(機械で日本語に翻訳)

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 This document may also be slightly outdated in relation to the original.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

このガイドでは、Xillybus の demo bundle をベースにした bitstream がすでに FPGA にロードされており、FPGA が host によって (PCI Express または USB 3.x を介して) peripheral として認識されていることを前提としています。

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

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。 XillyUSB では別の driver が使用されます。

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 \\.\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、\\.\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 Windowsを参照してください。

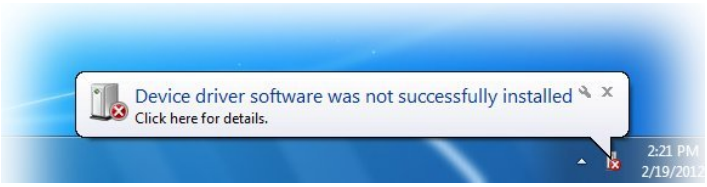
2

host driver の取り付け

2.1 インストール手順

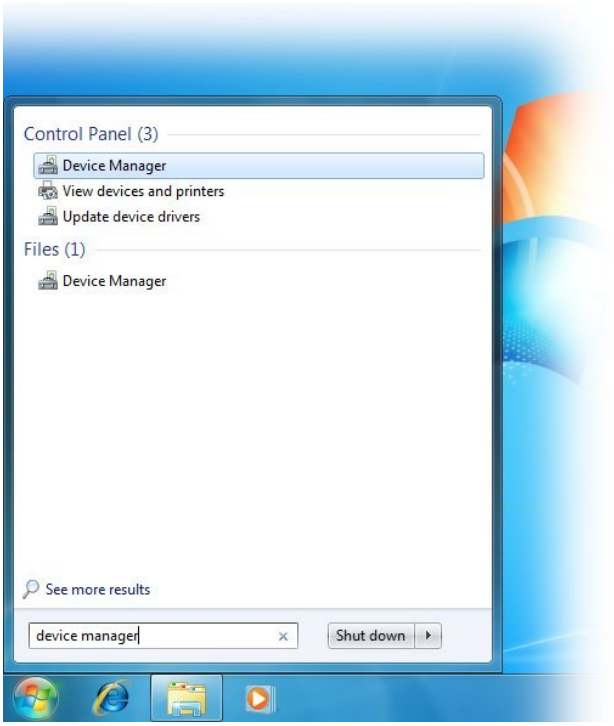
Windows driver を Xillybus または XillyUSB にインストールする場合、特別なことは何もありません。以下で説明する手順は、ディスク上の特定の場所から device driver をインストールする一般的な方法です。

Windows が boot プロセス中に初めて PCIe Xillybus IP core を検出すると、次のような warning bubble が表示される可能性があります。

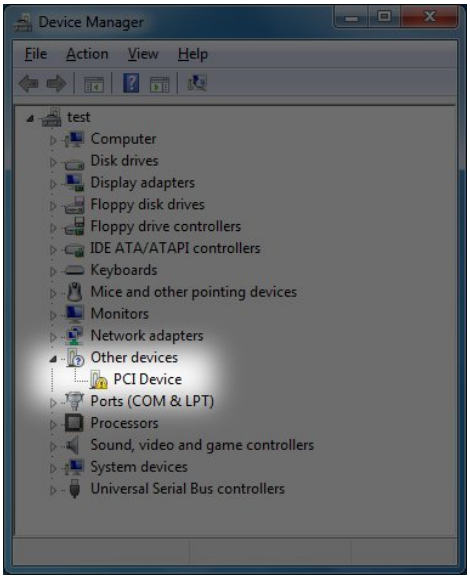


この通知は、新しいハードウェアが見つかったが、関連する driver がインストールされていないことを示しています。この状況は正常であり、Windows がまだ認識していないものを検出したことを示しています。 XillyUSB device を初めてコンピュータに接続したときも、同じ結果が予想されます。

このイベントに応答して、Device Manager を実行することから始めます。これは、下の図に示すように、[“Windows start”] ボタンをクリックしてから「“device manager”」と入力するのが最も簡単です。その後、上部のメニュー項目をクリックします。



