(機械で日本語に翻訳)

Getting started with Xillybus on a Windows 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 This document may also be slightly outdated in relation to the original.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、 Windows hostに Xillybus / XillyUSB 用の 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。

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 (PCI Express または USB 3.xを介して) 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。

これに到達するための手順は、次の 2 つ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

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 バイナリは PCIe トランスポートを使用してすべての Xillybus IP cores をカバーし、別のバイナリは USBベースの coresをカバーします。 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。

PCIe driverによって作成されたこれらの pipeに似た device filesは、 \\.\xillybus_*something*として表示されます。同様に、 XillyUSB の driver は、 \\.\xillyusb_00_*something*のように device files を作成します。ここで、 00 部分はデバイスのインデックスで

す。 01、 02 などで XillyUSB を複数台同時に接続する場合に交換する部品です。

ただし、 PCIe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 Windowsを参照してください。

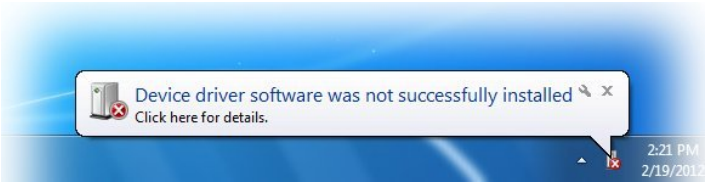
2

host driverの取り付け

2.1 インストール手順

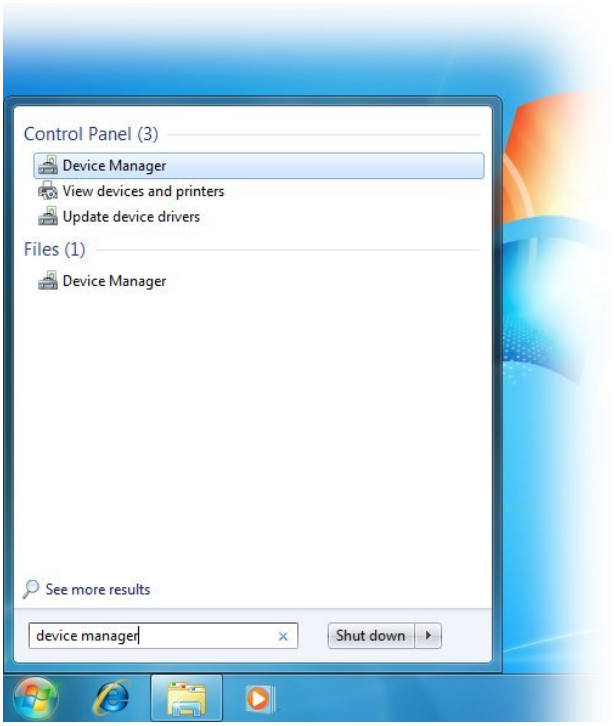
Xillybus または XillyUSBに Windows driver をインストールすることについて特別なことは何もありません。以下に説明する手順は、ディスク上の特定の場所から device driver をインストールするための一般的な方法です。

Windows が boot プロセスを初めて終了し、 PCIe Xillybus IP core が hostに表示される場合、または XillyUSB デバイスが接続されている場合、警告バブルが表示される可能性があります。ハードウェアが見つかったが、 driver がインストールされていないと表示されます。

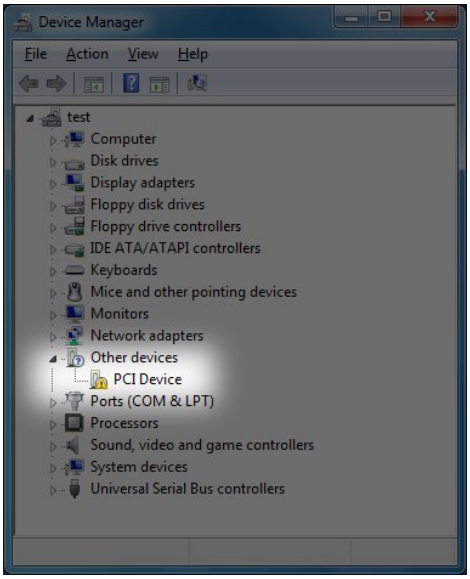


これは正常で、 Windows がまだ認識していないものを検出した兆候です。

Device Managerを実行することから始めます。これは、 “Windows start” ボタンをクリックして次のように “device manager” と入力し、一番上の項目をクリックすることで最も簡単に実行できます。



