(機械で日本語に翻訳)

Getting started with Xillybus on a Linux host

Xillybus Ltd.

www.xillybus.com

Version 4.1

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 This document may also be slightly outdated in relation to the original.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

このガイドでは、Linux host 上で Xillybus / XillyUSB を実行する目的で driver をインストールする手順について説明します。 IP core の基本機能を試す方法も示されています。

話を簡単にするために、host は compilation を実行できる完全な機能を備えたコンピューターであると仮定します。 embedded platform の手順は似ていますが、いくつかの単純な違いがあります (特に、cross compilation が必要な場合があります)。

また、このガイドでは、Xillybus の demo bundle に基づく bitstream がすでに FPGA にロードされており、FPGA が host (PCI Express、AXI bus、または USB 3.x を必要に応じて) によって peripheral として認識されていることも前提としています。

この段階に到達するための手順は、次のドキュメントのいずれかに概説されています (選択した FPGA に応じて異なります)。

- 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。 これらの device files は、他のファイルと同様に、開かれ、読み書きされます。ただし、これらのファイルはプロセス間では pipes と同様に動作します。この動作も TCP/IP streams に似ています。 host 上で実行されるプログラムにとっての違いは、stream の反対側が (同じコンピュータ上またはネットワーク上の別のコンピュータ上にある) 別の process ではなく、FPGA 内の FIFO であることです。 TCP/IP stream と同様に、Xillybus stream は高速データ転送で効率的に動作するように設計されていま

すが、stream は少量のデータが時折送信される場合にも良好なパフォーマンスを発揮します。

host 上の 1 つの driver は、PCIe を介して host と通信するすべての Xillybus IP cores で使用されます。別の driver は AXI インターフェイス用に設計されています。 XillyUSB用に別のdriverもあります。

FPGA で別の IP core が使用されている場合、driver を変更する必要はありません。 streams とその属性は、driver が host のオペレーティング システムにロードされると、driver によって自動的に検出されます。それに応じて、device files が /dev/xillybus_*something* 形式のファイル名で作成されます。同様に、XillyUSB 用の driver は、/dev/xillyusb_00_*something* 形式で device files を作成します。これらのファイル名では、00 の部分が device のインデックスです。複数の XillyUSB device が同時にコンピュータに接続されている場合、この部分は 01、02 などに置き換えられます。

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。

- インストールが必要かどうかを確認しています。そうでない場合は、以下の他の手順をスキップしてください (おそらく最後の手順、つまり udev ファイルのコピーをスキップしないでください)。
- 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ことの確認)
- driver を kernel module として含むダウンロードしたファイルを解凍します。
- kernel module の compilation を実行する
- kernel module の取り付け
- udev ファイルをインストールすると、(root だけでなく) すべてのユーザーが Xillybus device files にアクセスできるようになります。

これらの手順は、command-line (“Terminal”) を使用して実行されます。 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浅い人にとって役立つかもしれません。

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

Linux kernels および Linux ディストリビューションの大部分は、何もしなくても Xillybus (PCIe または AXI の場合) をサポートします。これについては以下でさらに説明します。

とはいえ、Xillybus がすでにサポートされている場合でも、udev ファイルのインストールに関するセクション 2.7 を参照する価値はあります。

XillyUSB 用の driver は、バージョン 5.14 (2021 年 8 月リリース) 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 distributions

ほとんどの Linux ディストリビューションには、PCIe / AXI Xillybus driver がすでにインストールされています (“out of the box”)。例えば：

- 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 以降、Fedora 35 以降、およびこれら 2 つから派生したディストリビューションでインストールする必要なく動作します。

Linux が virtual machine 内で実行されている場合、PCIe bus 上の Xillybus は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 上で実行する必要があります。 XillyUSB は virtual machine 内で動作する場合があります。

セクション 2.7 では、Xillybus device files の permissions を永続的に変更する方法を示します。この変更により、これらのファイルはすべてのユーザー (root ユーザーだけでなく) がアクセスできるようになります。この変更は、デスクトップ コンピュータで Xillybus を使用する場合に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 以降の正式な 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 が “staging driver” として含まれていました。これは、Linux 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 では non-staging として認められました。 coding style に関連するいくつかの変更にもかかわらず、初期の driver (kernel のバージョン 3.12) と現在入手可能な driver との間に機能的な違いはほとんどありません。

staging driver がロードされると、kernel は system log で警告を発行します。この警告は、driver の品質が不明であることを示しています。 Xillybus に関しては、この警告は無視しても問題ありません。

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。

Linux distribution の一部である kernels について: Xillybus の drivers が kernel の source code の一部である場合でも、これらの drivers は、kernel がこれらの drivers を含むように構成されている場合にのみ compilation に含まれます。 Xillybus の drivers は、ほとんどの主流の Linux distributions には kernel modules として含まれていますが、各 distribution には、kernel に何を含めるかを選択するための独自の基準があります。したがって、distribution と一緒に出荷される kernel には、Xillybus が含まれない可能性があります。

このガイドでは、kernel modules の別個の compilation を利用した drivers のインストールに焦点を当てています。通常、これが最も簡単な方法です。ただし、いずれにしても kernel の compilation を実行する人は、代わりに Xillybus の drivers を含むように kernel を構成することを好むかもしれません。この方法については、セクション 2.9 で説明します。