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。

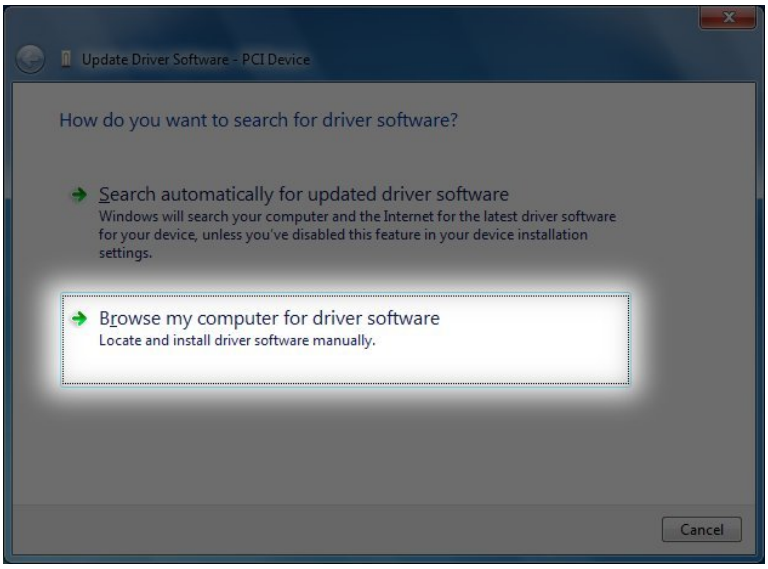
新しいエントリが Device Manager に表示されない場合は、いくつかの原因が考えられます。

- FPGA には間違った bitstream がロードされているか、bitstream がまったくロードされていません。
- FPGA board が PC から電源供給を受ける場合、PC computer の電源がオンになると、bitstream が FPGA にロードされます。この使用シナリオでは、FPGA による bitstream のロードが遅すぎるため、BIOS が boot 中に PCIe インターフェイスを検出できなかった可能性があります。 BIOS がコンピュータを初期化するとき、bitstream はすでに FPGA 内に存在している必要があります。

  この問題を解決するには、(コンピュータの電源を切らずに) Windows restart を実行するのが安全な方法です。ただし、Action > Scan for New Hardware を実行しても機能する可能性があります。
- board の間違った構成 (jumpers、DIP switches など)、間違った pin assignment、reference clock の間違った周波数など。

  この種の問題は、FPGA 上の PCIe block が検出されないために発生することに注意してください。このような問題は、PCIe bus とのインターフェイスにこの PCIe block (AMD または Intel によって供給される) を使用する Xillybus とは関係がありません。
- Xillybus / XillyUSB driverはすでにインストールされています。このシナリオでは、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” 項目を右クリックし、“Update Driver Software...” を選択します。次のウィンドウが開きます。



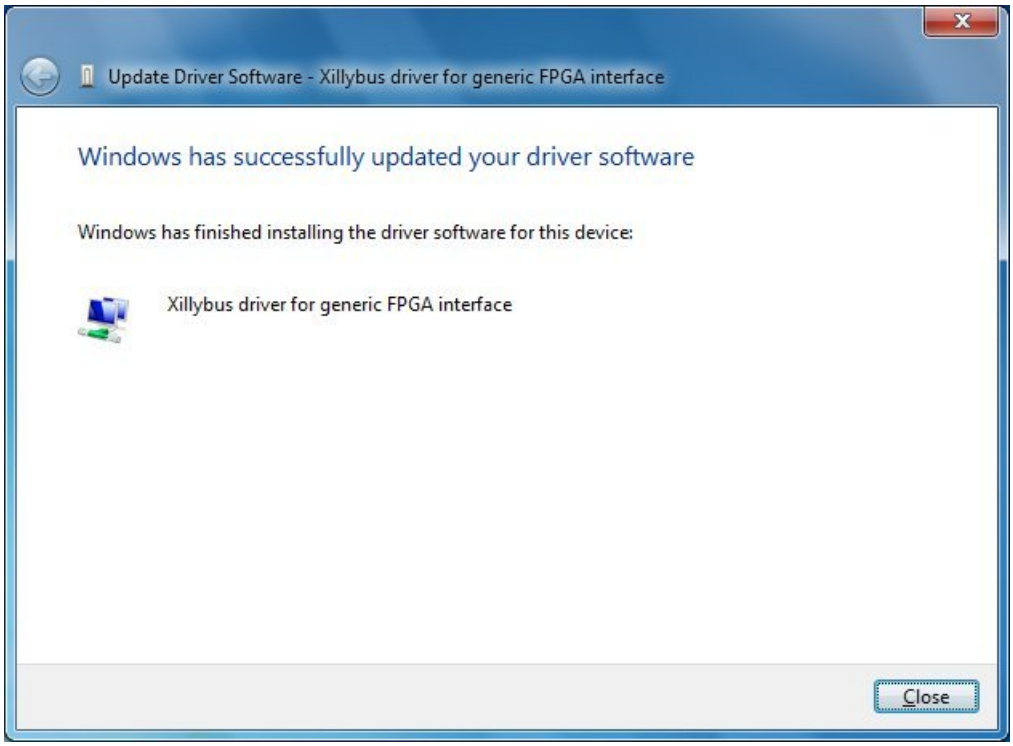
“Browse my computer for driver software”を選択します。これは次のウィンドウです。

