(機械で日本語に翻訳)

Getting started with Xillybus on a Linux host

Xillybus Ltd.

www.xillybus.com

Version 3.0

この文書はコンピューターによって英語から自動的に翻訳されているため、言語が不明瞭になる可能性があります。このドキュメントは、元のドキュメントに比べて少し古くなっている可能性もあります。

可能であれば、英語のドキュメントを参照してください。

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 This document may also be slightly outdated in relation to the original.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

序章

これは、 Linux hostで Xillybus / XillyUSB を使用するために driver をインストールし、 IP coreの基本機能を試すためのウォークスルー ガイドです。

簡単にするために、 host は compilation 機能を備えた本格的なコンピューターであると想定されています。 embedded プラットフォームの手順は似ていますが、明確な違いがあります (例: cross compilation)。

このガイドでは、 FPGA に、 Xillybus または XillyUSBのいずれかの IP core を含む demo bundleの bitstreamがロードされており、 host によって周辺機器として認識されていることを前提としています (PCI Express、 AXI bus、または USB 3.xを介して、関連する場合)。

これに到達するための手順は、次のいずれかのドキュメントで概説されています (選択した FPGAによって異なります)。

- 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 driver は、 named pipesのように動作する device files を生成します。それらは、他のファイルと同じように開かれ、読み書きされますが、プロセス間または TCP/IP streams間では pipes のように動作します。 hostで実行されているプログラムにとっての違いは、 stream の反対側が別のプロセス (ネットワークまたは同じコンピューター上) ではなく、 FPGA内の FIFO であることです。 TCP/IP streamと同様に、 Xillybus stream は、高速データ転送だけでなく、時々送受信される単一バイトでもうまく機能するように設計されています。

1 つの driver バイナリが、 PCIe および AXI トランスポートのすべての Xillybus IP cores をカバーします。別の driver が XillyUSB用に指定されています。 streams とその属性は、 hostのオペレーティング システムに読み込まれると driver によって自動検出され、それに応じて device files が作成されます。これらの pipeに似た device files は /dev/xillybus_*something* (または /dev/xillyusb_*something* と XillyUSB) として表示されます。

ただし、 PCIe / AXI インターフェース用の driver と XillyUSB用の driver が 1 つずつあることに注意してください。

host に関連するトピックの詳細については、 Xillybus host application programming guide for Linuxを参照してください。

2

host driverの取り付け

2.1 Xillybusの driverをインストールするためのステージ

Linux kernel driver のインストールは、次の手順で構成されます。

- インストールがまったく必要かどうかを確認します。そうでない場合は、最後のステップ (udev ファイルのコピー) を除いて、以下の他のステップをスキップしてください。
- 前提条件の確認 (compiler および kernel headers がインストールされている)
- ダウンロードしたファイルの解凍
- kernel module の compilation を実行する
- kernel module の取り付け
- udev 構成ファイルをコピーして、すべてのユーザーが Xillybus device files にアクセスできるようにする

これは、コマンドライン インターフェイス (“Terminal”) を使用して行います。Appendix A の短い Linux サバイバル ガイドは、このインターフェイスの経験が少ない人に役立つ場合があります。

2.2 本当に何かをインストールする必要がありますか?

2.2.1 全般的

Xillybus (PCIe または AXI) の driver は、以下で説明するように、 Linux kernels およびディストリビューションの大部分の一部です。ただし、 udev ファイルのインス

トールに関する段落 2.7 は一見の価値があります。

XillyUSB 用の driver は、2021 年 8 月にリリースされたバージョン 5.14の Linux kernel の一部です。

2.2.2 driverがプリインストールされたLinux ディストリビューション

一部の Linux ディストリビューションには、 PCIe / AXI Xillybus driver がプリインストールされています (“out of the box”)。たとえば、次のようになります。

- Ubuntu 14.04 以降
- かなり最近の Fedora ディストリビューション
- Xillinux (Zynq および Cyclone V SoC プラットフォームのみ)

driver が現在のシステム設定にインストールされているかどうかを簡単に確認するには、 shell promptで次のように入力します。

```
$ modinfo xillybus_core
```

driver がインストールされている場合は、それに関する情報が出力されます。それ以外の場合は、 “modinfo: ERROR: Module xillybus_core not found”と表示されます。

同様に、 XillyUSBの driverを確認するには、次のコマンドを実行します。

```
$ modinfo xillyusb
```

XillyUSB は、 Ubuntu 22.04、 Fedora 35 以降、および派生ディストリビューションをインストールする必要なく動作します。

Linux が仮想マシンで実行されている場合、 PCIe busでは Xillybus が検出されないことに注意してください。 driver を搭載したオペレーティング システムは、 bare metalで実行する必要があります。 XillyUSB は、仮想マシン内で動作する場合があります。

段落 2.7 では、 Xillybus device files パーミッションを永続的に変更する方法を概説しています。これにより、すべてのユーザーがアクセスできるようになります (root ユーザーのみではなく、これはデスクトップ コンピューターで必要になることがよくあります)。

2.2.3 Xillybusの driverを含むLinux kernels

Xillybusの driver (PCIe および AXI用) は、バージョン 3.12から始まる公式の Linux kernel に含まれています。 3.12 と 3.17の間のバージョンの kernels では、 driver は “staging driver”として含まれていました。これは、コミュニティが新しい driverを完全に受け入れる前の準備段階です。 Xillybusの driver は、バージョン 3.18で non-staging として認められました。いくつかのコーディング スタイルの変更にもかかわらず、 3.12 kernel の最も初期の driver と現在利用可能な driver の間に機能上の違いはほとんどありません。

staging driver がロードされると、 kernel はシステムのログに warning を発行し、品質が不明であることを示します。 Xillybusに関しては、この warning は無視しても問題ありません。

precompiled であり、ディストリビューションに含まれる kernels については、Xillybusの driver が kernel 構成で有効になっているディストリビューションもあれば、そうでないディストリビューションもあるため、 Xillybus は kernel images または kernel modulesに含まれない場合があります。

自分で kernel compilation を実行する場合は、 driver の別の compilation を kernel modules として実行するのではなく、 driverを含めるように構成することができます (段落 2.9を参照)。

しかし、 Xillybus をそのままサポートしないディストリビューションに固執したいユーザーにとっては、次に説明する kernel module オプションの方が通常は簡単です。