2.3 前提条件の確認

Linux システムには、kernel module compilation の基本ツールが不足している場合があります。これらのツールが存在するかどうかを確認する最も簡単な方法は、それらのツール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。これが正しい応答です。

```
$ 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。ただし、GNU make が見つからず、インストールする必要がある場合、出力は次のようになります。

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

C compilerも必要です。 compiler がインストールされているかどうかを確認するには、「“gcc”」と入力します。

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

この応答は、“gcc” がインストールされていることを示します。またエラーメッセージが出ましたが、“command not found”では出ませんでした。

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。これを確認するのは少し難しいです。これらのファイルが見つからないかどうかを知る一般的な方法は、kernel compilation が失敗して、header file が見つからないというエラーが表示された場合です。

kernel module compilation は一般的なタスクであるため、compilation 用にシステムを準備する方法に関して各 Linux distribution に固有の情報がインターネット上に多数あります。

Fedora、RHEL、CentOS、および Red Hat のその他の派生製品では、この種のコマンドによりコンピュータが準備される可能性があります。

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

Ubuntu および Debian に基づくその他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* ユーザーという概念に慣れていない人は、まずそれについて学ぶことをお勧めします。付録の「*A.4*」セクションを参照してください。

2.4 ダウンロードしたファイルを解凍する

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

source code または kernel module がある場所にディレクトリを変更します。 Xillybus driverの場合:

```
$ cd xillybus/module
```

XillyUSB driver の場合:

```
$ cd xillyusb/driver
```

“make” と入力して、モジュールの compilation を実行します。トランスクリプト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、kernel の必要なバージョンの名前です。 /lib/modules/に登場する名前です。あるいは、“Makefile” という名前のファイル内の次の行を編集します。

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

KDIR の値を必要な kernel headers の path に変更します。

2.6 kernel module の取り付け

ディレクトリを変更せずに、ユーザーを root に変更します (例: “sudo su” の場合)。次に、次のコマンドを入力します。

```
# make install
```

このコマンドは完了するまでに数秒かかる場合がありますが、エラーは生成されません。

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 kernel modules の既存のサブディレクトリにコピーします。次に、depmod を実行します。次の例は、kernel の関連バージョンが 3.10.0 である場合に、PCIe driver でこれを行う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
```

インストールしても、モジュールは kernel にすぐにはロードされません。 Xillybus ペリフェラルが検出された場合、これはシステムの次の boot で実行されます。モジュールを手動でロードする方法は、セクション 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、次回 USB device がコンピュータに接続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 にはその所有者 (root) のみがアクセスできます。 root として動作すること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、特定のルールに従って、device files が生成されるときに file permissions を変更します。

この機能を有効にする方法: 同じディレクトリに残り、root ユーザーのままになります。 udev rule ファイルを、システム内のそのようなファイ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。

例えば：

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

この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は permission mode 0666 を与える必要があることを意味します。 つまり、誰でも読み書きが許可されています。

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

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

udev ファイルを変更すると、異なる結果が得られることに注意してください。たとえば、代わりに device files の所有者を変更して、特定のユーザーのみがこれらのファイルにアクセスできるようにすることができます。

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 (Xillybus device が bus で検出されたと仮定します)。

システムが boot プロセスを実行するときに Xillybus PCIe / AXI 周辺機器が検出され、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。 driver がすでにインストールされているときに XillyUSB device がコンピュータに接続されている場合も、これは必要ありません。

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 driver 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 ファイルを調べて、必要に応じて “xillybus” または “xillyusb” という単語を含むメッセージがないか確認してください。多くの場合、貴重な手がかりがこのログ ファイルに見つかります。同じログ情報には、“dmesg” コマンドでもアクセスできます。

/var/log/syslog ログ ファイルが存在しない場合は、おそらく /var/log/messages が代わりに使用されます。おそらくコマンド “journalctl -k” を試してください。

2.9 公式の Linux kernel の Xillybus drivers

前述したように、Xillybus 用の driver は、バージョン v3.12.0 以降、Linux kernel に含まれています。したがって、kernel 全体の compilation を実行して、この kernel が Xillybus をサポートすることが可能です。これは、上で示したように kernel modules を個別にインストールする代わりの方法です。

機能の観点から見ると、kernel compilation を使用する方法は、セクション 2.3 から 2.6 で説明されている手順と同じ結果をもたらします。

compilation を対象とした kernel に Xillybus の driver を含めるには、いくつかの kernel configuration options を有効にする必要があります。 driver を含める方法は 2 つあります。 kernel modules として、または kernel image の一部として。

たとえば、これは kernel の構成ファイル (.config) で Xillybus の driver を PCIe インターフェイスに対して有効にする部分です。

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

“m” は、driver が kernel module として含まれていることを意味します。 “y” とは、kernel image に driver が含まれることを意味します。

同様に、XillyUSB (kernel v5.14 以降) の場合:

```
CONFIG_XILLYUSB=m
```

.config に変更を加える一般的な方法は、kernel の構成ツールを使用することです。 “make config”、“make xconfig”、または “make gconfig”。

xconfig および gconfig は、Xillybus の drivers を見つけるために文字列 “xillybus” を検索できるため、より使いやすい GUI ツールです。 driver は、チェックボックスをクリックすると有効になります。 .config ファイルのテキスト表現は、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 より前のバージョンの 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ます。これにより、.config ファイルに次の行が作成されます。

```
CONFIG_STAGING=y
```

.config ファイルで Xillybus の driver を有効にした後、通常どおり kernel compilation を実行します。

kernel 5.14 以降、Xillybus または XillyUSB の driver が有効になると、CONFIG_XILLYBUS_CLASS という名前のオプション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、このオプションを手動で変更することは不要です (多くの場合、不可能です)。

3

“Hello, world” テスト

3.1 目標

Xillybus は、logic design のビルディング ブロックとして意図されたツールです。したがって、Xillybus の機能について学ぶ最良の方法は、Xillybus を自分の user application logic と統合することです。 demo bundle の目的は、Xillybus を使用するための出発点となることです。

したがって、可能な限り最も単純なアプリケーションが demo bundle に実装されています。 2 つの device files の間にある loopback。これは、FIFO の両側を FPGA の Xillybus IP Core に接続することで実現されます。その結果、host が 1 つの device file にデータを書き込むと、FPGA は別の device file を介して同じデータを host に返します。