“Browse...” ボタンを使用して、driver が保存されている場所 (driver が解凍された後) に移動します。

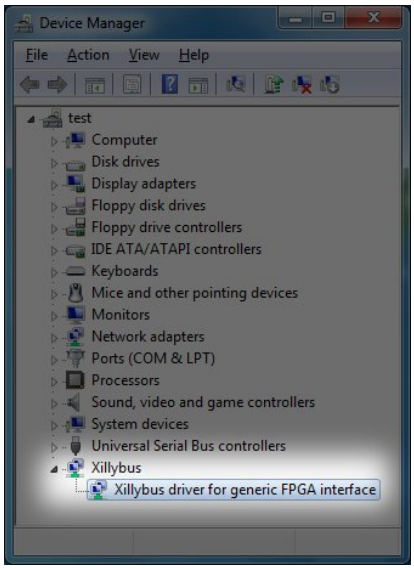
次のステップは、インストールを確認することです。

「“Install”」をクリックします。 driver のインストール プロセスは、Windows 7 では 10 20 秒かかります。 Windows の新しいバージョンでは、通常、必要な時間が大幅に短縮されます。

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 device が表示されます。



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

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。

上記のスクリーンショットは Xillybus または PCIe に関連しています。ただし、プロセスは XillyUSB でも同じですが、若干の違いがあります。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。

2.2 device files

アプリケーション コンピュータ プログラムは、標準の file I/O API を使用して Xillybus IP core と通信します。ただし、通常のファイルにアクセスする代わりに、device files は Xillybus と連携するために使用されます。

したがって、Xillybus の driver の目的は、FPGA と通信するメカニズムとしてオペレーティング システム内にこれらの device files を作成することです。これらの device files は、Microsoft の用語では “Windows objects” と呼ばれます。

単純なコンピュータ プログラムから Windows objects に直接アクセスするのは、信頼できない方法のように思えるかもしれません。ただし、Microsoft Windows の内部構造に詳しい人は、ソフトウェアとハードウェアのインターフェイスが、多くの

場合、まさにこのように行われていることを知っています。これらの device files がアプリケーション ソフトウェアに直接公開されることは実際には珍しいことです。むしろ、ハードウェアの製造元は、プログラムが API 経由でハードウェアにアクセスできるようにする DLL を供給することがよくあります。この DLL は、必要な機能を実現するために裏で device files を使用しています。

Xillybusのインターフェースはシンプルなので、そんなDLLは必要ありません。したがって、user application software は device files に直接アクセスします。しかし残念ながら、Windows には Xillybus の device files のリストを取得するための簡単な方法がありません。これは、Windows objects へのアクセスは高度なテクニックであると考えられているためです。したがって、オペレーティング システムからこの情報を取得するには、ユーティリティ プログラムをダウンロードする必要があります。ただし、以下で説明するように、device files のリストは他のソースからも入手できます。

WinObj ユーティリティ (Microsoft のサイトからダウンロード可能) を使用すると、Window の object tree でのナビゲーションが可能になります。 Xillybus / XillyUSB device files は、“subdirectory” では symbolic links として GLOBAL?? という名前で見つかります。 C: や COM1: など、他のよく知られた Windows objects も同じ場所にあります。