前述のとおり、 XillyUSB の driver は、 5.14のバージョンから Linux kernel に含まれています。

2.3 前提条件の確認

Linux のインストールでは、 kernel module compilation用の基本的なツールが不足している場合があります。このタスクは一般的であるため、各 Linux ディストリビューションに固有のハウツー ガイドが Web 上にあります。

これらのツールが存在するかどうかを知る最も簡単な方法は、それらを実行してみることです。たとえば、 command promptでは “make coffee” と入力します。これは正しい応答です。

```
$ make coffee

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

これはエラーですが、 “make” ユーティリティが存在することがわかります。 GNU make をインストールする必要がある場合は、次のようになります。

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

C compiler も必要です。 “gcc”と入力し、目的の応答を入力します。

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

再びエラー メッセージが表示されますが、 “command not found”ではありません。

これら 2 つのツールに加えて、 kernel headers もインストールする必要があります。これはチェックするのが少し難しいです。これらのファイルが見つからないことを知る一般的な方法は、 header file が見つからないというエラーで kernel compilation が失敗した場合です。

Fedora、 RHEL、 CentOS 、およびその他の Red Hat の派生物では、この種のコマンドでコンピューターの準備が整う可能性があります。

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

Ubuntu およびその他の Debianベースのディストリビューションの場合:

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

重要:

これらのインストール コマンドは、 *root*として発行する必要があります。 *root* ユーザーであるというこの概念に慣れていない人は、まず付録のパラグラフ *A.4*でそれについて学ぶことをお勧めします。

.

2.4 ダウンロードしたファイルの解凍

Xillybusのサイトから driver をダウンロードした後、ダウンロードしたファイルがある場所にディレクトリを変更します。 command promptで、次のように入力します ($ 記号を除く)。

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

XillyUSB driverの場合:

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

新しい command promptだけで、応答はありません。

2.5 kernel moduleの compilation を実行する

モジュールのソースがある場所にディレクトリを変更します。 XillyUSB driverの場合:

```
$ cd xillyusb/driver
```

Xillybus driverの場合:

```
$ cd xillybus/module
```

“make” と入力して、モジュールの compilation を実行します。セッションは次のようになります。

```
$ 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'
```

詳細は多少異なる場合がありますが、エラーや warnings は表示されません。XillyUSBの場合、単一のモジュール xillyusb.koのみが生成されます。

kernel modules の compilation は、 compilation中に実行されている kernel に固有であることに注意してください。

別の kernel を使用する場合は、 “make TARGET=kernel-version”と入力します。“kernel-version” は、 /lib/modules/に表示される、選択した kernel バージョンです。

2.6 kernel モジュールの取り付け

同じディレクトリにとどまり、 root になり、 “make install”と入力します。これには数秒かかる場合がありますが、エラーは発生しません。これが失敗した場合は、compilation によって生成された *.ko ファイルを既存の kernel driver サブディレクトリにコピーし、次のように depmod を実行します。この例は、関連する kernel が 3.10.0であると仮定して、 PCIe driverのファイルをコピーする方法を示しています。

```
# 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
```

インストールは、モジュールを kernel にすぐにはロードしません。これは、Xillybus 周辺機器が検出された場合、システムの次の boot で行われます。モジュールを手動でロードする方法は、段落 2.8に示されています。

XillyUSBの場合、 reboot は必要ありません。モジュールは、次に USB デバイスが接続されるか、別の方法で検出されたときに自動的にロードされます。

2.7 udev rule ファイルのコピー

デフォルトでは、 Xillybus device files はその所有者である rootのみがアクセスできます。 root としての作業を回避できるように、これらのファイルにすべてのユーザーがアクセスできるようにすることは非常に理にかなっています。 udev メカニズムは、一連のルールに従って device files が生成されると、これらのファイルのアクセス許可を変更します。

したがって、 rootと同じディレクトリに、 udev rule ファイルをすべてのルールが保存されている場所 (おそらく /etc/udev/rules.d/) にコピーします。

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

ファイルの内容は次のとおりです。

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

これは、 Xillybus device driver によって生成されたすべてのファイルに permission mode 0666を指定する必要があることを意味します。つまり、読み取りと書き込みは誰にでも許可されます。

udev ファイルの内容を変更することで、所有者のみのアクセス許可を保持し、代わりに所有者の ID を変更することができることに注意してください。

XillyUSBの場合、ファイルは 10-xillyusb.rulesであり、

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

2.8 モジュールのロードとアンロード

モジュールをロードする (そして Xillybusで作業を開始する) には、次のようにrootと入力します。

```
# modprobe xillybus_pcie
```

または、 XillyUSBの場合:

```
# modprobe xillyusb
```

これにより、 Xillybus device files が表示されます (対応するハードウェアが busに存在すると仮定します)。

システムが boot プロセスを実行したときに Xillybus PCIe / AXI 周辺機器が存在し、上記のように driver がすでにインストールされている場合、これは必要ないことに注意してください。または、 driver が既にインストールされているときに、XillyUSB デバイスがコンピューターに接続されました。

kernelのモジュールのリストを表示するには、 “lsmod”と入力します。 kernelからモジュールを取り外すには、(PCIe の場合) と入力します。

```
# rmmod xillybus_pcie xillybus_core
```

これにより、 device files が消えます。

何か問題が発生したと思われる場合は、 /var/log/syslog ログ ファイルを調べて、“xillybus” または “xillyusb”という単語を含むメッセージを確認してください。多くの場合、このログ ファイルには貴重な手がかりが見つかります。 “dmesg” コマンドでも同じログ情報にアクセスできます。

/var/log/syslog ログ ファイルが存在しない場合は、代わりにおそらく /var/log/messages です。

2.9 公式の Linux kernelのXillybus PCIe driver

前述のとおり、 Xillybus 用の driver は Linux kernel v3.12.0 以降に含まれています。したがって、 driver を個別にインストールする代わりに、 Xillybusをサポートするようにパラメーターを設定して、 kernel 全体の compilation を実行することができます。

driver が含まれている kernel compilation は、段落 2.3 から 2.6で説明されている手順と機能的に同等です。

compilation用の kernel に Xillybusの driver を含めるには、 kernel modules または kernel 自体の一部として、そのオプション パラメータを有効にする必要があります。たとえば、 Xillybusの driver を PCIe インターフェイス用に有効にする .config の部分は、 kernel module compilationでは次のようになります。

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

同様に、 XillyUSB (kernel v5.14 以降) の場合:

```
CONFIG_XILLYUSB=m
```

このファイルを変更する一般的な方法は、 kernelのビルド環境ツール (通常は “make config”、 “make xconfig” 、または “make gconfig”) を使用することです。後者の 2 つは、文字列 “xillybus” を検索し、チェックボックスをクリックしてパラメーターを有効にする GUI ツールです。上記の構成パラメーターのテキスト表現は、正しいオプションが設定されていることを確認するのに役立ちます。

3.18より前の kernels では、 Xillybusを有効にする前に staging drivers を有効にする必要がある場合があり、その結果、 .config ファイルに次の行が表示されます。

```
CONFIG_STAGING=y
```

.config ファイルで Xillybusの driver が有効になったら、一般的な方法に従って kernel compilation を実行します。

kernel 5.14以降、 Xillybus および/または XillyUSB をアクティブ化すると、 CONFIG_XILLYBUS_CLASSが自動的に有効になります。これは、構成システムの依存関係ルールの結果です。したがって、このフラグを手動で設定する必要はありません (kernelを構成するための GUI ユーティリティからは不可能です)。

3

“Hello, world” テスト

3.1 目標

Xillybus は開発キットです。そのため、そのメリットは、実際の designで使用することによって最もよくわかります。したがって、最初の FPGA コードには、作業の基礎として、可能な限り単純な実装が含まれています。 coreとのインターフェイスの 2 つのペアの間に接続された 2 つの FIFOs があります。 core に送信されるすべてのデータは、これらの FIFOsに保存され、 hostにループバックされます。 RAM も coreに接続され、メモリまたは registersへのアクセスを示します。

以下に示すテストは、 FPGAとの通信をトリガーするだけです。システムの仕組みをよりよく理解するために、 FPGA にわずかな変更を加え、カスタム logic を FIFOsに取り付けることをお勧めします。

3.2 前提条件

- demo bundle から取得した bitstream は、 FPGAにロードされます。これは、Getting started with the FPGA demo bundle for Xilinx または Getting started with the FPGA demo bundle for Intel FPGAで説明されています。

  Zynq および Cyclone V SoC プラットフォームでは、 demo bundle が Xillinuxの一部であるため、これはデフォルトで当てはまります。 Getting started with Xillinux for Zynq-7000 または Getting started with Xillinux for Cyclone V SoC (SoCKit)を参照してください。

- Xillinux プラットフォームを除く: コンピューターは boot を適切に実行し、 PCIe / USB インターフェイスが検出されました (これは “lspci” または “lsusb”で確認できます)。

- あなたは UNIX / Linux command-line インターフェイスにかなり慣れています。これについては、付録 A が少し役立つかもしれません。

3.3 簡単な loopback テスト

このテストの目的は、 loopback が実際に機能することを確認することです。最も簡単な方法は、 UNIX コマンドライン ユーティリティである “cat” を使用することです。

“Terminal” デスクトップ項目をダブルクリックするなどして 2 つの terminal ウィンドウを開くか、デスクトップにない場合はメニューで探します。

command prompt での最初の terminal ウィンドウ タイプ (ドル記号を入力しないでください。それは promptです):

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

これにより、 “cat” プログラムは、 xillybus_read_8 device fileから読み取ったものをすべて出力します。この段階では何も起こらないと予想されます。

XillyUSB を使用している場合は、代わりに *device file* が *xillyusb_00_read_8* として表示されます。 *“xillyusb”* プレフィックスは明らかであり、 *“00”* インデックスは、複数の *USB* デバイスを同じ *host*に接続できるようにすることを目的としています。以下では、 *device files* の *PCIe / AXI* の命名規則が使用されます。

2 番目の terminal ウィンドウで、次のように入力します。

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

> リダイレクト文字に注意してください。これは、 “cat” に、入力されたものをすべて xillybus_write_8に送信するように指示します。

次に、2 番目の terminalにテキストを入力し、 ENTERを押します。同じテキストが最初の terminalに表示されます。一般的な UNIX の慣習により、 ENTER が押されるまで xillybus_write_8 には何も送信されません。

これら 2 つの “cat” コマンドの試行中にエラー メッセージが表示された場合は、入力ミスがないか確認してください。ただし、エラーが許可が拒否された場合を除きます。後者の場合、段落 2.7 で説明されている手順に従い、 driver モジュールをリロードする (またはコンピューターで reboot を実行する) ことをお勧めします。

または、 root ユーザーとして Xillybus device files と対話することもできます。これは、デスクトップ コンピューターではあまり推奨されません (ただし、 embedded

プラットフォームでは一般的です)。詳細については、付録の段落 A.4を参照してください。

それ以外の場合は、 “xillybus”という単語を含むメッセージの “dmesg” コマンドの出力を確認し、これらのいずれかが問題を示しているかどうかを調べます。または、 /var/log/syslog (ディストリビューションが Red Hat、 Fedora、 CentOS 、および同様のディストリビューションのいずれかである場合は、 /var/log/messages) からメッセージを取得します。

どちらの “cat” も CTRL-Cで停止できます。他の簡単なファイル操作も同様に機能します。たとえば、最初の terminal がまだ読み取っている状態で、2 番目の terminalに次のように入力します。

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

これは、 FIFO の両端が Xillybusの IP coreに接続されている限り機能します。もちろん、 FPGA コードを変更すると、上記の操作の結果も変わります。

FPGA 内の FIFOs は、 overflow または underflowの危険にさらされていないことに注意してください。 core は、 FPGA内の ’full’ および ’empty’ 信号を尊重します。必要に応じて、 Xillybus driver は、 FIFO が I/O の準備が整うまでアプリケーションを強制的に待機させます (blocking = により user space プログラムを強制的にスリープ状態にします)。

/dev/xillybus_read_32 と /dev/xillybus_write_32の間に FIFO を挟んだ device files の別のペアがあります。これらの device files は 32 ビット ワードの粒度で動作し、FPGAの FIFO も同様です。上記と同じテストを試みると、同様の動作になりますが、1 つの違いがあります。すべての I/O は 4 バイトのチャンクで実行されるため、入力が 4 バイトのチャンクの境界に達していない場合、最後のバイトは送信されないままになります。

4

サンプル host アプリケーション

4.1 全般的

Xillybus / XillyUSB用の host driver を含むバンドルには、4 つまたは 5 つの単純な C プログラムがあります。 demoapps ディレクトリは、次のファイルで構成されています。

- Makefile – このファイルには、 “make” ユーティリティがプログラムの compilationに対して使用するルールが含まれています。
- streamread.c – ファイルから読み取り、データを standard outputに送信します。
- streamwrite.c – standard inputからデータを読み取り、ファイルに送信します。
- memread.c – seekを実行した後にデータを読み取ります。 FPGAでメモリ インターフェイスにアクセスする方法を示します。
- memwrite.c – seekを実行した後にデータを書き込みます。また、 FPGAでメモリ インターフェイスにアクセスする方法も示します。

これらのプログラムの目的は、正しいコーディング スタイルを示し、カスタム アプリケーションを作成するための基礎として機能することです。ただし、これらはどちらも実際のアプリケーションでの使用を意図したものではありません。特に、セクション 5で説明されているように、高帯域幅のデータ転送には適していないためです。

これらのプログラムは非常に単純であり、 Xillybus host application programming guide for Linuxで詳細に説明されている UNIXの一般的なファイル アクセス方法にすべて準拠しているため、ここでは個別に説明しません。

これらのプログラムは、 fopen()、 fread()、 fwrite() セットではなく、単純な関数 open()、 read()、 write() などを使用することに注意してください。後者は、 C ライブラリ レベルで data buffers が原因で予期しない動作を引き起こす可能性があるためです。

5 番目のプログラム fifo.cは、 PCIe用の driver とのバンドルにのみ存在します。これは、連続 data streaming用の userspace RAM FIFO の実装を示しています。device fileの RAM buffers はほとんどすべてのシナリオに十分なスペースを生成するように構成できるため、このプログラムが役立つことはめったにありません。これらのシナリオは、 XillyUSBが提供する帯域幅が比較的低いため、 XillyUSBでは達成できません。したがって、 fifo.c はその driverとともに含まれていません。

4.2 編集と compilation

Linux 環境のプログラムの compilation に精通している人は、次の段落まで飛ばしてもかまいません。プログラムの設定に異常はありません。

何よりもまず、ディレクトリを C ファイルがある場所に変更します。