開いた Device Manager は次のようになります (重要な部分が強調表示されています)。



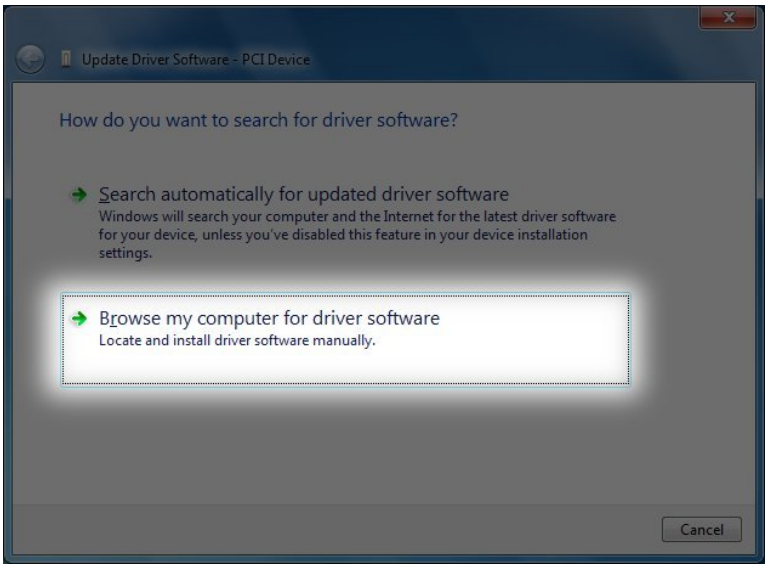
このスクリーンショットは、 PCIe シナリオに関連しています。 XillyUSBについては、 “Universal Serial Bus controllers” グループに新しい項目が表示されます。

Device Managerに新しいエントリが表示されない場合、 FPGA に bitstream が適切にロードされていると仮定すると、いくつかの原因が考えられます。

- FPGAの PCIe インターフェイスが検出されません。おそらく、ボードの構成ミス (ジャンパーなど)、ピン割り当ての誤り、 reference clock 周波数の誤りなどが原因です。この種の問題は、 FPGA の PCIe ブロック (Xilinx または Intel) は検出されません。このような問題は、 PCIe busとのインターフェイスにこの PCIe ブロックを使用する Xillybusとは関係ありません。
- FPGA カードが PCから電源を取得する場合、 PC computer の電源がオンになると、 bitstream が FPGA にロードされます。この場合のみ、 FPGA が bitstream をロードするのが遅すぎるため、 BIOS が boot中に PCIe インターフェイスを検出しなかった可能性があります。

  Action > Scan for New Hardware も機能する可能性がありますが、 Windows restart を (電源を入れ直さずに) 実行することは、これを修正する安全な方法です。
- Xillybus / XillyUSB driver はすでにインストールされています。その場合、 Device Manager はインストール手順の最後に示されている例のようになります。

“PCI Device” アイテムを右クリックし、次のウィンドウを開く “Update Driver Software...” を選択します。



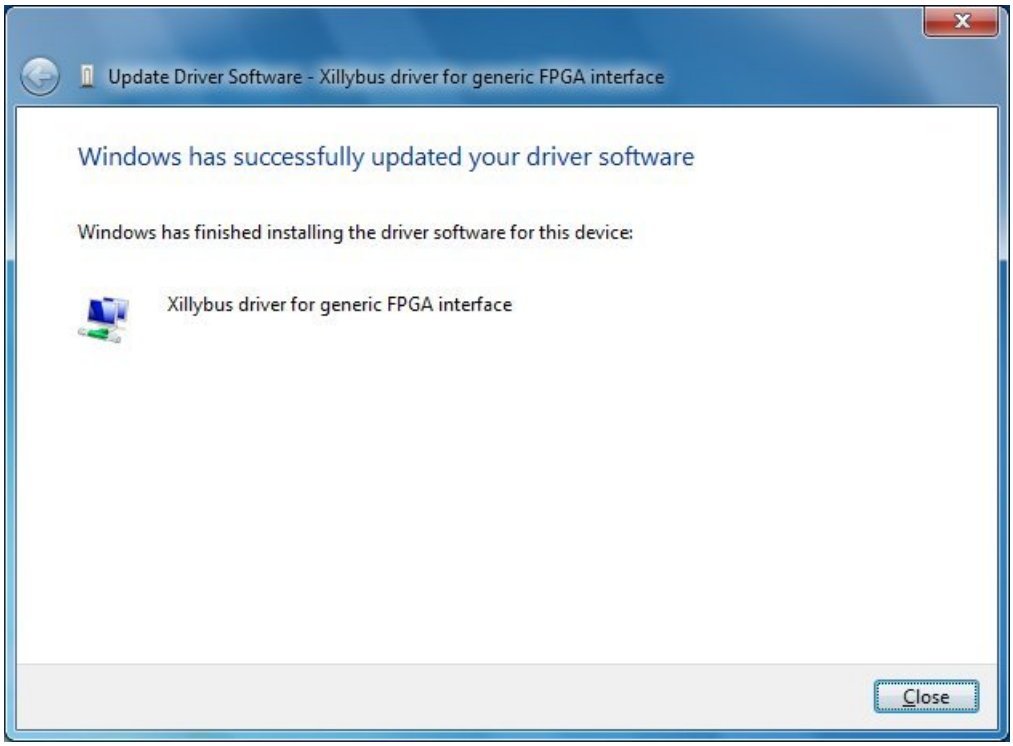
“Browse my computer for driver software”を選択すると、次のウィンドウが開きます。

