익숙하지 않은 사람들은 Linux 컴퓨터에서 작업을 수행하는 데 어려움을 겪을 수 있습니다. 기본 command-line 인터페이스는 30년 이상 동일했습니다. 따라서 각각의 모든 명령을 사용하는 방법에 대한 많은 온라인 자습서가 있습니다. 이 짧은 가이드는 시작일 뿐입니다.

A.1 일부 키 입력

이것은 가장 일반적으로 사용되는 키 입력의 요약입니다.

- CTRL-C: 현재 실행 중인 프로그램 중지
- CTRL-D: 이 세션 종료(terminal 창 닫기)
- CTRL-L: 화면 지우기
- TAB: command prompt에서 이미 작성된 것에 대해 autocomplete을 시도하십시오. 이는 예를 들어 긴 파일 이름에 유용합니다. 이름의 시작 부분을 입력한 다음 [TAB]을 입력합니다.
- 위쪽 및 아래쪽 화살표: command prompt에서 기록에서 이전 명령을 제안합니다. 이것은 방금 수행한 작업을 반복하는 데 유용합니다. 이전 명령의 편집도 가능하므로 이전 명령과 거의 동일하게 수행하는 데에도 좋습니다.
- space: 컴퓨터가 terminal pager로 무언가를 표시할 때 [space]는 “page down”을 의미합니다.
- q: “Quit”. 페이지별 표시에서 “q”는 이 모드를 종료하는 데 사용됩니다.

A.2 도움을 받다

flags와 옵션을 모두 기억하는 사람은 아무도 없습니다. 추가 도움을 받을 수 있는 두 가지 일반적인 방법이 있습니다. 한 가지 방법은 “man” 명령이고 두 번째 방법은 help flag입니다.

예를 들어, “ls” 명령에 대해 자세히 알아보려면(현재 디렉터리의 파일 나열):

```
$ man ls
```

’$’ 기호는 command prompt이며 컴퓨터는 명령을 받을 준비가 되었음을 알리기 위해 출력합니다. 일반적으로 prompt는 더 길며 사용자 및 현재 디렉터리에 대한 일부 정보를 포함합니다.

manual page는 terminal pager와 함께 표시됩니다. 탐색하려면 [space], 화살표 키, Page Up 및 Page Down을 사용하고 종료하려면 ’q’를 사용하십시오.

명령 실행 방법에 대한 간략한 요약을 보려면 `--help` 플래그를 사용하십시오. 일부 명령은 `-h` 또는 `-help`(단일 대시 사용)에 응답합니다. 다른 명령은 구문이 잘못된 경우 도움말 정보를 인쇄합니다. 시행 착오의 문제입니다. ls 명령의 경우:

```
$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort.

Mandatory arguments to long options are mandatory for short options too.
  -a, --all                  do not ignore entries starting with .
  -A, --almost-all           do not list implied . and ..
      --author               with -l, print the author of each file
```

...

(그리고 계속된다)

A.3 파일 표시 및 편집

파일이 짧을 것으로 예상되는 경우(또는 terminal window의 스크롤 막대를 사용해도 되는 경우) 다음과 같이 console에 파일 내용을 표시할 수 있습니다.

```
$ cat filename
```

더 긴 파일에는 terminal pager가 필요합니다.

```
$ less filename
```

텍스트 파일 편집의 경우 선택할 수 있는 편집기가 많이 있습니다. 가장 인기 있는(하지만 시작하기 가장 쉬운 것은 아님) emacs(및 XEmacs)와 vi가 있습니다. vi 편집기는 배우기 어렵지만 항상 사용할 수 있고 항상 작동합니다.

권장되는 단순 GUI 편집기는 gedit 또는 xed 중 사용 가능한 것입니다. 데스크탑 메뉴 또는 command line에서 시작할 수 있습니다.

```
$ gedit filename &
```

끝에 '&'는 명령이 "in the background"를 실행해야 함을 의미합니다. 또는 간단히 말해서 명령이 완료되기 전에 다음 command prompt가 나타납니다. GUI 응용 프로그램은 이와 같이 시작하는 것이 가장 좋습니다.

이 예에서 'filename'은 물론 path일 수도 있습니다. 예를 들어, 시스템의 기본 log file을 보려면:

```
# less /var/log/syslog
```

파일의 끝으로 이동하려면 "less"가 실행 중인 동안 shift-G를 누르십시오.

로그 파일은 일부 컴퓨터의 root 사용자만 액세스할 수 있습니다.

A.4 root 사용자

모든 Linux 컴퓨터에는 ID 0이 있는 "root"라는 이름의 사용자가 있습니다. 이 사용자는 superuser라고도 합니다. 이 사용자의 특별한 점은 모든 작업을 수행할 수 있다는 것입니다. 다른 모든 사용자는 파일 및 리소스 액세스에 제한이 있습니다. 모든 사용자에게 모든 작업이 허용되는 것은 아닙니다. 이러한 제한은 root 사용자에게는 적용되지 않습니다.

이는 다중 사용자 컴퓨터의 개인 정보 보호 문제만은 아닙니다. 모든 것을 할 수 있다는 것은 shell prompt에서 간단한 명령으로 하드 디스크의 모든 데이터를 삭제하는 것을 포함합니다. 여기에는 실수로 데이터를 삭제하거나 컴퓨터를 일반적으로 쓸모없게 만들거나 시스템을 공격에 취약하게 만드는 몇 가지 다른 방법이 포함됩니다. Linux 시스템의 기본 가정은 root 사용자가 자신이 무엇을 하고 있는지 알고 있다는 것입니다. 컴퓨터는 root에게 확실한 질문을 하지 않습니다.