accesschk という名前の command-line ツールもあります。このツールは Microsoft's website からダウンロードできます。 Xillybus / XillyUSB device files の名前を取得するコマンドは次のとおりです。

```
> 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。

これら 2 つのツールを使用して device files のリストを取得することは可能ですが、その必要はありません。 device files の名前は事前にわかっています。

PCIe インターフェイスが使用される場合、Xillybus の device files には `\\.\xillybus_` 形式のプレフィックスが付きます。 XillyUSB の場合、プレフィックスは `\\.\xillyusb_nn_` です。

これは、demo bundle の PCIe バリアントによって生成される 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 になります。

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

IP Core Factory で生成されたカスタム IP core については、次のようになります。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 ファイルにあります。

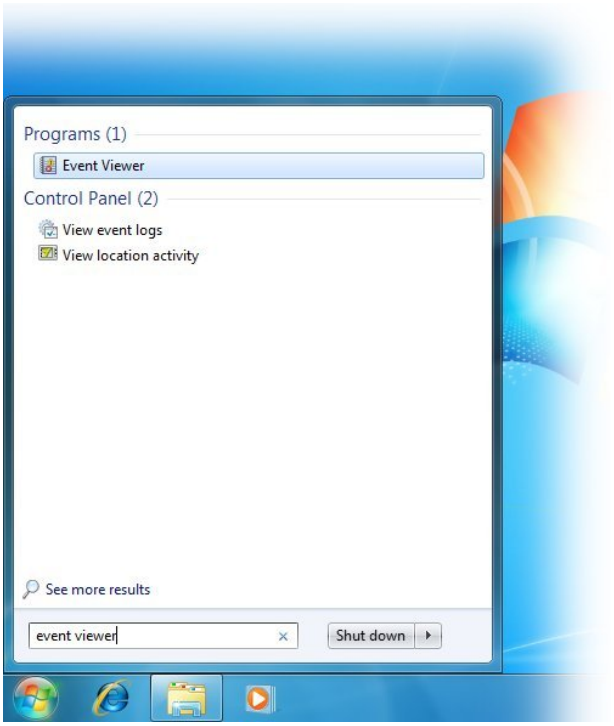
多くのプログラミング言語 (例: C/C++) では、ファイル名の backslashes の前に escape character が必要であることに注意してください。したがって、device file の名前を `\\\\.\\xillybus_read_8` などと記述する必要がある場合があります。

2.3 診断情報の取得

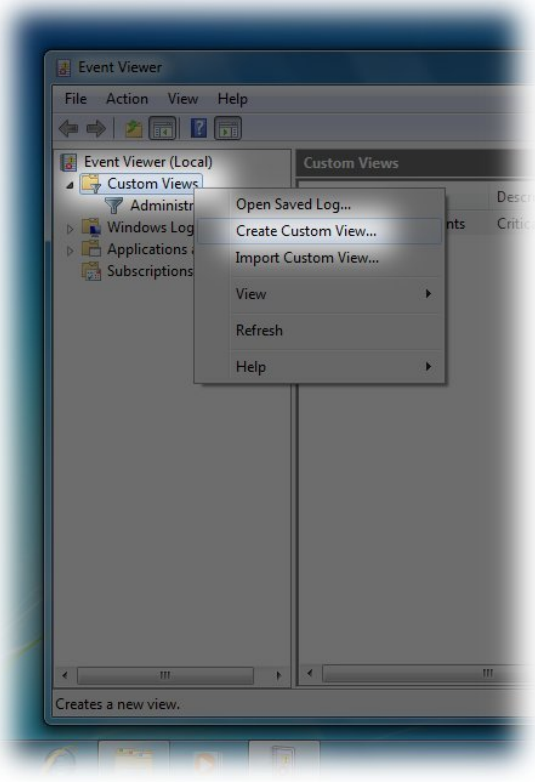
Xillybus / XillyUSB 用の driver は、オペレーティング システムのメイン event logger に診断メッセージを送信します。これらのメッセージには、driver が初期化に失敗したときに何が問題になったかに関する情報が含まれています (たとえば、DMA buffers に十分なメモリがありませんでした)。問題解決に役立つその他のメッセージも送信されます。

次に説明する手順は、Event Viewer 内に custom view を作成する方法を示しています。このタスクの目的は、Xillybus または XillyUSB に関連するメッセージを表示することです。

まず、Event Viewerを開きます。これは、以下に示すように、[“Windows start”] ボタンをクリックしてから “event viewer” と入力するのが最も簡単です。上部のメニュー項目をクリックします。

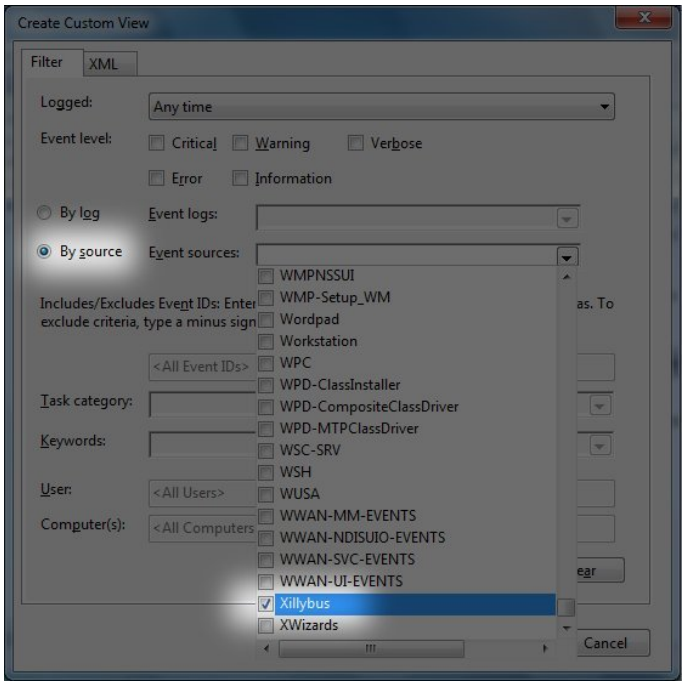


Event Viewerが開きます。 “Custom Views” を右クリックし、メニューから “Create Custom View...” を選択します。



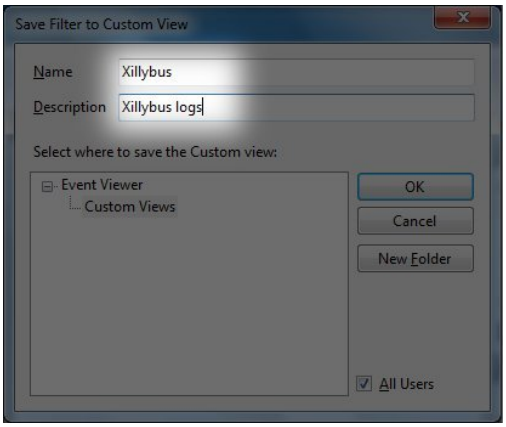
「“Create Custom View”」というタイトルのウィンドウが開きます。このウィンドウの目的は、表示するメッセージを選択するフィルターを定義することです。 “By source”を選択します。 drop-down menuではXillybusを選択してください。他のオプションについてはデフォルトのままにします。

代わりに XillyUSB からメッセージを取得するには、メニューで XillyUSB を選択します。