“Browse...” ボタンを使用して、 driver が解凍された場所に移動します。ディレクトリ名は、 driverのバージョンと解凍先によって異なる場合があります。

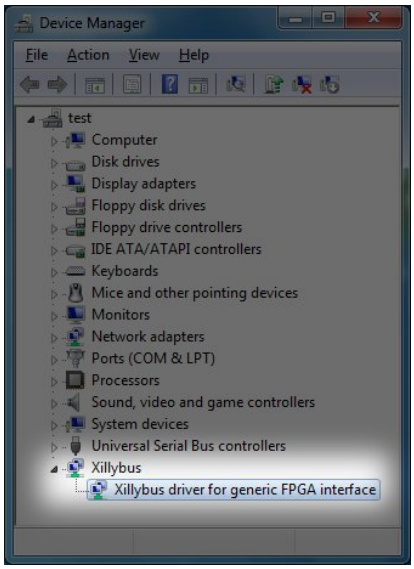
次のステップは、インストールを確認することです。

“Install”をクリックします。 driver をインストールするプロセスは、 Windows 7では10 20 秒かかり、 Windowsの新しいバージョンではおそらくそれよりもはるかに短い時間です。

次のウィンドウで、インストールが正常に完了したことが通知されます。



Device Manager には、新しくインストールされたデバイスが表示されます。



繰り返しますが、これらのスクリーンショットは PCIeの Xillybus に関連していますが、プロセスは XillyUSBでも同じですが、わずかな違いがあります。特に、 Device Manager の新しいグループは、 “Xillybus”ではなく “XillyUSB”と呼ばれます。

この時点で、 driver がシステムにインストールされ、自動的にロードされます。システムが PCIe bus上の Xillybus IP core で boot を実行するたびに、または XillyUSB デバイスが hostに接続されると、リロードされます。

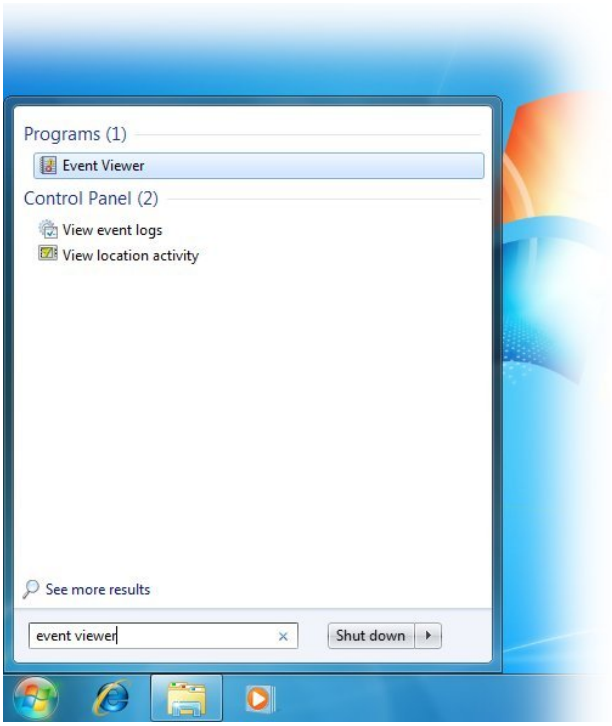
次に説明するように、 Xillybusの log messagesを表示するように Event Viewer をセットアップすることをお勧めします。