```
$ cd demoapps
```

5 つのプログラムすべての compilation を実行するには、 shell promptで “make” と入力します。次のセッションが予定されています。

```
$ 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’ で始まる 5 行は、 “make” ユーティリティによって生成された compilerの呼び出しです。これらのコマンドは、任意のプログラムの compilation に対して個別に使用できます (ただし、そうする理由はありません。単に “make”を使用してください)。

一部のシステムでは、 POSIX threads のライブラリがインストールされていない場合 (たとえば、一部の Cygwin インストール)、5 番目の compilation (fifo.cの) が失敗することがあります。 fifo.c が目的のプログラムでない限り、これは無視できます。

“make” ユーティリティは、必要なものだけで compilation を実行します。ソースファイルの 1 つだけが変更された場合、そのファイルのみが、その後の “make”の呼び出しで compilation を受けます。したがって、通常の手順では、対象のソースファイルを編集してから、必要に応じて recompilation に “make” を使用します。

compilationによって生成された executables を削除するには、 “make clean”と入力します。

前述のように、 compilation ルールは Makefileにあります。その構文はやや難しいと思われるかもしれませんが、幸いなことに、正確に理解していなくても編集できるファイルの 1 つです。

指定された Makefile は、現在のディレクトリ内のファイルのみに関連しています。つまり、ディレクトリ全体のコピーを作成し、衝突することなくコピーを操作できるということです。 C ファイルを追加して、他のソース ファイルに加えて “make” が compilation を実行するようにルールを簡単に変更することもできます。

たとえば、 memwrite.c が新しいファイル mygames.cにコピーされたとします。これは、 GUI インターフェイスまたは command lineで実行できます。

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

Makefileの編集に入ります。多数のエディターがあり、それぞれを呼び出す方法も多数あります。手っ取り早く始めるには、 gedit または xedを使用します。これらは、デスクトップ上のアイコンで呼び出すか、 shell promptで直接呼び出すことができます。 Makefileを編集するには、

```
$ gedit Makefile &
```

コマンドの末尾にある「&」は、アプリケーションが別の shell promptを取得するまで待機しないことを意味します。これは、特に GUI アプリケーションの起動に適しています。

Makefileには、次の行があります。

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

それは実際には名前の間にスペースを入れた名前のリストです。 mygames (.c サフィックスなし) をリストに追加します。

4.3 実行

3.3 の段落に示されている単純な loopback の例は、2 つのアプリケーションで実行できます。 compilationの後に、最初の terminalを入力します。

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

これは、 device fileから読み取るプログラムです。 executableを選択するときは、 "./" で明示的に現在のディレクトリを指す必要があることに注意してください。

次に、2 番目のウィンドウで (ディレクトリの変更が必要であると仮定します):

```
$ cd demoapps
$ ./streamwrite /dev/xillybus_write_8
```

これは多かれ少なかれ "cat"と同じように機能しますが、 "streamwrite" は文字単位で機能しようとし、 ENTERを待たない点だけが異なります。これは config_console()と呼ばれる関数で行われますが、これは無視できます。入力の即時効果のためだけに存在します。

上記の例は、 PCIe / AXIの Xillybus に関するものです。 XillyUSBでは、 device file の名前の接頭辞が若干異なるため、たとえば xillybus_read_8ではなく xillyusb_00_read_8 になります。

重要:

streamread と *streamwrite* は、実装をシンプルに保つために、 *I/O* を *128* バイトのチャンクで実行します。データ レートが重要な場合は、より大きな *buffers* を使用する必要があります。段落 *5.3*を参照してください。

4.4 メモリーインターフェース

memread および memwrite プログラムは、 device fileで lseek() 関数呼び出しを行うことによって FPGA のメモリにアクセスする方法を示しているため、より興味深いものです。 demo bundleでは、 xillybus_mem_8 のみが seekingを許可することに注意してください。また、読み取りと書き込みの両方で開くことができる唯一の device file です。

メモリに書き込む前に、 hexdump ユーティリティを使用して現在の状況を観察します。

```
$ 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
```