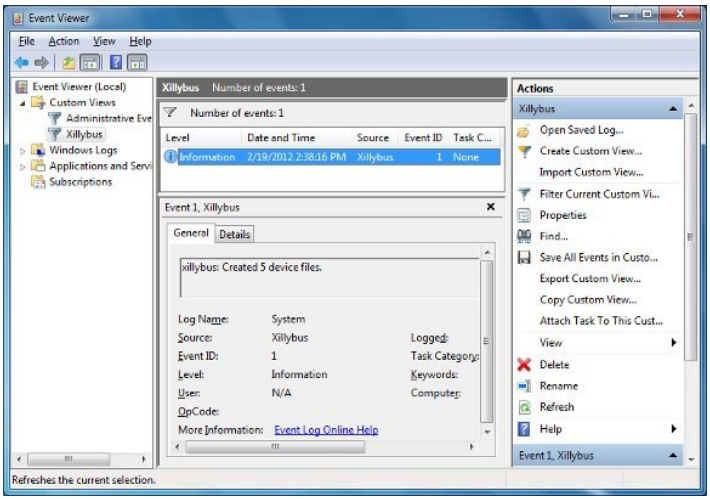


drop-down menu に Xillybus / XillyUSB エントリが見つからない場合は、driver が正しくインストールされているかどうかを確認してください。

[“OK”] をクリックすると、この custom view に名前と説明を割り当てるための別のウィンドウが開きます。これは個人的な選択の問題です。



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



上の画像は、Xillybus が正常に起動し、5 つの device files が作成されたことを通知する 1 つのメッセージを示しています。これは、FPGA に demo bundle がロードされている場合、driver のインストールが成功した直後に予想されることです。

メッセージはコンピュータの reboot では削除されません。したがって、この custom view 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 (履歴が意図的に削除されない限り)。

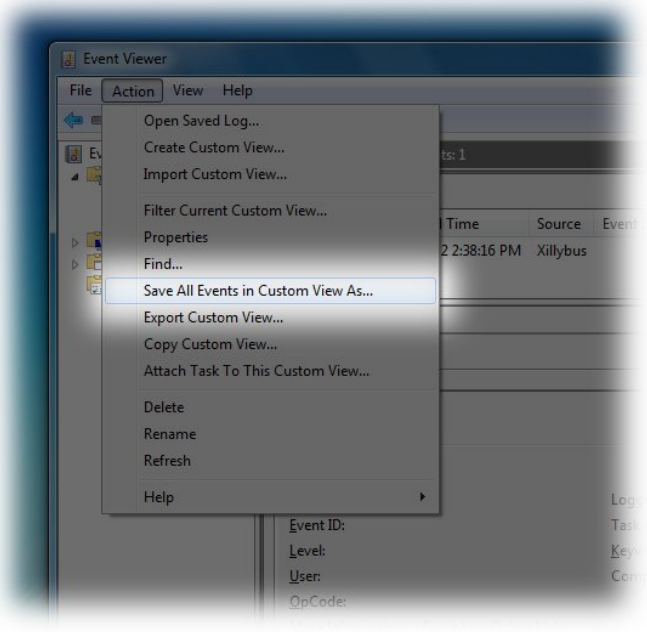
PCIe driver からのメッセージのリストとその説明は、次の場所にあります。

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

ただし、メッセージ 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。

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。サポートを求める場合は、driver からのメッセージを含むファイルを送信することをお勧めします。これらのメッセージには貴重な情報が含まれていることがよくあります。

このようなファイルを作成するには、“Action” メニュー項目を選択し、次に “Save All Events in a Custom View As...” を選択します。



ファイル選択ウィンドウが開きます。ヘルプを求める場合は、出力形式として CSV を選択します。

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、3.4 セクションを参照してください。

3.2 準備

“Hello, world” テストは、Command Prompt ウィンドウを使用して単純な command-line プログラムを実行することで構成されます。

最初のステップとして、Xillybus package for Windows をダウンロードします。これは、driver を提供するのと同じ Web ページで入手できる zip ファイルです。この zip ファイルには、これらのプログラムの source code と、実行可能な executable binaries が含まれています。

最も簡単な方法は、“Hello world” テストを実行するために executable binaries を使用することです。ただし、セクション 4.2 で詳しく説明されているように、これらのプログラムの compilation を実行することも可能です。

もう 1 つの可能性は、セクション 4.3 で説明されているように、Linux に似た作業環境を作成することです。この可能性を選択した場合は、Getting started with Xillybus on a Linux host ガイドの “Hello world” テストの手順に従ってください。

3.3 簡単な loopback テスト

2 つの Command Prompt ウィンドウを使用した loopback テストの簡単な例を次に示します。