2.2 診断情報の取得

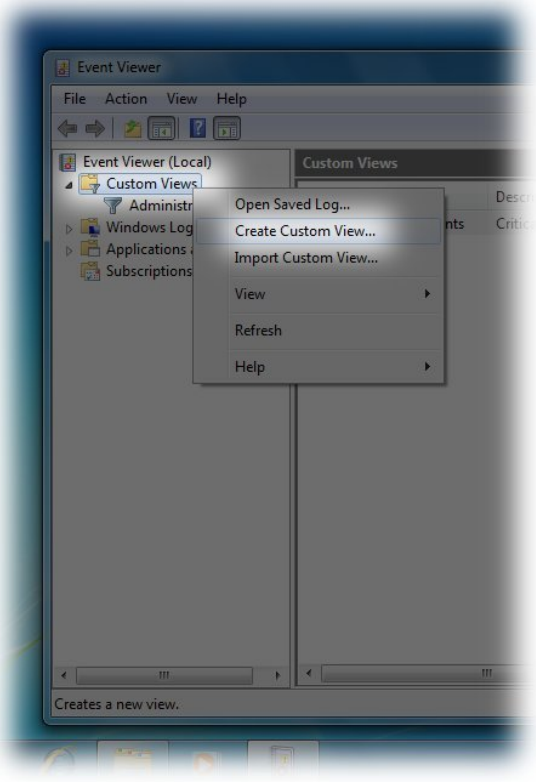
Xillybus / XillyUSB 用の driver は、診断メッセージをオペレーティング システムのメイン event loggerに送信します。これらのメッセージには、 driver が初期化に失敗したときの問題 (たとえば、 DMA buffersに十分なメモリがなかった) に関する情報や、予期しない動作を理解するのに役立つその他のメッセージが含まれます。

次に説明する手順は、 event viewerでカスタム ビューを作成する方法を示しているため、 Xillybus関連のメッセージのみが表示されます。

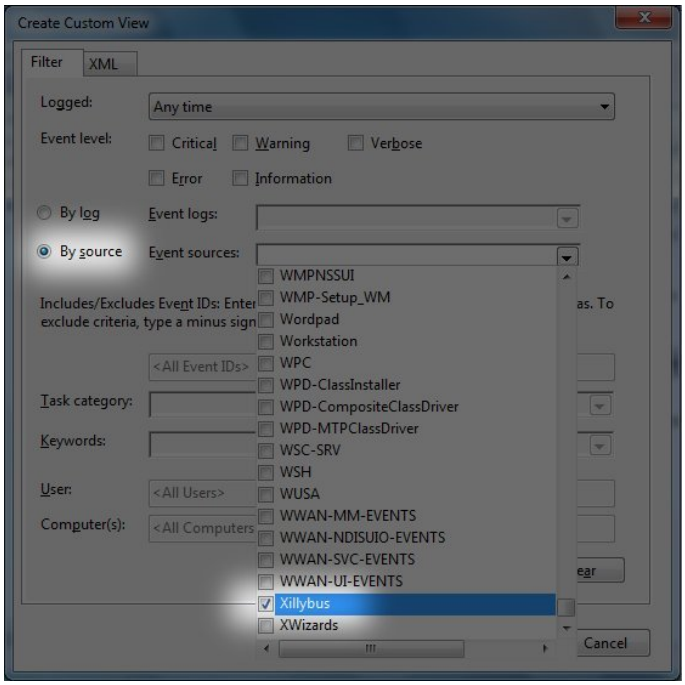
まず、 Event Viewerを開きます。これは、 “Windows start” ボタンをクリックして次のように “event viewer” と入力し、一番上の項目をクリックすることで最も簡単に実行できます。



Event Viewer が開きます。 Xillybus メッセージのみのカスタム ビューを作成するには、 “Custom Views” を右クリックし、メニューから “Create Custom View...” を選択します。