出力は異なる場合があります: これは FPGAの RAM を反映しており、最初は他の値が含まれている可能性があります (以前の実験による可能性が最も高い)。

-C および -v フラグは、 hexdump に示されている出力形式を使用するように指示します。 “-n 32” 部分は、最初の 32 バイトのみを要求します。メモリ配列の長さはわずか 32 バイトなので、それ以上読む必要はありません。

何が起こったかについての簡単な説明: hexdump は /dev/xillybus_mem_8 を開き、そこから 32 バイトを読み取りました。 lseek() 関数呼び出しを許可するすべてのファイルは、開かれたときにデフォルトで位置 0 から始まるため、出力はメモリ配列の最初の 32 バイトになります。

アドレス 3 のメモリの値を 170 (hex 形式の0xaa) に変更します。

```
$ ./memwrite /dev/xillybus_mem_8 3 170
```

そして、配列全体を再度読み取ります。

```
$ 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
```

明らかに、うまくいきました。

memwrite.c の特別な部分は、 “lseek(fd, address, SEEK_SET)”と書かれているところです。このコマンドを使用すると、 FPGAのアドレスが設定され、後続の読み取りまたは書き込みがその位置から開始されます (データが流れるにつれてインクリメントされます)。

5

高帯域幅パフォーマンスのガイドライン

Xillybusの IP cores のユーザーは、宣伝されているデータ レートが実際に満たされていることを確認するために、データ帯域幅テストを実行することがよくあります。これらのレートを達成するには、データ フローを大幅に遅くする可能性のある障害を回避する必要があります。そのほとんどは、 host がデータを十分に迅速に処理していないことが原因です。

IP coreの機能を最大限に活用するには、アプリケーション プロジェクトでこれらの障害を回避することがさらに重要です。

このセクションはガイドラインの集まりであり、最も一般的な間違いに基づいています。これらのガイドラインに従うと、帯域幅の測定値が公開されているものと同じか、わずかに高くなるはずです。

予想される帯域幅を満たしていないという苦情の最も一般的な理由は、測定が正しくないことです。推奨される方法は、以下の 5.3 の段落に示すように、 Linuxの “dd” コマンドを使用することです。

このセクションの情報は、 “Getting Started” ガイドとしては比較的高度なものであり、他のドキュメントで説明されているトピックを参照しています。それにもかかわらず、多くのユーザーが IP coreに慣れる初期段階でパフォーマンス テストを実行するため、このガイドではそれを示しています。

5.1 loopbackをしないでください

demo bundle の logic は、反対方向に進む streams のカップル間のデータの loopback を実装します。これは初期テストには役立ちますが、パフォーマンスのテストには適していません。問題は、 Xillybus IP coreによって作成されるデータ転送バーストが、 loopback FIFO を完全に満たすか、完全に空にすることです。これが発生

するたびに、 IP coreのデータ転送が一時的に停止します。これらの一時停止により、測定されたスループットが大幅に低下します。

FPGA から hostへの最大データ レートを使用する実際のシナリオでは、 application logic は、 Xillybus IP core がそれを排出するのと同じ速さで FPGAの FIFO を満たします。したがって、 FIFO が空になることは決してありません。同様に、他の方向では、 application logic は IP core がいっぱいになるのと同じ速さで FIFO を空にするため、いっぱいになることはありません。

もちろん、機能の観点からは、 FIFO が最初のケースで空になり、2 番目のケースでいっぱいになっても問題ありません。これにより、 Xillybus stream が一時的に停止するだけです。すべてが正しく機能しますが、最大レートではありません。

demo bundleに基づいたパフォーマンス テストが必要な場合は、非常に簡単に変更できます。たとえば、32 ビット インターフェイスをテストするには、user_r_read_32_empty および user_w_write_32_full 信号を FIFO (fifo_32) から切断し、それらを定数ゼロに結び付けます。これにより、 FIFO が overflow と underflow の両方の状態に陥り、誤ったデータが発生する可能性がありますが、データ レートは最適になります。このようなパフォーマンス テストでアプリケーション データを使用する必要がある場合は、これら 2 つの信号が決して High にならないようにする任意の配置で問題ありません。

loopback を開くと、各方向を個別にテストできますが、同時に両方の方向をテストする正しい方法でもあることに注意してください。

5.2 ディスクやその他のストレージを含めないでください

ディスク、 solid-state drives 、およびその他の種類のコンピューター ストレージは、多くの場合、帯域幅の期待が満たされない理由です。オペレーティング システムの caching メカニズムは、基盤となる物理メディアを必要としない高速データ転送の短距離バーストを可能にするため、混乱を助長します。 cache は最新のコンピューターでは非常に大きくなる可能性があるため、ディスクの速度制限が明らかになる前に、いくつかの Gigabytes データが流れる可能性があります。これは、データ レートのこの突然の変化には他に明確な説明がないため、 Xillybusのデータ トランスポートで何か問題が発生したとユーザーに思わせる可能性があります。

solid-state drives (flash) では、特に長時間連続して書き込むときに、さらに混乱を招く要因があります。 flash drive への書き込みには、未使用の blocksの消去が含まれます。これは、 flash memory へのデータの書き込みは、消去された、つまり空の blocks に対してのみ許可されるためです。

flash drive には通常、すでに消去されている blocks のプールがあります。これによ

り、データを書き込むための空き領域があるため、書き込み操作が高速になります。ただし、空の blocksがなくなると、 flash drive が強制的に blocks を消去し、場合によってはそのデータを再編成するため、大幅な速度低下が発生します。

これらの理由から、 Xillybusの帯域幅のテストでは、メディアが十分に高速であるように見える場合でも、ストレージ メディアを使用しないでください。よくある間違いは、 Xillybus stream からディスク上の大きなファイルにデータをコピーしようとして、かかる時間を測定することです。この操作は機能的には正しくても、パフォーマンス測定は完全に間違っている可能性があります。

ストレージがアプリケーションの一部であることが意図されている場合 (例: data acquisition)、メディアで広範囲にわたる長期テストを実行して、期待どおりであることを確認することをお勧めします。短い benchmark test は非常に誤解を招く可能性があります。