소프트웨어 설치를 포함한 시스템 유지 관리를 위해 root로 작업해야 합니다. 일을 망치지 않는 요령은 ENTER를 누르기 전에 생각하고 명령이 정확히 필요한 대로 입력되었는지 확인하는 것입니다. 일반적으로 설치 지침을 정확히 따르는 것이 안전합니다. 그들이 정확히 무엇을 하고 있는지 이해하지 않고 수정하지 마십시오. root가 아닌 사용자로 동일한 명령

을 반복할 수 있는 경우(다른 파일이 포함될 수 있음) 해당 사용자로 어떤 일이 발생하는지 확인하십시오.

root의 위험 때문에 root로 명령을 실행하는 일반적인 방법은 sudo 명령을 사용하는 것입니다. 예를 들어 기본 로그 파일을 봅니다.

```
$ sudo less /var/log/syslog
```

사용자가 “sudo”를 사용할 수 있도록 시스템을 구성해야 하기 때문에 이것이 항상 작동하는 것은 아닙니다. 시스템에 사용자의 암호가 필요합니다(root 암호가 아님).

두 번째 방법은 “su”를 입력하고 모든 명령이 root로 주어지는 세션을 시작하는 것입니다. 이는 여러 작업을 root로 수행해야 할 때 편리하지만 root임을 잊어버리고 생각 없이 잘못된 것을 작성할 가능성이 더 크다는 의미이기도 합니다. root 세션을 짧게 유지하십시오.

```
$ su
Password:
# less /var/log/syslog
```

이번에는 root 암호가 필요합니다.

shell prompt의 변경은 일반 사용자에서 root로의 ID 변경을 나타냅니다. 의심스러운 경우 “whoami”를 입력하여 현재 user name을 가져옵니다.

일부 시스템에서는 sudo가 관련 사용자에 대해 작동하지만 root로 세션을 호출하는 것이 여전히 필요할 수 있습니다. “su”를 사용할 수 없는 경우(주로 root 암호를 알 수 없기 때문에) 간단한 대안은 다음과 같습니다.

```
$ sudo su
#
```

A.5 선택한 명령

마지막으로 다음은 일반적으로 사용되는 몇 가지 Linux 명령입니다.

몇 가지 파일 조작 명령(이를 위해 GUI 도구를 사용하는 것이 더 나을 수 있음):

- cp– 파일을 복사합니다.
- rm– 파일을 제거합니다.
- mv– 파일을 이동합니다.
- rmdir– 디렉토리를 제거합니다.

그리고 일반적으로 알아두는 것이 좋습니다.

- ls– 현재 디렉토리(또는 지정된 경우 다른 디렉토리)의 모든 파일을 나열합니다. “ls -l”은 속성과 함께 파일을 나열합니다.
- lspci– bus의 모든 PCI(및 PCIe) 장치를 나열합니다. Xillybus가 PCIe 주변 장치로 감지되었는지 확인하는 데 유용합니다. lspci -v, lspci -vv 및 lspci -n도 사용해 보십시오.
- lsusb– bus의 모든 USB 장치를 나열합니다. XillyUSB가 주변 장치로 감지되었는지 확인하는 데 유용합니다. lsusb -v 및 lsusb -vv도 사용해 보십시오.
- cd– 디렉토리 변경
- pwd– 현재 디렉토리 표시
- cat– 파일을 standard output로 보냅니다. 또는 argument가 제공되지 않으면 standard input을 사용하십시오. 이 명령의 원래 목적은 파일을 연결하는 것이었지만 파일에서 일반 입력 및 출력을 위한 스위스 칼로 끝났습니다.
- man– 명령의 manual page를 표시합니다. 또한 “man -a”를 사용해 보십시오(때로는 명령에 대해 둘 이상의 manual page가 있음).
- less– terminal pager. standard input의 파일 또는 데이터를 페이지별로 표시합니다. 위 참조. 또한 명령의 긴 출력을 표시하는 데 사용됩니다. 예를 들어:

  ```
  $ ls -l | less
  ```

- head– 파일 시작 표시
- tail– 파일의 끝을 표시합니다. 또는 -f 플래그를 사용하면 더 좋습니다. 도착할 때 끝 + 새 줄을 표시합니다. 예를 들어 로그 파일에 적합합니다(root).

  ```
  # tail -f /var/log/syslog
  ```

- diff– 두 텍스트 파일을 비교합니다. 아무 것도 표시되지 않으면 파일이 동일합니다.
- cmp– 두 개의 바이너리 파일을 비교합니다. 아무 것도 표시되지 않으면 파일이 동일합니다.
- hexdump– 파일 내용을 깔끔한 형식으로 표시합니다. 플래그 -v 및 -C가 선호됩니다.
- df– 마운트된 디스크와 각 디스크에 남은 공간을 표시합니다. 더 나은 “df -h”

- make– Makefile의 규칙에 따라 프로젝트 빌드(compilation 실행) 시도
- gcc– GNU C compiler.
- ps– 실행 중인 프로세스 목록을 가져옵니다. “ps a”, “ps au” 및 “ps aux”는 서로 다른 양의 정보를 제공합니다.

그리고 몇 가지 고급 명령:

- grep– 파일 또는 standard input에서 textual pattern을 검색합니다. 패턴은 regular expression이지만 텍스트일 경우 문자열을 검색합니다. 예를 들어, 기본 로그 파일에서 “xillybus”라는 단어를 case insensitive 문자열로 검색하고 페이지 다음에 출력 페이지를 표시합니다.

```
# grep -i xillybus /var/log/syslog | less
```

- find– 파일을 찾습니다. 복잡한 인수 구문이 있지만 이름, 나이, 유형 또는 생각할 수 있는 모든 항목에 따라 파일을 찾을 수 있습니다. man page를 참조하십시오.