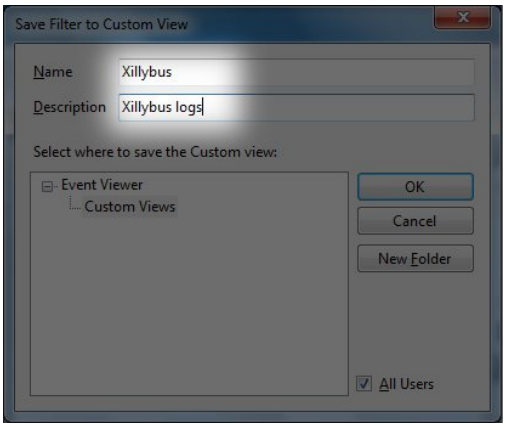


カスタム ビューのフィルタリングを定義するためのウィンドウが開きます。 “By source” を選択し、 drop-down menuで必要に応じて Xillybus または XillyUSBを選択します。他のオプションはデフォルトのままにします。

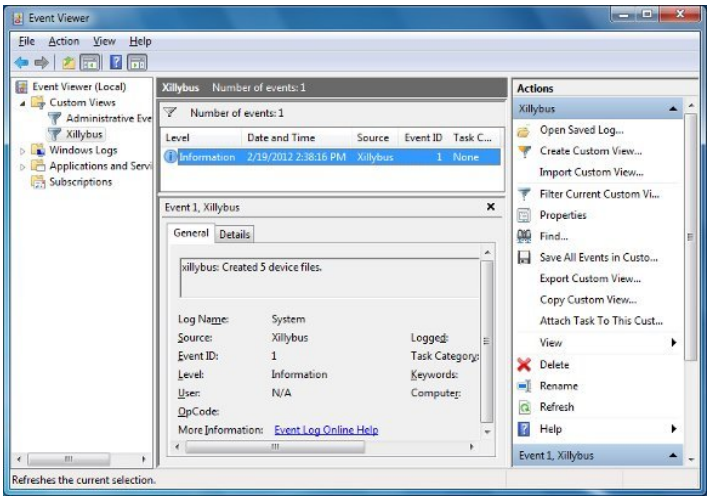


drop-down リストに Xillybus / XillyUSB エントリが見つからない場合は、 driver が正しくインストールされているかどうかを確認してください。

“OK”をクリックすると、このカスタム ビューに名前と説明を割り当てるためのウィンドウが開きます。これは個人的な選択に開放されています:



“OK”をクリックすると、 Event Viewer は次のようになります。



上記の例では、このビューに 1 つのログ エントリがあり、 Xillybus が適切に開始され、5 つの device filesが作成されたことを通知しています。これは、 driverを正常にインストールした直後に予想されることであり、 FPGA には demo bundleがロードされています。

コンピューターの reboot が発生してもログ エントリは削除されないため、このカスタム ビューには、 driver がインストールされてからの履歴が表示されます (ログがクリーンアップされない限り)。

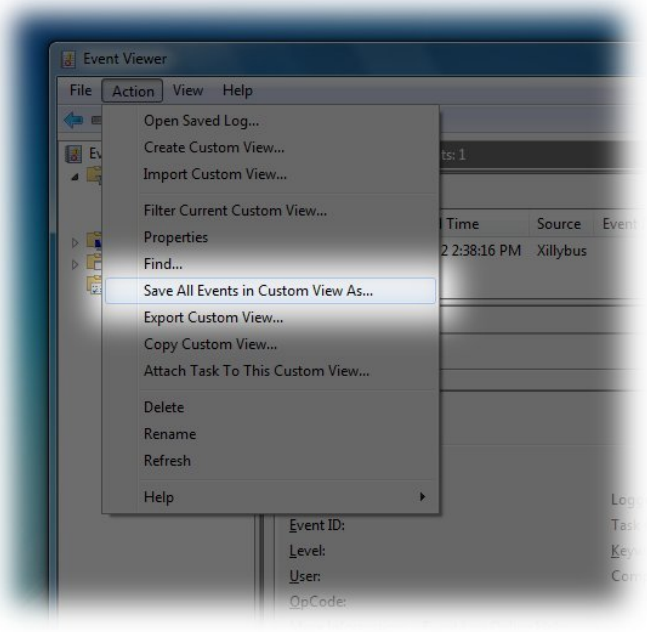
PCIe driverについては、多くのログ メッセージの説明が次の場所にあります。

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。

いずれにせよ、問題がすぐに解決されない場合は、発生する可能性のある問題を報告する電子メールをためらわずに送信してください。

ログ メッセージをファイルにエクスポートすることもできます。サポートを依頼するときは、貴重な情報を含むログ メッセージ ファイルを添付することをお勧めします。これを行うには、 “Action” メニュー項目と “Save All Events in a Custom View As...”を選択します。