5.3 大きなチャンクの読み取りと書き込み

各 read() および write() 操作は、オペレーティング システム上で system call を生成し、 CPU サイクルで犠牲になります。したがって、十分な大きさの buffer サイズを使用することが重要です。これは、帯域幅テストだけでなく、高性能アプリケーションにも当てはまります。

buffer の一般的な適切なサイズは、 read() および write() 関数呼び出しごとに 128 kB 前後です。そのような関数呼び出しのそれぞれが 128 kBを処理することを必ずしも意味するわけではありませんが、これらの関数呼び出しが最大で 128 kBを処理できることを意味します。

streamread と streamwrite のサンプル ユーティリティ (4を参照) はパフォーマンスの測定には適していないことに注意してください。これらの buffer のサイズは 128 バイト (kBではない) です。これにより、コード例が簡素化されますが、パフォーマンス テストには遅すぎます。

次の shell コマンドを使用すると、速度をすばやく確認できます (必要に応じて /dev/xillybus_* names を置き換えます)。

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

これらは、 CTRL-Cで停止するまで実行されます。 “count=” パラメータを追加して、一定量のデータのテストを実行します。

5.4 CPU の消費量に注意してください

速度に関して CPUの機能を過大評価するのはよくある間違いです。一般に信じられていることとは異なり、利用可能な最速の CPUs でさえ、 100-200 MB/s よりも高速なデータで意味のあることを行うのに苦労しています (threadによる)。コンピューター プログラムは、集中的なアプリケーションのボトルネックになることが多く、必ずしもデータ トランスポートとは限りません。 buffer のサイズが不十分な場合 (前述のように)、 CPU の消費量が過剰になることもあります。

したがって、 “top”などを使用して、 CPU の消費量に注意することが重要です。それでもなお、これらのプログラムの出力を適切に解釈することが重要です。特に複数のcore マシンではそうです。たとえば、 quad-core コンピュータ上の 25% CPU は、 CPU の消費量が少ないことを意味しますか?それとも、特定の thread上の 100% ですか? “top” を使用する場合は、プログラムのバージョンによって異なります。

もう 1 つの注意点は、 system callsの処理時間の測定方法と表示方法です。オペレーティング システムの overhead が処理速度を低下させる場合、それは特定のプロセスの CPU パーセンテージに表示されますか?

“time” ユーティリティを使用するのが簡単な方法です。例えば、

```
$ 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
```

下部の “time” の出力は、この操作にかかった 1.080 秒のうち、 processor が user space プログラムで 5 ms を費やし、1.074 秒がシステム コールの処理に費やされたことを示しています。これをまとめると、 processor が常にビジーであったことは明らかであり、 processor がボトルネックでした。この例では実際の I/O が実行されていないため、これは当然のことです。

5.5 読み取りと書き込みを相互に依存させない

双方向の通信を必要とするアプリケーションの場合、よくある間違いは、1 つのメイン ループで構成される single-threaded コンピュータ プログラムを作成する

ことです。ループごとに、データのチャンクが FPGAに書き込まれ、チャンクが FPGAから読み取られます。

2 つの streams が機能的に独立している場合、これで問題ない可能性があります。ただし、このようなプログラムは coprocessing アプリケーション用に作成されることが非常に多く、プログラムは処理のためにチャンクを送信してから結果を読み戻す必要があるという誤解に基づいているため、各ループは一定量のデータの処理を完了します。

このメソッドは非効率的であるだけでなく、実行がスタックする可能性もあります (記述の仕方によっては)。 Xillybus host application programming guide for Linux のセクション 6.6 では、このトピックについて詳しく説明し、より適切なコーディング手法を提案しています。

5.6 hostの RAM bandwidth 制限を知る

これは主に embedded システムおよび/またはリビジョン XL / XXL IP coreを使用する場合に適用されます。各マザーボード (または embedded システム) は、 DDR RAM メモリ ユニットへの帯域幅が制限されています。非常に要求の厳しいアプリケーションでは、これがボトルネックになる可能性があります。

FPGA から user space プログラムに送られるデータの各チャンクには、2 つの RAM 操作が必要であることに注意してください。1 つ目は、データが FPGA から DMA bufferに書き込まれるときです。 2 つ目は、データが user space プログラムからアクセス可能な buffer にコピーされる場合です。同様の理由で、データが反対方向に進む場合も、2 つの RAM 操作が必要です。

DMA buffers と user space buffers の分離は、 read() と write() (または同様の関数呼び出し) を使用するすべての I/O のオペレーティング システム要件です。

したがって、リビジョン XL IP core が両方向で同時にテストされ、各方向で約 3.5 GB/s が期待される場合、 RAMからこの帯域幅の 4 倍、つまり 14 GB/sが必要になります。すべてのマザーボードがこの機能を備えているわけではありません。また、 host は RAM を他の目的にも同時に使用することに注意してください。

リビジョン XXLでは、同じ理由で、一方向の単純なテストでも RAMの帯域幅能力を超える可能性があります。

5.7 DMA buffers 十分な大きさ

これが問題になることはめったにありませんが、言及する価値があります。 DMA buffers 用に host に割り当てられた RAM スペースが小さすぎる場合、 host が data

stream を小さなチャンクに分割することを余儀なくされるため、データ転送が遅くなる可能性があります。 CPU サイクル。

すべての demo bundles には、パフォーマンス テストに十分な DMA メモリがあり、 IP Core Factory で生成された cores にも同じことが当てはまり、目的の “Expected BW” が実際の期待値に設定され、 “Autoset Internals” が有効になっています。 “Buffering” は 10 msに設定する必要がありますが、どのオプションでも問題ない可能性が高くなります。

一般的に言えば、要求されたレートで 10 ms のデータに対応する合計 RAM スペースを持つ 4 つの DMA buffersは、帯域幅テストの目的には十分です。

5.8 適切なデータ幅を使用する

明らかに、 stream の最大データ レートは、 streamのデータ幅と bus_clkの周波数によって制限されます。しかし、それに加えて、リビジョン Aの IP cores では 32 ビットの streams を使用する必要があります。これは、8 ビットおよび 16 ビットの streams が実際にペイロード データの送信に使用するよりも多くの PCIe 帯域幅を消費するためです。

リビジョン B 以降のIP cores では、 PCIe ブロックのデータパスよりも広いデータ幅を選択できます。たとえば、リビジョン Bの IP core をテストするには、 PCIe データパスが 64 ビット幅であるため、当然 64 ビット幅の streamが必要になります。 application logic の方がデータへのアクセスが高速になるため、幅の広い stream を選択すると、わずかに良い結果が得られる可能性があります。ただし、その差は無視できる可能性があります。