以下のいくつかのセクションでは、この単純な機能をテストする方法について説明します。このテストは、Xillybus が正しく動作することを確認する簡単な方法です。 FPGA の IP Core は期待どおりに動作し、host は PCIe 周辺機器を正しく検出し、driver は正しくインストールされています。それに加えて、このテストは、FPGA で logic design に小さな変更を加えることで、Xillybus がどのように動作するかを学ぶ機会でもあります。

最初のステップとして、logic と FPGA および device files がどのように連携するかを理解するために、demo bundle で簡単な実験を行うことをお勧めします。これだけで、多くの場合、独自のアプリケーションのニーズに合わせて Xillybus を使用する方法が明確になります。

前述の loopback の他に、demo bundle には RAM と追加の loopback も実装されています。この追加の loopback については、以下で簡単に説明します。 RAM に関しては、メモリ アレイまたは registers にアクセスする方法を示します。詳細については、4.4 セクションを参照してください。

3.2 準備

“Hello world” テストを実行するには、いくつかの準備が必要です。

- 2 セクションで説明されているように、Xillybus の driver がコンピュータにインストールされています。
- FPGA には、demo bundle から作成された bitstream (変更なし) をロードする必要があります。これを実現する方法は、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。

  Xillinux (Zynq または Cyclone V SoC と組み合わせて) を使用する場合は、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。 demo bundle はデフォルトでこのシステムにすでに組み込まれています。
- PCIe のみに関連: コンピュータが boot を実行したときに、FPGA が PCIe bus 上で検出されました。これは、“lspci” コマンドを使用して確認できます。
- USB のみに関連: FPGA は USB port を介してコンピュータに接続されており、コンピュータは FPGA を USB device として検出しました。これは、“lsusb” コマンドを使用して確認できます。
- Linux command-line の使用には慣れているはずです。付録 A がこれに役立つかもしれません。

これらの準備が正しく行われていれば、Xillybus の device files が使用できるはずです。たとえば、/dev/xillybus_read_8 という名前のファイルが存在する必要があります。

3.3 簡単な loopback テスト

このテストを実行する最も簡単な方法は、“cat” という名前の Linux command-line utility を使用することです。

terminal windows を 2 つ開きます。一部のコンピュータでは、“Terminal” という名前のアイコンをダブルクリックすることでこれを行うことができます。そのようなアイコンがない場合は、デスクトップのメニューで検索してください。

最初の terminal window では、command prompt で次のコマンドを入力します (ドル記号は入力しないでください。 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 から読み込んだ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、*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 devices* 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。このガイドでは、*PCIe* と *AXI* の命名規則が使用されています。

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
```

> 文字に注目してください。これは、console で入力されたすべてのものを xillybus_write_8 (redirection) に送信するように “cat” 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。最初の terminal にも同じテキストが表示されます。 ENTER が押されるまで、xillybus_write_8 には何も送信されません。これは、Linux コンピュータの一般的な規則です。

これら 2 つの “cat” コマンドはどちらも CTRL-C で停止できます。

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、まず device files が作成されたこと (つまり、/dev/xillybus_read_8 と /dev/xillybus_write_8 が存在すること) を確認してください。また、タイプミスがないか確認してください。

エラーが “permission denied” の場合は、セクション 2.7 に示すように修正できます。ただし、udev ファイルは、kernel modules が kernel にロードされている場合にのみ有効になることに注意してください。 kernel modules をリロードする方法については、セクション 2.8 を参照してください。あるいは、コンピュータ上で reboot を実行します。

“permission denied” エラーを克服するもう 1 つの方法は、root ユーザーとして Xillybus device files を操作することです。これはデスクトップ コンピュータではあまり推奨されません (ただし、embedded platforms では一般的に行われます)。詳細については、付録セクション A.4 を参照してください。

その他のエラーについては、セクション 2.8 のガイドラインに従って、/var/log/syslog で詳細情報を検索するか、“dmesg” コマンド (または kernel log を取得するための同様の方法) を使用してください。

簡単なファイル操作を実行することも可能です。たとえば、最初の terminal で “cat” コマンドを停止せずに、2 番目の terminal で次のように入力します。

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

FPGA 内の FIFOs は、overflow または underflow に対して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 の準備ができるまでコンピュータ プログラムを強制的に待機させます。これは blocking と呼ばれ、user space program を強制的にスリープさせることを意味します。

loopback を間に挟んだ device files の別のペアがあります。/dev/xillybus_read_32と/dev/xillybus_write
これらの device files は 32 ビット ワードで動作し、これは FPGA 内の FIFO にも当てはまります。したがって、これらの device files を使用した “hello world” テストは同様の動作になりますが、次の 1 つの違いがあります。 すべての I/O は 4 バイトのグループで実行されます。したがって、入力が 4 バイトの境界に達していない場合、入力の最後のバイトは送信されないままになります。

4

host アプリケーションの例

4.1 全般的

Xillybus の device files にアクセスする方法を示す 4 つまたは 5 つの単純な C プログラムがあります。これらのプログラムは、host driver 用 Xillybus / XillyUSB を含む圧縮ファイル内にあります (Web サイトからダウンロードできます)。 “demoapps” ディレクトリを参照してください。この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、プログラムの compilation の目的で “make” ユーティリティによって使用され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実行後にデータを書き込みます。 FPGA のメモリ インターフェイスにアクセスする方法を示します。

これらのプログラムの目的は、正しい coding style を表示することです。独自のプログラムを作成するための基礎として使用することもできます。ただし、これらのプログラムはどちらも、特に高いデータ レートではパフォーマンスが良くないため、実際のアプリケーションで使用することを目的としていません。高帯域幅のパ

フォーマンスを実現するためのガイドラインについては、5 の章を参照してください。

これらのプログラムは非常に単純で、Linux コンピュータ上のファイルにアクセスするための標準的な方法を示しているだけです。これらの方法については、Xillybus host application programming guide for Linuxで詳しく説明されています。これらの理由から、ここではこれらのプログラムについての詳細な説明は行いません。

これらのプログラムは、open()、read()、write() などの低レベル API を使用することに注意してください。よりよく知られている API (fopen()、fread()、fwrite() など) は、C runtime library によって維持される data buffers に依存しているため、避けられます。これらの data buffers は、特に FPGA との通信が runtime library によって遅延することが多いため、混乱を引き起こす可能性があります。

driver を PCIe 用にダウンロードすると、“demoapps” ディレクトリに 5 番目のプログラムが見つかります。fifo.c。このプログラムは、userspace RAM FIFO の実装を示します。 device file の RAM buffers はほぼすべてのシナリオに十分に対応でき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ほとんどありません。したがって、fifo.c は、非常に高いデータ レートの場合、および RAM buffer が非常に大きい (つまり、複数の gigabytes) 必要がある場合にのみ役立ちます。

fifo.c を必要とするデータ レートは XillyUSB では不可能であるため、このプログラムは XillyUSB の driver には含まれていません。

4.2 編集と compilation

Linux の compilation プログラムの経験がある場合は、次のセクションに進んでください。 Xillybus のサンプルプログラムの compilation は、通常の方法で “make” で実行されます。

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” と入力するだけです。次のトランスクリプトが期待されます。