これにより、ファイル選択ウィンドウが開きます。推奨される出力形式は CSVです。

3

コマンド ライン テスト

3.1 序章

driver は、トランスポートのタイプ (PCIe または USB) に応じて、 `\\.\xillybus_` または `\\.\xillyusb_nn_` プレフィックスを持つ device files を介してそのインターフェイスを公開します。これらのオペレーティング システム オブジェクトはファイルのように動作し、同じインターフェイスでアクセスできます。 Microsoft Windows の内部に精通していない人は、これは信頼できないアクセス方法だと誤解するかもしれませんが、実際には、 Windows コンピュータのハードウェア アクセスの多くは、このように device files を介して行われます。

これらの device files が user applicationに直接公開されることは実際にはまれです。むしろ、それらは通常、 DLLも提供するハードウェアのベンダーによって定義された API を介してアクセスされます。ただし、この DLL は通常、 APIで公開されている wrapper functions 内の device files にアクセスします。

Xillybusのインターフェースは非常に単純であるため、この DLL は不要であり、user application ソフトウェアは device files に直接アクセスします。

Microsoft Windowsのパラダイムは、 device files が直接アクセスされないということであるため、それらのリストをすぐに取得する方法はありません。これは、 PCIe バリアント用に demo bundle によって生成された device files のリストです。

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

- \\.\xillybus_mem_8

hostに接続された単一の XillyUSB デバイスの場合、これらは device filesです。

- \\.\xillyusb_00_read_8
- \\.\xillyusb_00_write_8
- \\.\xillyusb_00_read_32
- \\.\xillyusb_00_write_32
- \\.\xillyusb_00_mem_8

IP Core Factoryによって生成されたカスタム IP coresの device files のリストは、ダウンロードした zip ファイルに含まれる README ファイルにリストされています。

ほとんどのプログラミング言語 (C/C++ を含む) を使用する場合、 backslashes の前に escape character が必要であることに注意してください。したがって、 Cで関数の引数として使用される場合、 device file の名前は \\\\.\\xillybus_read_8 のように記述されます。

WinObj ユーティリティ (Microsoftのサイトからダウンロードできます) を使用すると、 Windowの内部オブジェクト構造を参照できます。 Xillybus / XillyUSB device files は、よく知られている C:、 COM1: などとともに、 GLOBAL??という名前の “subdirectory” で symbolic links として表示されます。

command-lineを使用して Xillybus / XillyUSB device files の名前を取得するには、Microsoftの Web サイトから accesschk ユーティリティをダウンロードし、

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

この操作では、他の多くのグローバル device files がリストされることに注意してください。

3.2 command line インターフェースの使用

Xillybus device files は、 Windowsの user space アプリケーションとうまく連携するように設計されています。ただし、 command-line tools を使用すると、インターフェイスを使用した最初のステップに最適であり、データ出力を簡単に取得できます。

command line でのテストは、 4で説明されているように、サンプル host アプリケーションで実行できます。または、次に説明するように、汎用の command-line utilities を使用することもできます。

すぐにプログラミングに取り掛かることを好む人は、 Xillybus host application programming guide for Windowsを参照することをお勧めします。

残念ながら、 Windows のコマンドライン ツールは、すべての UNIXベースのシステムで利用できるものほど豊富ではありません。 DOS command promptの組み込みコマンドは、期待どおりに使用しないと役に立たなくなります。また、 Windows で提供される shell プログラムの標準セットは、明らかに何か役に立つものではありませんでした。幸いなことに、 UNIX コマンドは多くの点で Windows に移植されています。これらの移植されたユーティリティを使用すると、 Getting started with Xillybus on a Linux host に示されている “Hello world” の使用例を Windows マシンで実行できます。

注意すべき点がいくつかあります。

- Windows は、よく知られている UNIX `/dev/` ディレクトリではなく、 `\\.\` パスにそのグローバル device files を保持します。たとえば、 device file が Linux指向のコンテキストで `/dev/xillybus_read_8`として指定されている場合、 Windows ユーザーは `\\.\xillybus_read_8` を参照する必要があります。
- backslash は一般に escape characterとして解釈されるため、 Cygwin およびほとんどのプログラミング言語で使用する場合、 `\\.\` パスを `\\\\.\\` として記述する必要があります。つまり、 `/dev/xillybus_read_8` は Cygwin では `\\\\.\\xillybus_read_8` としてアクセスされ、 C や Perlなどの言語でも同じことが言えます。段落 3.4を参照してください。
- DOS 環境は backslash を escape characterと見なさないため、 device file を引数としてコマンドライン バイナリ プログラムに渡す場合、 DOS command window には backslash の escape はありません。内部で arguments を処理する方法に応じて、 scripts内で異なるレベルの escaping が必要になる場合があります。