リビジョン XL のIP cores は、128 ビット data streams (または 256 ビット、大きな違いはないはず) でテストする必要があります。 XXLの場合、これらの IP cores は 256 ビット data streamsでテストする必要があります。

5.9 cache 同期による速度低下

x86から派生した processors (32 および 64 ビット) は coherent cacheを使用するため、これは embedded システムにのみ適用されます。この問題は、コヒーレント ポート (ACP) に接続されているため、 Zynq processor の AXI bus を使用する Xillybus (たとえば、 Xillinux) には適用されませんが、 Zynq processor が PCIe busで Xillybus を使用する場合には適用されます。

いくつかの embedded processorsでは、特に PCIe bus が ARM processorで使用されている場合、 DMA buffers にアクセスするときに cache を同期する必要があるた

め、トランスポートが大幅に遅くなります。これは、 Ethernet、 USB ポート、その他の PCIe デバイスなど、 processor上の DMAベースの I/O に適用されるため、Xillybus の問題ではありません。

cache による速度低下は、 system call 状態 (“time” ユーティリティの“sys” 行出力) での CPU の不当な消費によって検出されますが、(上記の 5.3 の段落で述べたように) 大規模な buffers を使用しています。これは、 processorの cache同期オペコードの呼び出しで CPU サイクルが浪費された結果です。

x86/x64 アーキテクチャには、同期を必要としない coherent cache があるため、この問題は存在しません。

5.10 パラメータのチューニング

ダウンロード可能な demo bundles の PCIe ブロックは、主流の x86ベースのprocessorで要求された帯域幅を処理するように調整されています。同様に、 IP Core Factory と “Autoset Internals” で生成される streams は、通常、パフォーマンスと FPGA リソース使用率の最適なバランスを提供し、各 streamに定義された帯域幅を確保します。

まれに、リビジョン B 以降の cores のみで、要求されたデータ レートを達成するために、 PCIe ブロックのパラメーターをさらに微調整する必要がある場合があります (特に host から FPGAへの streams で)。これについては、 The guide to defining a custom Xillybus IP coreのセクション 4.5 で説明されています。

ただし、この例外的なシナリオでも、変更されるのは Xillybus IP coreのパラメーターではなく、 PCIe ブロックのパラメーターであることに注意してください。 IP coreのパラメータを調整してデータ レートを改善しようとするのはよくある間違いですが、ほとんどの場合、問題は上記の問題のいずれかにあります。

6

トラブルシューティング

Xillybus / XillyUSB 用の drivers は、意味のある log messagesを生成するように設計されました。したがって、何か問題があるように見える場合は、 “dmesg” コマンドの出力で “xillybus” という単語を含むメッセージを探すことをお勧めします。同じ log messsages が /var/log/syslog (または一部のシステムでは /var/log/messages) にあります。

実際、すべてが正常に動作しているように見えても、 system log を監視することをお勧めします。

PCIe / AXI driver からのメッセージとその説明のリストは、次の場所にあります。

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

ただし、メッセージ テキスト自体で Google を使用すると、特定のメッセージを簡単に見つけることができます。

A

Linux command lineの短いサバイバル ガイド

command line インターフェースに慣れていない人は、 Linux マシンで何かを行うのが少し難しいと感じるかもしれません。基本的なコマンド インターフェイスは 30 年以上同じであるため、各コマンドの使用方法に関するオンライン チュートリアルが多数あります。この短いガイドは初心者にすぎません。

A.1 いくつかのキーストローク

これは、最も一般的に使用されるキーストロークの概要です。

- CTRL-C: 現在実行中のアプリケーションを停止します
- CTRL-D: このセッションを終了します (terminal ウィンドウを閉じます)
- CTRL-L: クリアスクリーン
- TAB : command promptで、現在書かれている項目のオートコンプリートを試みます。長いファイル名の場合に便利です。名前の先頭を入力してから、[TAB]を入力します。
- 上矢印と下矢印: コマンド promptで、コマンド履歴から前のコマンドを提案します。行ったばかりのことを繰り返すのに便利です。以前のコマンドの編集も可能なので、以前のコマンドとほぼ同じことをするのにも適しています。
- space: コンピュータが terminal pagerで何かを表示する場合、 [space] は“page down”を意味します。
- q: “Quit”。ページごとの表示では、 “q” を使用してこのモードを終了します。

A.2 助けを求める

すべてのフラグとオプションを本当に覚えている人はいません。さらにヘルプを得るには、2 つの一般的な方法があります。 1 つは “man” コマンドで、2 番目はヘルプ フラグです。

たとえば、 “ls” コマンド (現在のディレクトリ内のファイルを一覧表示) について詳しく知るには、次のようにします。

```
$ man ls
```

「$」記号はコマンド promptであることに注意してください。これは、コマンドの準備ができていることを示すためにコンピューターが出力するものです。ほとんどの場合、 prompt の方が長く、ユーザーと現在のディレクトリに関する情報が含まれています。

マニュアルページは terminal pagerで示されています。 [space]、矢印キー、 Page Up および Page Down を使用してナビゲートし、 ’q’ を使用して終了します。

コマンドの要約を短くするには、 `--help` フラグを指定してコマンドを実行します。一部のコマンドは、 `-h` または `-help` (ダッシュ 1 つ) に応答します。他の人は、構文が適切でないときはいつでもヘルプ情報を出力します。試行錯誤です。 ls コマンドの場合:

```
$ 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

...
```

（そしてそれは続く）

A.3 ファイルの表示と編集

ファイルが短いと予想される場合 (または terminal windowのスクロールバーを使用しても問題ない場合)、 console でその内容を表示するには、次のようにします。

```
$ cat filename
```

長いファイルには terminal pagerが必要です。

```
$ less filename
```

テキスト ファイルの編集に関しては、多くのエディターから選択できます。 emacs (および Xemacs) と viは、最も人気のある (ただし、必ずしも最も簡単に開始できるとは限りません)。 vi エディターは習得が非常に難しいことで知られていますが、その単純さからいつでも利用でき、いつでも機能します。

推奨される単純な GUI エディターは、 geditまたは xedのうち、利用可能な方です。デスクトップメニューまたは command lineから開始できます。

```
$ gedit filename &
```