```
$ 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、compiler を使用するために “make” が要求するコマンドです。これらのコマンドは、プログラムの compilation に対して個別に使用できます。しかし、そうする理由はありません。 “make”を使用してください。

一部のシステムでは、POSIX threads library がインストールされていない場合 (Cygwin の一部のインストールなど)、5 番目の compilation (fifo.c の) が失敗する可能性があります。 fifo.c を使用するつもりがない場合、このエラーは無視してかまいません。

“make” ユーティリティは、必要な場合にのみ compilation を実行します。ファイルが 1 つだけ変更された場合、“make” はそのファイルのみの compilation を要求します。したがって、通常の作業方法は、編集したいファイルを編集してから、“make” を recompilation に使用することです。無駄なcompilationが発生しません。

以前の compilation によって生成された executables を削除するには、“make clean” を使用します。

上で述べたように、Makefile には compilation のルールが含まれています。このファイルの構文は単純ではありませんが、幸いなことに、常識的に考えてこのファイルを変更できることがよくあります。

Makefile は、Makefile 自体と同じディレクトリにあるファイルに関連します。したがって、ディレクトリ全体のコピーを作成し、このレプリカ内にあるファイルを操作することができます。ディレクトリの 2 つのコピーは相互に干渉しません。

C ファイルを追加し、Makefile を簡単に変更して、“make” もこの新しいファイルの compilation を実行することもできます。たとえば、memwrite.c が mygames.c という名前の新しいファイルにコピーされるとします。これは、GUI インターフェイスまたは command line を使用して実行できます。

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

次のステップは、Makefile を編集することです。多くのテキスト エディタがあり、それぞれを実行する方法も数多くあります。ほとんどのシステムでは、「“gedit”」または「“xed”」と入力することで、shell prompt から GUI editor を起動できます。ただし、コンピュータのデスクトップのメニューで GUI text editor を見つけるのは簡単です。 vim、emacs、nano、pico など、terminal window 内で動作するテキスト エディタも多数あります。

どの editor を使用するかは好みと個人的な経験の問題です。たとえば、次のコマンドを使用して Makefile の編集を開始できます。

```
$ xed Makefile &
```

コマンドの末尾の「&」は、プログラムが終了するまで待機しないように shell に指示します。 すぐに次のshell promptが登場します。 GUIアプリケーションなどの起動に適しています。

Makefile で変更する必要がある行は次のとおりです。

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

この行は次のように変更されます。

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

次回 “make” を入力すると、compilation または mygames.c が実行されます。

4.3 プログラムの実行

セクション 3.3 に示されている簡単な loopback の例は、2 つのサンプル プログラムを使用して実行できます。

“demoapps” はすでに current directory であり、compilation はすでに「make」で完了していると仮定します。

最初の terminal に次のように入力します。

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

device fileから読み込むプログラムです。

コマンドが “./” で始まることに注意してください。 executable のディレクトリを明示的に指定する必要があります。この例では、式 “./” を使用して current directory をリクエストします。

そして、2 番目の terminal window では次のようになります。

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

これは、“cat” の例とほぼ同様に機能します。違いは、“streamwrite” はデータを device file に送信する前に ENTER を待機しないことです。代わりに、このプログラムは各 character 上で個別に動作しようとします。これを実現するために、プログラムでは config_console() と呼ばれる関数を使用します。この機能は、キーボードの入力に対する即時応答を目的としてのみ使用されます。 Xillybusとは関係ありません。

上記の例は、Xillybus または PCIe / AXI に関連しています。 XillyUSB では、device files の名前には若干異なる接頭辞が付いています。たとえば、xillybus_read_8 ではなく xillyusb_00_read_8 です。

重要:

streamread および *streamwrite* によって実行される *I/O* 操作は非効率的です。これらのプログラムを簡素化するために、*I/O buffer* のサイズはわずか *128* バイトです。高いデータレートが必要な場合は、より大きな *buffers* を使用する必要があります。 *5.3* のセクション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、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。これは、device file 上で lseek() への関数呼び出しを行うことで実現されます。 Xillybus host application programming guide for Linux には、この API を Xillybus の device files と関連させて説明するセクションがあります。

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。この device file は、読み取りと書き込みの両方で開くことができる唯一のものでもあります。

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できます。

```
$ 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
```

この出力はメモリ配列の最初の 32 バイトです。hexdump は /dev/xillybus_mem_8 を開き、この device file から 32 バイトを読み取りました。 lseek() を許可するファイルを開くと、初期位置は常に 0 になります。したがって、出力はメモリ配列内の位置 0 から位置 31 までのデータで構成されます。