3.3 UNIXのような代替手段

セクション 4 の Windows 用のサンプル host アプリケーションは、無料でダウンロードできる Microsoftの compilerに基づいていることに注意してください。このセクションの UNIX ユーティリティに関する説明は、これらのサンプル アプリケーションとは関係ありません。

Windows マシンで UNIX ユーティリティを実行するには、いくつかの可能性があります。これらはいくつかの提案です:

- Cygwinをインストールします。これは重量級の選択肢であり、完全な GNU C compiler と開発環境が含まれる場合があります。これは、 Linux の感覚が必要な場合や、単純なシステム インターフェイスに基づくコマンド ライン ツールのみを開発する場合に推奨される選択肢です。
- Windows用の UNIX ユーティリティの Gnuwin32 port をダウンロードしてインストールします。特に、 Coreutils および Util-Linux-NG パッケージのみをインストールすると、このガイドの例で使用されるユーティリティがカバーされます (両方をインストールする場合)。 Gnuwin32のセットアップ ユーティリティは、 DOS Command Windowの execution pathを変更**しない**ことに注意してください。
- unixutils サブディレクトリにある Windows パッケージ (ダウンロード可能) で Xillybus が提供するユーティリティを使用します。これらは、このサイトの例をサポートするために厳選された Gnuwin32 ユーティリティのセットです。これは、コマンド ライン ユーティリティの使用を最小限に抑える場合に推奨される選択肢です。

3.4 Cygwin ノート

Xillybus は、 Cygwinで広範囲にテストされています。 Cygwin bash terminal での次のセッションは**正しい**ことに注意してください。

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

- この警告メッセージは、パス名で backslashes を初めて使用したときに表示されますが、**無視し**てください。 environment variable を推奨どおりに設定して、それを回避してください。通常のファイル名の場合、確かにスラッシュを使用することをお勧めしますが、 Cygwin は `//./` を `\\.\` に変換し**ない**ため、 backslashes は必須です。

- Cygwin は単一の backslash を escape characterとして扱うため、 backslashes は 2 倍になっていることに注意してください。

4

サンプル host アプリケーション

4.1 全般的

Windows用の Xillybus パッケージには、 demoapps サブディレクトリの下に 6 つの単純な C コマンドライン プログラムがあります。 precompiled executables は precompiled-demoappsにあります。これらの C ソースは、 Windows SDKの一部として無料でダウンロードできる Microsoft's Visual C++ compilerを使用した compilation 用に作成されました。これらを Visual Studio に採用することは難しくありません。

Cygwin で compilation を実行することを好むユーザーは、 Linuxの手順に従い、プログラムを実行するときに /dev/ プレフィックスを \\\\.\\ に置き換えます。MinGW ツールセットを使用する場合、同様の手順が可能です。詳細については、コマンドライン ノートを参照してください。

とにかく、 Microsoftの compilerを使い続けたい人のために、 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、 "nmake" ユーティリティが 5 つのプログラムのうち compilation を実行するために使用する規則が含まれています。
- 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。
- fifo.c – 連続データ streaming用のユーザー空間 RAM FIFO の実装を示します。このプログラムは Xillybus host application programming guide for Windowsで説明されています。
- winstreamread.c– ファイルから読み取り、データを standard outputに送信します。このプログラムは、標準の APIの代わりに、ファイルにアクセスするために Windows によって提供される API の使用を示します。

最後のプログラムを除くすべてのプログラムは、 Microsoftの compilerを備えた compilation を対象としていますが、従来の UNIX スタイルで書かれています。

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、 fifo.c を除いて、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。セクション 5で説明されているように、特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 _open()、 _read()、 _write() などの関数を使用することに注意してください。後者は、 C library 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。

fifo.c プログラムは、連続 data streamingのための 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。

4.2 Compilation

Xillybusが提供する Windows パッケージには precompiled binaries がありますが、デモ アプリケーションの compilation は、コードを変更するために明らかに必要です。

Microsoft では、 software development kit (SDK) を無料でダウンロードできます。このキットには、特に C compiler とビルド ユーティリティが含まれており、以下で説明する compilation 手順で使用されるツールセットです。ただし、 compilation