通常の Command Prompt ウィンドウを開き、ディレクトリを Xillybus package for Windows の “precompiled-demoapps” サブディレクトリに変更します。独自の compilation の結果 (セクション 4.2 を参照) を使用するには、代わりにディレクトリを XP32_DEBUG に変更します。

この Command Prompt ウィンドウに次のように入力します。

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

これにより、“streamread” 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。

backslashes は重複していないことに注意してください。これが Cygwin で実行された場合 (通常の Command Prompt ウィンドウではなく)、代わりに `\\\\.\\xillybus_read_8` が実行されるでしょう。

次に、別の Command Prompt ウィンドウを開きます。最初の Command Prompt と同じディレクトリに移動し、次のように入力します。

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

このコマンドは、2 番目のウィンドウに入力された内容をすべて device file (つまり `\\.\xillybus_write_8`) に送信します。

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

これら 2 つのコマンドの試行中にエラー メッセージを受信した場合は、次のアクションが推奨されます。

- タイプミスがないか確認してください。
- driver がインストールされていること、および FPGA が Xillybus デバイスとして検出されていることを確認します。 Device Manager を開いて、セクション 2.1 の最後の画像と比較してください。
- 2.3 セクションで説明されているように、Event Viewer のエラーを確認します。
- 2.2 セクションで説明されているように、device files が作成されていることを確認します (`xillybus_read_8` および `xillybus_write_8` を検索)。

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。

また、streamwrite は各行で個別に動作し、ENTER が押されるまでは FPGA に何も送信しないことにも注意してください。これは、Linux 用の同じ名前のプログラムとは異なります。

loopback を間に挟んだ device files の別のペアがあります。xillybus_read_32とxillybus_write_32。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。

3.4 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 Windows 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。

このセクションでは、“hexdump” という名前のツールを使用して、FPGA の RAM のコンテンツを表示します。このツールは、Xillybus package for Windows の “unixutils” サブディレクトリにあります。あるいは、セクション 4.3 では、このツールを入手するための他のオプションが提案されています。

メモリに書き込む前に、hexdump を使用して現在の状況を観察できます。