出力が異なる可能性があります。 この出力は FPGA の RAM を反映していますが、他の値が含まれている可能性があります。特に、RAM での以前の実験の結果、これらの値はゼロとは異なる場合があります。

hexdump の flags について一言: 上に示した出力の形式は、“-C” および “-v” の結果です。 “-n 32” は、最初の 32 バイトのみを表示することを意味します。メモリ配列

の長さはわずか 32 バイトなので、それ以上を読み取っても意味がありません。

memwrite を使用して配列内の値を変更できます。たとえば、次のコマンドを使用すると、アドレス 3 の値が 170 (hex 形式では 0xaa) に変更されます。

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

コマンドが機能したことを確認するには、上記の hexdump コマンドを繰り返すことができます。

```
$ 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、コマンドは機能しました。

memwrite.c で重要なのは “lseek(fd, address, SEEK_SET)” と書かれている部分です。この関数呼び出しは device file の位置を変更します。その結果、FPGA 内部でアクセスされる配列要素のアドレスが変更されます。後続の読み出し動作または書き込み動作はこの位置から開始されます。このようなアクセスのたびに、転送されたバイト数に応じて位置が増分されます。

seeking を可能にする device file は、FPGA に設定コマンドを簡単に送信するのにも役立ちます。すでに述べたように、lseek() を許可するファイルを開いた場合、初期位置は常に 0 です。これは、次の例のようなコマンドにも当てはまります。

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

“echo” コマンドの “-n” 部分に注目してください。これにより、“echo” が出力の最後に newline character を追加できなくなります。

このコマンドは、“0” の ASCII code (値 0x30) をアドレス 0 に書き込みます。同様に、値 0x31 はアドレス 1 に書き込まれます。したがって、この単純な “echo” コマンドを使用して、複数の registers の値を一度に設定できます。

FPGAでは実装が簡単なので便利な方法です。たとえば、文字 “0” と “1” のみが “echo” コマンドで使用されることを意図しているとします。したがって、bit 0のみが重要です。これは、“echo” コマンドの 3 バイト目から値を取得する register の例です。

```
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];
```

この方法は、FPGA や logic の開発中にテストを実行する場合に特に関連します。

5

高帯域幅パフォーマンスのガイドライン

Xillybus および IP cores のユーザーは、宣伝されているデータ転送速度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。これらの目標を達成するには、データ フローを大幅に遅くする可能性があるボトルネックを回避する必要があります。

このセクションは、最も一般的な間違いに基づいたガイドラインをまとめたものです。これらのガイドラインに従うと、公表されているものと同等かわずかに優れた帯域幅測定結果が得られます。

もちろん、Xillybus に基づくプロジェクトの実装では、このプロジェクトが IP core の機能を最大限に活用できるように、これらのガイドラインに従うことが重要です。

多くの場合、問題は host がデータを十分に速く処理しないことです。 公表されている数値を達成できないという苦情の最も一般的な理由は、データ レートの測定が間違っていることです。推奨される方法は、以下の 5.3 セクションに示すように、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です。この説明では、他のドキュメントで説明されている高度なトピックについても参照します。ただし、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、これらのガイドラインはこのガイドに記載されています。

5.1 loopbackをしないでください

demo bundle (FPGA の内側) では、2 つの streams ペアの間に loopback があります。これにより “Hello, world” テストが可能になります (3 のセクションを参照) が、テストのパフォーマンスには悪影響を及ぼします。

問題は、Xillybus IP core がデータ転送バーストで FPGA 内の FIFO を急速にいっぱいにしてしまうことです。この FIFO がいっぱいになるため、データの流れが一時的に停止します。

loopbackはこのFIFOで実装されているので、このFIFOの両側がIP coreに接続されています。 FIFO にデータが存在すると、IP core は FIFO からこのデータを取得し、host に送り返します。これもすぐに起こるので、FIFO は空になります。ここでも、データ フローが一時的に停止します。

データ フローが一時的に停止した結果、測定されたデータ転送速度は予想よりも低くなります。これは、FIFO が浅すぎることと、IP core が FIFO を埋めることと空にすることの両方を担当するために発生します。

実際のシナリオでは、loopback は存在しません。むしろ、FIFOの反対側にはapplication logicがあります。最大のデータ転送速度を達成する使用シナリオを考えてみましょう。 このシナリオでは、IP core がこの FIFO を埋めるのと同じくらい早く、application logic は FIFO からのデータを消費します。したがって、FIFO がいっぱいになることはありません。

反対方向についても同様に: application logic は、IP core がデータを消費するのと同じくらい早く FIFO をいっぱいにします。したがって、FIFO が空になることはありません。

機能的な観点から見ると、FIFO が時々満杯になったり空になったりすることは問題ありません。これにより、データ フローが一時的に停止するだけです。すべてが正しく動作しますが、最大速度では動作しません。

demo bundle は、パフォーマンス テストの目的で簡単に変更できます。 たとえば、/dev/xillybus_read_32 をテストするには、FPGA 内の user_r_read_32_empty を FIFO から切断します。代わりに、この signal を定数ゼロに接続します。その結果、IP core は FIFO が決して空ではないと認識します。したがって、データ転送は最大速度で実行されます。

これは、IP core が空の FIFO から読み取ることがあることを意味します。その結果、host に到着するデータは常に有効であるとは限りません (underflow のため)。しかし、スピードテストの場合、これは問題ではありません。データの内容が重要な場合、考えられる解決策は、application logic ができるだけ早く FIFO を埋めることです (たとえば、 counter の出力で)。

/dev/xillybus_write_32 をテストする場合も同様に: user_w_write_32_full を FIFO から切り離し、この signal を定数ゼロに接続します。 IP core は FIFO がフルになることはないと認識するため、データ転送は最大速度で実行されます。 FIFO に送信されるデータは、overflow により部分的に失われます。

loopback を切断すると、各方向を個別にテストできることに注意してください。ただし、これは両方向を同時にテストする正しい方法でもあります。

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

帯域幅の期待が満たされない原因は、ディスク、solid-state drives、その他の種類のコンピューター ストレージにあることがよくあります。ストレージメディアの速度を過大評価するのはよくある間違いです。

オペレーティング システムの cache メカニズムが混乱をさらに深めています。データがディスクに書き込まれるとき、必ずしも物理記憶媒体が関与するわけではありません。むしろ、データは RAM に書き込まれます。このデータがディスク自体に書き込まれるのは後になってからです。ディスクからの読み取り操作に物理メディアが関与しない可能性もあります。これは、同じデータが最近すでに読み取られている場合に発生します。

cache は、最新のコンピューターでは非常に大きくなる場合があります。したがって、ディスクの実際の速度制限が明らかになる前に、いくつかの Gigabytes のデータが流れる可能性があります。これにより、ユーザーは Xillybus のデータ転送に何か問題があるのではないかと考えるようになります。 データ転送速度のこの突然の変化については、他に説明がありません。

solid-state drives (flash) では、特に長時間の連続書き込み操作中に混乱の原因がさらに発生します。 flash drive の低レベル実装では、flash への書き込みの準備として、メモリの未使用セグメント (blocks) を消去する必要があります。これは、flash memory へのデータの書き込みは、消去された blocks に対してのみ許可されるためです。

出発点として、flash drive には通常、すでに消去された blocks が大量にあります。これにより、書き込み操作が高速になります。 データを書き込むためのスペースがたくさんあります。ただし、消去された blocks がなくなると、flash drive は blocks を消去し、場合によってはデータの defragmentation を実行することになります。これにより、明らかな説明のない大幅な速度低下が発生する可能性があります。

これらの理由から、Xillybus の帯域幅のテストにはストレージ メディアを使用しないでください。短期間のテストではストレージ メディアが十分に高速であるように見えても、誤解を招く可能性があります。

Xillybus device file からディスク上の大きなファイルにデータをコピーするのにかかる時間を測定してパフォーマンスを見積もるのは、よくある間違いです。この操作は機能的には正しいとしても、この方法でパフォーマンスを測定すると、完全に間違っていることが判明す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として意図されている場合 (例: data acquisition)、このストレージ メディアを徹底的にテストすることをお勧めします。 記憶媒体が期待を満たしていることを確認するには、記憶媒体に対して広範囲かつ長期的なテストを行う必要があります。短い benchmark test は非常に誤解を招く可能性があります。

5.3 大部分の読み取りと書き込み

read() および write() への各関数呼び出しは、オペレーティング システムへの system call をもたらします。したがって、これらの関数呼び出しを実行するには、多くの CPU cycles が必要になります。したがって、実行される system calls が少なくなるように、buffer のサイズが十分に大きいことが重要です。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。

通常、各関数呼び出しの buffer には、128 kB が適切なサイズです。これは、そのような各関数呼び出しが最大 128 kB に制限されることを意味します。ただし、これらの関数呼び出しで転送できるデータは少なくなります。

セクション 4.3 (streamread および streamwrite) で説明したサンプル プログラムはパフォーマンスの測定には適していないことに注意することが重要です。 これらのプログラムの buffer サイズは 128 バイトです (kB ではありません)。これにより例は簡素化されますが、プログラムがパフォーマンス テストするには遅すぎます。

次の shell コマンドを速度チェックに使用できます (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
```