末尾の「&」は、コマンドを “in the background”で実行する必要があることを意味します。または簡単に言えば、コマンドが完了する前に次の command prompt が表示されます。 GUI アプリケーションは、このように開始するのが最適です。

もちろん、これらの例の 'filename' もパスにすることができます。たとえば、システムのメイン log fileを表示するには:

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

ファイルの最後にジャンプするには、 “less” の実行中に shift-Gを押します。

一部のコンピューターでは、ログ ファイルにアクセスできるのは root ユーザーのみであることに注意してください。

A.4 root ユーザー

Linux を含むすべての UNIX マシンには、 ID 0 を持つユーザーと、 superuserとしても知られる “root”という名前のユーザーがいます。このユーザーの特別な点は、すべてが許可されていることです。ファイル、リソース、および特定の操作へのアクセスに関する一般的な制限は、ユーザーがどのユーザーであるかに基づいて適用されますが、 root ユーザーには課されません。

これは、マルチユーザー コンピューターのプライバシーの問題だけではありません。 shell promptで簡単なコマンドを使用して、ハードディスク内のすべてのデータを削除することもすべて許可されます。誤ってデータを削除したり、一般的にコンピュータを役に立たなくしたり、システムを攻撃に対して脆弱にしたりするいく

つかの方法が含まれています。 UNIX システムの基本的な前提は、 root にアクセスできる人は誰でも、自分が何をしているかを知っているということです。コンピューターは、 root に対して、よろしいですかという質問をしません。

root として動作することは、ソフトウェアのインストールを含むシステムのメンテナンスに必要です。物事を台無しにしないための鍵は、 ENTERを押す前に考え、コマンドが必要に応じて正確に入力されたことを確認することです。インストール手順に正確に従うことは、通常安全です。彼らが何をしているのかを正確に理解することなく、変更を加えないでください。 root 以外のユーザーとして同じコマンドを繰り返すことができる場合 (他のファイルが関係している可能性があります)、そのユーザーとして何が起こるかを試してみてください。

rootであることの危険性のため、コマンドを root として実行する一般的な方法は、sudo コマンドを使用することです。たとえば、メイン ログ ファイルを表示します。

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

特定のユーザーにこの特別なモードを許可するようにシステムを設定する必要があるため、これは常に機能するとは限りません。システムには、ユーザーのパスワードが必要です (root パスワードではありません)。

2 番目の方法は、 “su” と入力し、すべてのコマンドが rootとして与えられるセッションを開始することです。これは、 rootとしていくつかのタスクを実行する必要がある場合に便利ですが、 rootであることを忘れて、何も考えずに間違ったことを書く可能性が高くなることも意味します。 root セッションは短くしてください。

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

今回は、 root パスワードが必要です。

shell prompt の変更は、通常のユーザーから rootへの ID の変更を示します。不明な場合は、 “whoami” と入力して現在の user nameを入手してください。

一部のシステムでは、関連するユーザーに対して sudo が機能しますが、それでもrootとしてセッションを呼び出すことが望ましい場合があります。 “su” を使用できない場合 (主に root のパスワードがわからないため)、簡単な代替手段は次のとおりです。

```
$ sudo su
#
```

A.5 選択したコマンド

最後に、一般的に使用される UNIX コマンドをいくつか紹介します。

いくつかのファイル操作コマンド (これには GUI ツールを使用する方がよいでしょう):

- cp – 1 つまたは複数のファイルをコピーします。
- rm – 1 つまたは複数のファイルを削除します。
- mv – ファイルを移動します。
- rmdir – ディレクトリを削除します。

そして、一般的に知っておくことが推奨されるもの：

- ls – 現在のディレクトリ (指定されている場合は別のディレクトリ) 内のすべてのファイルを一覧表示します。 “ls -l” はそれらを属性とともにリストします。
- lspci – bus上のすべての PCI (および PCIe) デバイスを一覧表示します。 Xillybus が PCIe 周辺機器として検出されたかどうかを判断するのに役立ちます。 lspci -v、 lspci -vv 、 lspci -nもお試しください。
- lsusb – bus上のすべての USB デバイスを一覧表示します。 XillyUSB が周辺機器として検出されたかどうかを判断するのに役立ちます。 lsusb -v と lsusb -vvも試してください。
- cd – ディレクトリの変更
- pwd – 現在のディレクトリを表示
- cat – ファイルを standard outputに送信します。または、引数が指定されていない場合は standard input を使用します。このコマンドの本来の目的はファイルを連結することでしたが、最終的にはファイルとの間の単純な入力と出力のためのスイス ナイフになりました。
- man – 特定のコマンドで manual page を表示します。また、 “man -a” を試してください (1 つのコマンドに対して複数の manual page がある場合もあります)。
- less – terminal pager。 standard input からのファイルまたはデータをページごとに表示します。上記を参照。コマンドの長い出力を表示するためにも使用されます。例えば、

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

- head – ファイルの先頭を表示
- tail – ファイルの終わりを表示します。または、 -f フラグを使用することでさらに効果的です: 到着時に末尾 + 新しい行を表示します。たとえば、ログファイルに適しています (rootとして):

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

- diff – 2 つのテキスト ファイルを比較します。何も言わない場合、ファイルは同一です。
- cmp – 2 つのバイナリ ファイルを比較します。何も言わない場合、ファイルは同一です。
- hexdump – ファイルの内容をきれいな形式で表示します。フラグ -v および -C が推奨されます。
- df – マウントされたディスクと、それぞれにどれだけのスペースが残っているかを表示します。さらに良いことに、 “df -h”
- make – Makefileのルールに従って、プロジェクトのビルド (compilation の実行) を試みます。
- gcc – GNU C compiler。
- ps – 実行中のプロセスのリストを取得します。 “ps a”、 “ps au” 、および “ps aux” は、異なる量の情報を提供します。通常は多すぎます。

そして、いくつかの高度なコマンド:

- grep – ファイル内の pattern または standard inputを検索します。パターンは regular expressionですが、テキストだけの場合は文字列を検索します。たとえば、メイン ログ ファイルで単語 “xillybus”を case insensitive 文字列として検索し、次のページに出力ページを表示します。

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

- find – ファイルを検索します。引数の構文は複雑ですが、ファイルの名前、年齢、種類、または考えられるあらゆる条件に基づいてファイルを見つけることができます。 man pageを参照してください。