```
> 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
```

この出力はメモリ配列の最初の 32 バイトです。hexdump は 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 \\.\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 \\.\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 を使用できる 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。これは、memory array の代わりに registers を FPGA に配置することで実現されます。これは、2 をアドレス指定する書き込み操作に応答して新しい値を受け取る register の例です。

```
reg [7:0] my_register;

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

同様に、FPGA の logic で “case” ステートメントを使用することにより、registers の値を読み取ることができます。

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 6 つの C プログラムがあります。これらのプログラムは、Xillybus package for Windows 内にあります。Xillybus package for Windows は、driver を提供する同じ Web ページからダウンロードできる zip ファイルです。

このファイル内の “precompiled-demoapps” サブディレクトリに precompiled executables があります。

source code は “demoapps” サブディレクトリにあります。これらの C プログラムは Microsoft’s Visual C++ compiler を対象としており、Microsoft の SDK の一部として無料でダウンロードできます。これらのプログラムは Visual Studio でも使用できます。

Cygwin 内でサンプル プログラムを実行することもできます (セクション 4.3 を参照)。 MinGW もこの目的に使用できます。これら 2 つのツールのいずれかを選択した場合は、Linux 用の source code を使用し、Getting started with Xillybus on a Linux host の指示に従う必要があります。 /dev/ プレフィックスの \\\\.\\ への置換に関するセクション 4.4 も参照してください。

“demoapps” サブディレクトリは次のファイルで構成されます。

- Makefile – このファイルには、プログラムの compilation の目的で “nmake” 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 ィ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 ィンターフェイスにアクセスする方法を示します。
- fifo.c – 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。
- 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 streamread.c と同じことを行いますが、winstreamread.c は標準の API の代わりに Microsoft の file I/O API を使用してデモンストレーションします。

すべてのプログラム (winstreamread.c を除く) は、Microsoft の compiler を備えた compilation を対象としているにもかかわらず、古典的な Linux スタイルで作成されています。

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、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 Windows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。

4.2 Compilation

すでに述べたように、サンプル プログラムを試すために compilation は必要ありません。 Xillybus package for Windows には、Windows コンピュータ上ですぐに実行できるファイルが含まれています。しかし明らかに、変更を加えるにはこれらのプログラムの compilation が必要です。

Microsoft Visual Studio の使用に慣れている人は、おそらくこの compiler の使用を好み、このツールの使用方法を知っているでしょう。サンプル プログラムは単純な Command Prompt アプリケーションです。

ただし、以下のガイドラインは Microsoft の software development kit (SDK) 7.1 に基づいています。これは古くてシンプルな開発キットであり、無料でダウンロードできます。以下の手順はこのソフトウェアに基づいています。主に、タスクを完了するために必要な手順が少ないためです。

Windows SDK 7.1をダウンロードしてインストールします。 "Start menu" で Program Files を開き、Microsoft Windows SDK v7.1 > Windows SDK 7.1 Command Prompt を選択します。これにより、Command Prompt ウィンドウが開きます。このウィンドウには、compilation 用に構成された複数の environment variables が含まれています。このウィンドウ内のテキストは黄色のフォントで表示されます。