は、他の compilers 、特に Microsoftの compilers でも簡単であることが期待されます。

Windows SDKをインストールした後、 “Start menu” で Program Files エントリを開き、 Microsoft Windows SDK v7.1 > Windows SDK 7.1 Command Prompt を選択します (他のバージョンも適用されます)。これにより、特定の environment variables セット (および黄色のテキスト フォント) を含む DOS command ウィンドウが開きます。

C ファイルがある場所にディレクトリを変更します。

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

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

```
> nmake

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

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

“cl” で始まる 6 行は、 “nmake” 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 “nmake”を使用してください)。 “link” コマンドについても同様で、オブジェクト ファイルをライブラリにリンクし、 executablesを作成します。

“nmake” 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 “nmake”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 に対して “nmake” を呼び出します。 compilationによって生成された executables を削除するには、 “nmake 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 “nmake” 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。

executables (および object files) は、ビルド プロセス中に生成される XP32_DEBUG サブディレクトリにあります。ディレクトリの名前が示すように、これらのファイルは 32-bit Windows XPを対象としていますが、64 ビット プラットフォームを含むそれ以降の Windows プラットフォームでも実行されます。

4.3 実行

compilationによって作成されたばかりの executables に基づいて、または precompiled binariesを使用して、2 つの terminals を使用した単純な loopback の例を次に示します。 compilationの後、ディレクトリを XP32_DEBUG に変更するか、 compilation をスキップして precompiled-demoapps サブディレクトリを使用します。最初の DOS command promptで次のように入力します。

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

backslashesの escaping がないことに注意してください。これをプレーンな DOS command ウィンドウではなく Cygwin で実行すると、 `\\\\.\\xillybus_read_8`になります。

これは device fileから読み取るプログラムです。 2 番目の DOS ウィンドウで、正しいディレクトリに変更した後:

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

次に、2 番目の DOS ウィンドウ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 DOS ウィンドウに表示されます。 standard inputの予期される動作に従って、 ENTER が押されるまで何も送信されないことに注意してください。

これら 2 つの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。それ以外の場合は、 2.2の段落で説明されているように、 Event Viewerのエラーを確認してください。

これら 2 つのコマンドのいずれも、 CTRL-Cで停止できます。

この loopback テストは、 FIFO の両端が 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)。

device files と FIFO の間に \\.\xillybus_read_32 と \\.\xillybus_write_32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。

Linux用の streamwrite のバージョンとは異なり、 Windows 用の streamwrite は consoleの設定を変更しようとしないため、行ごとに動作することに注意してください。

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 (Windows packageの unixutils ディレクトリにあります) を使用して現在の状況を確認します。

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

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 \\.\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 \\.\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 \\.\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 の段落に示されているように、“dd” コマンドを使用することです。このユーティリティは、 Windows package for Xillybusの executable として利用できます。

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。

代わりに、 Xillybusの Windows パッケージの unixutils サブディレクトリにある “dd” コマンドライン アプリケーションを使用することをお勧めします。 Cygwin を使用すると、デフォルトで “dd” ユーティリティが提供されるだけでなく、 /dev/zero および /dev/null がデータ ソースおよびデータ コンシューマとして提供されるため、簡単に使用できます。

次の shell コマンドを Cygwin 内で使用して、速度をすばやくチェックできます (必要に応じて `\\\\.\\xillybus\_*` の名前を置き換えてください)。

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

```
dd if=\\\\.\\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。

コマンドライン テストの詳細については、セクション 3 を参照してください。

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、 Task Managerなどを使用して、 CPU の消費量に注意することが重要です。それにもかかわらず、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: quad-core コンピュータ上の 25% CPU は、 CPU 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? system monitoring のさまざまなユーティリティでは、この情報がさまざまな方法で表示されます。

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 Windows 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

これは主にリビジョン XL / XXL IP coreを使用する場合に当てはまります。各マザーボードの DDR RAM メモリ ユニットへの帯域幅は限られています。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 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、システムの event logで意味のある log messages を生成するように設計されました。したがって、 2.2の段落で説明されているように、 Xillybus関連のイベントのログをフィルタリングしてメッセージを探すことをお勧めします。

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

PCIe driverのメッセージの一部は、次の場所にリストされ、説明されています。

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。

問題がすぐに解決されない場合は、電子メールで支援を求めることを躊躇しないでください。