これらのコマンドは、CTRL-C で停止されるまで実行されます。一定量のデータのテストを実行するには、“count=” を追加します。

5.4 CPUの消費量に注意してください

データ転送速度が高いアプリケーションでは、コンピュータ プログラムがボトルネックになることが多く、必ずしもデータ転送がボトルネックになるわけではありません。

よくある間違いは、CPU の機能を過大評価することです。一般に信じられているのとは異なり、データ レートが 100-200 MB/s を超えると、最速の CPUs であっても、データを使って意味のあることを行うのは困難になります。 multi-threading を

使用するとパフォーマンスを向上させることができますが、これが必要であるとは意外かもしれません。

場合によっては、buffers のサイズが不適切であること (前述のとおり) が、CPU の過剰な消費につながる可能性があります。

したがって、CPU の消費量に注意を払うことが重要です。この目的には、“top” などのユーティリティ プログラムを使用できます。ただし、このプログラム (および同様の代替プログラム) の出力は、複数の processor cores を搭載したコンピューター (つまり、最近のほぼすべてのコンピューター) では誤解を招く可能性があります。たとえば、processor cores が 4 つある場合、25% CPU は何を意味しますか? CPU の消費量が少ないのでしょうか、それとも特定の thread 上の 100% でしょうか? “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 つ注意すべき点は、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 がデータ フローを遅くする場合、これはどのように測定されますか?

これを調べる簡単な方法は、“time” ユーティリティを使用することです。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、“dd” の完了にかかった時間が 1.080 秒であったことを示しています。この時間のうち、processor は 5 ms の間に user space program を実行し、system calls で 1.074 秒間ビジーでした。したがって、この特定の例では、processor がほぼ常に system calls の実行で忙しいことがわかります。 “dd” はここでは何も行っていないため、これは驚くべきことではありません。

5.5 読み取りと書き込みを相互に依存させない

双方向通信が必要な場合、1 台の thread だけを使用してコンピュータ プログラムを作成するのはよくある間違いです。このプログラムには通常、読み取りと書き込みを行うループが 1 つあります。 反復ごとに、データは FPGA に向かって書き込まれ、その後データは逆方向に読み取られます。

たとえば 2 つの streams が機能的に独立している場合など、このようなプログラムでは問題がない場合があります。ただし、このようなプログラムの背後にある意図は、FPGA が coprocessing を実行することであることがよくあります。このプログラミング スタイルは、プログラムは処理のためにデータの一部を送信し、その後結果を読み取る必要があるという誤解に基づいています。したがって、反復はデータの各部分の処理を構成します。