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で compilation を実行するには、「"nmake"」と入力します。次のトランスクリプトが期待されます。

```
> 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、compiler を使用するために “nmake” 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。 “nmake”を使用してください。 “link” コマンドも同様で、object files および libraries で linking を実行し、executables を作成します。

executables (および object files) は、XP32_DEBUG サブディレクトリにあります。このサブディレクトリは、必要に応じて compilation プロセス中に作成されます。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 用です。ただし、executables は、64 ビット バージョンを含む Windows の以降のバージョンで動作します。

“nmake” 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、“nmake” 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、“nmake” 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、“nmake 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、“nmake” もこの新しいファイルの compilation を実行することもできます。

4.3 Linux のツールを Windows で使用する

Linux を使用するユーザーは、単純なタスクを実行するために標準の command line ツールを使用する傾向があります。これらのツールは、主に Windows を使用する人々の間ではあまり知られていません。主な理由は、残念ながら、Windows 用の command-line ツールは、すべての Linux コンピュータに存在するツールほど有用で

はないことです。

すでに述べたように、このガイドの手順の代わりに、Getting started with Xillybus on a Linux host で詳しく説明されているように “Hello world” テストを実行することができます。 Windows でこれを行うには、コンピュータ上でいくつかのツールを使用できるようにする必要があります。これを実現するための代替手段がいくつかあります。

- Gnuwin32 プロジェクトから 2 つのパッケージをダウンロードしてインストールします。 CoreutilsとUtil-Linux-NG。これら 2 つのパッケージは、“Hello world” テストのニーズをカバーします (また、このタスクに必要ではないプログラムも提供します)。これらのパッケージが Gnuwin32 の setup ツールでインストールされた場合でも、Command Prompt の execution path には変更がないことに注意してください。
- Xillybus によって提供される Windows 用のツールを使用します。 これらは、Xillybus package for Windows の “unixutils” サブディレクトリにあります。この方法で入手したツールは、“Hello world” テストに十分なように Gnuwin32 パッケージから選択されたものです。
- Cygwinをインストールします。この方法を選択するということは、Linux に似た command-line インターフェイスを提供するシステム全体をインストールすることを意味します。この種のインストールには、GNU C compiler およびソフトウェア開発用のその他のツールが含まれる場合があります。 Linux や command-line の操作に慣れている人には、これをお勧めします。

4.4 Linuxとの違い

Getting started with Xillybus on a Linux host で説明されている “Hello world” テストを Windows コンピュータで実行する場合、注意すべき相違点がいくつかあります。

最も重要な違いは、path から device files までが `/dev/` ではなく `\\.\` であることです。したがって、たとえば、Linux のガイドで `/dev/xillybus_read_8` について言及されている場合、Windows の正しいファイル名は `\\.\xillybus_read_8` です。

device file の名前には backslashes が含まれているため、状況によっては escape characters が必要になります。 backslash 自体は escape character として扱われることがよくあります。したがって、ファイル名内の backslash ごとに 2 つの backslashes が必要になります。つまり、`\\.\` は `\\\\.\\` として記述する必要があります。たとえば、Linux のガイドで `/dev/xillybus_read_8` について言及し

ている場合、状況によってはファイル名 `\\\\.\\xillybus_read_8` を使用する必要があります。

しかし、常にそうとは限りません。 Command Prompt からプログラムを実行する場合、escape characters は必要ありません。 Command Prompt は、backslash を他の character と同様に扱います。

ほとんどのプログラミング言語では、追加の backslashes が必要です。 scripts の内部では、追加の backslashes が必要になる場合があります。これは、script 内で arguments がどのように処理されるかによって異なります。

4.5 Cygwinの警告メッセージ

Cygwin の command-line インターフェイスでは追加のバックスラッシュが必要です。ただし、`\\\\.\\` プレフィックスを初めて使用する場合、Cygwin は次のような警告を表示する可能性があります。

```
$ 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}
```

この警告は無視してください。

Xillybus は Cygwin で広範囲にテストされており、上で示した device files へのアクセス方法は正しいです。通常のファイル名には、スラッシュを使用することをお勧めします。ただし、Cygwin は `//./` を `\\.\` に変換しません。したがって、backslashes の使用は必須です。

この警告を回避するには、environment variable に関する警告メッセージの提案に従ってください。

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、以下のセクション 5.3 に示すように、Linux の “dd” コマンドを使用することです。このツールは、Xillybus package for Windows 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、`\\.\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)。

`\\.\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 4.1 および 3.3 (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 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。

テストでの Cygwin の使用の詳細については、セクション 4.3 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

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。たとえば、Task Manager はこの目的に使用できます。ただし、このプログラムが表示する情報は、複数の processor cores を搭載したコンピュータ (つまり、最近のほぼすべてのコンピュータ) では誤解を招く可能性があります。たとえば、processor cores が 4 台ある場合、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? system monitoring 用のツールが異なれば、この情報は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 Windows のセクション 6.5 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 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 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、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ています。したがって、何かが間違っていると思われる場合は、関連するメッセージを検索することをお勧めします。これは、セクション 2.3 で説明されているように、event log にフィルターを適用することによって行われます。

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 event log を確認することをお勧めします。

PCIe 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。