この方法は非効率であるだけでなく、プログラムが頻繁に停止します。 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、より適切なプログラミング手法を提案しています。

5.6 host の RAM の限界を知る

これは主に embedded systems および/または revision XL / XXL IP core を使用する場合に関係します。 マザーボード (または embedded processor) と DDR RAM の間のデータ帯域幅には制限があります。この制限は、コンピュータを通常に使用する場合にはほとんど気付かれません。ただし、Xillybus を使用する非常に要求の厳しいアプリケーションでは、この制限がボトルネックになる可能性があります。

FPGA から user space program にデータを転送するたびに、RAM で 2 つの操作が必要になることに注意してください。 最初の操作は、FPGA がデータを DMA buffer に書き込むときです。 2 番目の操作は、driver がこのデータを user space program がアクセス可能な buffer にコピーするときです。同様の理由で、データを逆方向に転送する場合も、RAM で 2 つの操作が必要になります。

オペレーティング システムでは DMA buffers と user space buffers を分離する必要があります。 read() および write() (または同様の関数呼び出し) を使用するすべての I/O は、この方法で実行する必要があります。

たとえば、XL IP core のテストでは、各方向で 3.5 GB/s、つまり合計で 7 GB/s という結果が得られることが予想されます。ただし、RAM は 2 倍のアクセスがあります。したがって、RAM の帯域幅要件は 14 GB/s です。すべてのマザーボードにこの機能があるわけではありません。また、host は他のタスクに RAM を同時に使用することにも注意してください。

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 十分な大きさの DMA buffers

これが問題になることはほとんどありませんが、それでも言及する価値がありま

す。 DMA buffers に対して host に割り当てられている RAM が少なすぎると、データ転送が遅くなる可能性があります。その理由は、hostがdata streamを小さなセグメントに分割することを余儀なくされているためです。これにより、CPU cycles が無駄になります。

すべての demo bundles には、パフォーマンス テストに十分な DMA メモリが搭載されています。これは、IP Core Factory で正しく生成された IP cores にも当てはまります。 “Autoset Internals” が有効になり、“Expected BW” は必要なデータ帯域幅を反映します。おそらくどのオプションでも問題ありませんが、“Buffering” は 10 ms として選択する必要があります。

一般に、帯域幅テストにはこれで十分です。 10 ms 中のデータ転送に相当する合計量の RAM を持つ少なくとも 4 つの DMA buffers。もちろん、必要なデータ転送速度を考慮する必要があります。

5.8 データワードに正しい幅を使用する

当然のことですが、application logic は、FPGA 内の各 clock cycle に対して 1 ワードのデータのみを IP core に転送できます。 したがって、データ ワードの幅と bus_clk の周波数により、データ転送速度には制限があります。

さらに、デフォルト リビジョン (revision A IP cores) では IP cores に関連する制限があります。 ワード幅が 8 ビットまたは 16 ビットの場合、PCIe の機能はワード幅が 32 ビットの場合ほど効率的に使用されません。 したがって、高いパフォーマンスを必要とするアプリケーションとテストでは、32 ビットのみを使用する必要があります。これは、revision B IP cores 以降のリビジョンには適用されません。

リビジョン B 以降、ワード幅は最大 256 ビットになります。ワードの幅は少なくとも PCIe block の幅と同じである必要があります。 したがって、データ帯域幅テストには、次のデータ ワード幅が必要です。

- デフォルトのリビジョン (Revision A): 32ビット。
- Revision B: 少なくとも 64 ビット。
- Revision XL: 少なくとも 128 ビット。
- Revision XXL: 256ビット。

データ ワードが上記で必要な幅よりも広い場合 (可能な場合)、通常はわずかに良い結果が得られます。その理由は、application logic と IP core 間のデータ転送の改善です。

5.9 cache synchronizationによる速度低下

この問題は、x86 ファミリ (32 ビットおよび 64 ビット) に属する CPUs ベースのコンピュータには当てはまりません。 Xillybus を Zynq processor の AXI bus と一緒に使用している人 (たとえば、 Xillinux と併用) も、このトピックを無視してかまいません。

ただし、一部の embedded processors では、DMA buffers を使用する場合、cache の明示的な同期が必要です。これにより、CPU の周辺機器とのデータ転送が大幅に遅くなります。

この問題は Xillybus に固有のものではありません。 同様の動作は、DMA をベースとするすべての I/O (Ethernet、USB、その他の周辺機器など) で観察されます。

cache による速度の低下は、CPU の消費量を見ることで明らかになります。 CPU が system call 状態 (“time” ユーティリティの “sys” 行出力) で過度の時間を費やす場合、cache に問題があることを示している可能性があります。これは、CPU が cache synchronization の実行に多くの時間を費やしているために発生します。

ただし、最初に小型の buffers の可能性を除外することが重要です (上記の 5.3 および 5.7 のセクションで説明したように)。

これらの CPUs には coherent cache があるため、x86 ファミリではこの問題は発生しません。したがって、cache synchronization は必要ありません。 IP core は ACP port を介して CPU に接続されているため、Xillinux にも同じことが当てはまります。

ただし、Zynq processor が Xillybus を PCIe bus とともに使用すると、この問題が発生します。他のいくつかの embedded processors、特に ARM processors も影響を受けます。

5.10 パラメータのチューニング

demo bundles の PCIe block のパラメータは、公表されているデータ転送速度をサポートするために選択されます。パフォーマンスは、x86 ファミリに属する CPU を搭載した一般的なコンピューターでテストされています。

また、IP Core Factory で生成される IP cores は通常、微調整を必要としません。 “Autoset Internals” が有効になっている場合、streams はパフォーマンスと FPGA のリソースの使用率の間で最適なバランスをとる可能性があります。したがって、要求されたデータ転送速度は各 stream で保証されます。

したがって、PCIe block または IP core のパラメータを微調整しようとしてもほとんどの場合無意味です。 IP cores (revision A) のデフォルト リビジョンでは、この

ようなチューニングは常に無意味です。このようなチューニングによってパフォーマンスが向上する場合、問題は application logic または user application software の欠陥である可能性が非常に高くなります。この状況では、この欠陥を修正することで得られるものはさらに多くあります。

ただし、例外的なパフォーマンスを必要とするまれなシナリオでは、要求されたデータ レートを達成するために PCIe block のパラメータをわずかに調整する必要がある場合があります。これは、host から FPGA までの streams に特に関係します。 The guide to defining a custom Xillybus IP core のセクション 4.5 では、このチューニングを実行する方法について説明しています。

この微調整が有益な場合でも、変更されるのは Xillybus IP core のパラメータではないことに注意してください。 PCIe blockのみ調整しています。 IP core のパラメータを調整してデータ転送速度を向上させようとするのはよくある間違いです。むしろ、この問題はほとんどの場合、この章で前述した問題の 1 つです。

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。これらを入手するためのいくつかの代替手段を次に示します。

- “dmesg” コマンドの出力。
- “journalctl -k” コマンドの出力。
- log fileでは。これは、/var/log/syslog や /var/log/messages などのオペレーティング システムによって異なります。

何か問題があると思われる場合は、“xillybus” または “xillyusb” という単語を含むメッセージを検索することをお勧めします。すべてが正常に動作しているように見える場合でも、時々 system log を検査することをお勧めします。

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

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

ただし、メッセージのテキストで Google を使用すると、特定のメッセージを見つけやすくなります。

A

Linux command line の短いサバイバル ガイド

command line インターフェイスに慣れていない人は、Linux コンピュータで作業を行うのが難しいと感じるかもしれません。基本的な command-line インターフェイスは 30 年以上同じです。そのため、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たくさんあります。この短いガイドは単なる入門書です。

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: 現在実行中のプログラムを停止します
- 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: 画面をクリアする
- TAB: command prompt では、すでに書き込まれたものに対して autocomplete を試みます。これは、長いファイル名などに便利です。 名前の先頭に「[TAB]」を入力します。
- 上矢印と下矢印: command prompt では、履歴から以前のコマンドを提案します。これは、実行したばかりのことを繰り返す場合に便利です。過去のコマンドの編集も可能なので、前回のコマンドとほぼ同じことを行う場合にも適しています。
- 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

flags とオプションのすべてを実際に覚えている人はいません。さらに助けを求める一般的な方法が 2 つあります。 1 つの方法は “man” コマンドで、2 つ目の方法は help flag 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
```

「$」という記号は command prompt であり、コンピューターがコマンドの準備ができていることを示すために出力するものであることに注意してください。通常、prompt はより長く、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。

manual page は terminal pager と一緒に示されています。 [space]、矢印キー、Page Up および Page Down を使用して移動し、’q’ を使用して終了します。

コマンドの実行方法についての簡単な概要については、`--help` フラグを使用してください。一部のコマンドは、`-h` または `-help` (単一ダッシュ付き) に応答します。他のコマンドは、構文が間違っている場合にヘルプ情報を出力します。それは試行錯誤の問題です。 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

ファイルが短いことが予想される場合 (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)、そのコンテンツを console に表示するには、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、選択できるエディタが多数あります。最も人気のある (ただし、必ずしも使い始めるのが最も簡単であるとは限りません) は emacs (および XEmacs) と vi です。 vi エディタは習得が難しいですが、いつでも利用でき、常に動作します。

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' は path にすることもできます。たとえば、システムのメイン log file を表示するには、次のようにします。

```
# 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

すべての Linux コンピュータには、ID 0 を持つ “root” という名前のユーザーがいます。このユーザーは superuser としても知られています。このユーザーの特別な点は、すべての操作が許可されていることです。他のすべてのユーザーには、ファイルやリソースへのアクセスに制限があります。すべての操作がすべてのユーザーに許可されるわけではありません。これらの制限は root ユーザーには課されません。

これは、マルチユーザー コンピュータ上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上のすべてのデータを削除することも含まれます。他にも、データを誤って削除したり、コンピュータを一般的に使用できなくなったり、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。 Linux システムの基本的な前提は、root ユーザーが誰であっても、自分が何をしているかを知っているということです。コンピュータは root に「よろしいですか」という質問をしません。

root として動作させるには、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が必要です。物事を台無しにしないための秘訣は、ENTER を押す前によく考え、コマンドが要求どおりに正確に入力されたことを確認することです。通常は、インストール手順に正確に従えば安全です。何をしているのかを正確に理解せずに変更を加えないでください。 root ではない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
```

ユーザーが “sudo” を使用できるようにシステムを構成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。システムはユーザーのパスワードを必要とします (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” が使用できない場合 (主に root パスワードが不明なため)、簡単な代替方法は次のとおりです。

```
$ sudo su
#
```

A.5 選択したコマンド

最後に、一般的に使用される Linux 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 (または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リストします。 “ls -l” は、ファイルとその属性をリストします。
- lspci – bus 上のすべての PCI (および PCIe) デバイスをリストします。 Xillybus が PCIe ペリフェラルとして検出されているかどうかを確認するのに役立ちます。 lspci -v、lspci -vv、lspci -n も試してください。
- lsusb – bus 上のすべての USB デバイスをリストします。 XillyUSB がペリフェラルとして検出されているかどうかを確認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。 argument 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単純な入出力を行うためのスイスナイフとして機能しました。
- man– コマンドの manual page を表示します。 “man -a” も試してください (1 つのコマンドに複数の manual page が存在する場合があります)。
- less–terminal pager。 standard input 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。 -f フラグを使用するとさらに良いでしょう: 末尾と新しい行を到着時に表示します。ログ ファイルに適しています。たとえば (root など):

```
# 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。さらに良いのは、“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。

そして、いくつかの高度なコマンド:

- grep – ファイル内の textual 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。