(機械で日本語に翻訳)

# Getting started with the FPGA demo bundle for Xilinx

*Xillybus Ltd.*

www.xillybus.com

*Version 3.1*

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。

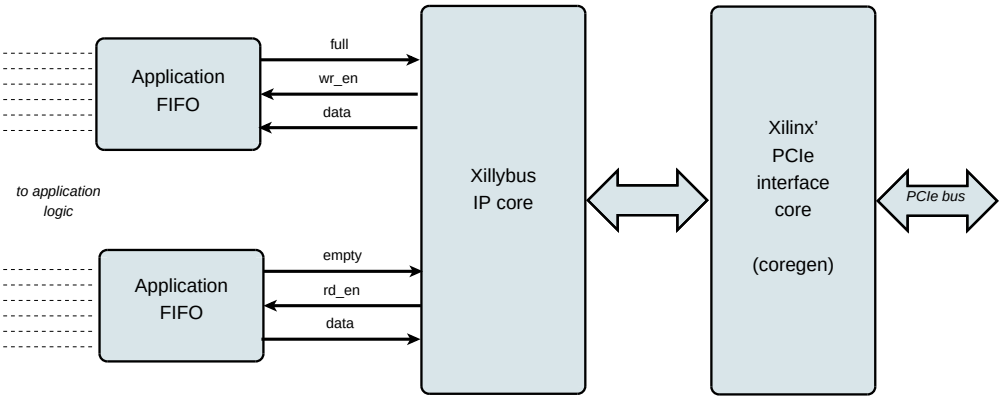
*This document has been automatically translated from English by a computer, which may result in unclear language. This document may also be slightly outdated in relation to the original.*

*If possible, please refer to the document in English.*

# 1

# 序章

Xillybus は、 FPGA と、 Linux または Microsoft Windowsを実行する host との間のデータ転送用の DMAベースのエンドツーエンド ソリューションです。 FPGA logic の設計者だけでなく、ソフトウェアのプログラマーにもシンプルで直感的なインターフェイスを提供します。




上記のように、 FPGA 上の application logic は、標準の FIFOsと対話するだけで済みます。

たとえば、図の下部の FIFO にデータを書き込むと、 Xillybus IP core は、 FIFOのもう一方の端でデータを送信できることを認識します。間もなく、 IP core は FIFOからデータを読み取り、それを hostに送信して、 userspace softwareで読み取り可能にします。データ転送メカニズムは、 FIFOと相互作用するだけの FPGAの application logic に対して透過的です。

一方、 Xillybus IP core は、 PCI Expressの Transport Layer levelを利用してデータフローを実装し、 TLP パケットを生成および受信します。下位層については、開発ツールの一部である Xilinxの公式 PCIe coreに依存しており、追加のライセンスは必

要ありません ( WebPACK エディションを使用している場合でも)。

コンピューター上のアプリケーションは、 named pipesのように動作する device files と対話します。 Xillybus IP core と driver は、 FPGAs の FIFOs と hostの関連する device files の間でデータを効率的かつ直感的に転送します。

XillyUSBでは、 MGT transceiver を使用して USB 3.0 インターフェイスを実装します。 USB 3.0 インターフェイスは、上記の PCIe インターフェイスの代わりにデータ転送に使用されます。

IP core は、オンライン Web アプリケーションを使用して、顧客の仕様に従って即座に構築されます。 streamsの数、方向、およびその他の属性は、 designの帯域幅パフォーマンス、同期、およびシンプルさの間で最適なバランスを実現するために、お客様が定義します。このガイドで説明されているように、 demo bundleで準備手順を実行した後、 http://xillybus.com/custom-ip-factoryでカスタム IP core をビルドしてダウンロードすることをお勧めします。

このガイドでは、 FPGA を Xillybus IP coreで迅速にセットアップする方法について説明します。 Xillybus IP coreは、ユーザー提供のデータ ソースとデータ コンシューマーに接続して、実際のアプリケーション シナリオ テストを行うことができます。 IP core は demo bundleに含まれており、Web サイトからダウンロードできます。

その名前にもかかわらず、 demo bundle はデモンストレーション キットではなく、有用なタスクをそのまま実行できる完全に機能する starter designです。

興味のある方は、 Xillybus host application programming guide for Linux または Xillybus host application programming guide for Windowsの付録 A に Xillybus の実装方法に関する簡単な説明があります。

# 2

# 前提条件

## 2.1 ハードウェア

Xillybus FPGA demo bundle は、ダウンロード ページにリストされているように、いくつかのボードとデバイスで動作するようにパッケージ化されています (以下のセクション 2.2 を参照してください)。

他のボードの所有者は、ピン配置に必要な変更を行い、 MGTの reference clock が適切に処理されることを確認した後、自分のハードウェアで demo bundle を実行できます。これは、経験豊富な FPGA エンジニアにとっては簡単なことです。これについてはセクション 4.3で詳しく説明します。

## 2.2 FPGA プロジェクト

Xillybus demo bundle は、 Xillybus サイトのダウンロード ページからダウンロードできます。 PCIeベースの coresの場合:

http://xillybus.com/pcie-download

XillyUSBの場合:

http://xillybus.com/usb-download

demo bundle には、単純なテストを目的とした Xillybus IP coreの特定の構成が含まれています。そのため、特定のアプリケーションではパフォーマンスが比較的低くなります。

カスタム IP cores は、 IP Core Factory Webアプリケーションを使用して構成、自動構築、およびダウンロードできます。このツールの使用については、http://xillybus.com/custom-ip-factory にアクセスしてください。

Xillybus IP coreを含むダウンロードされたバンドルは、この使用が “evaluation”という用語に合理的に一致する限り、無料で使用できます。これには、 core をエンドユーザー designsに組み込み、実際のデータとフィールド テストを実行することが含まれます。 core の使用方法に制限はありません。この使用の唯一の目的が、特定のアプリケーションに対するその機能と適合性を評価することである限りです。

## 2.3 開発ソフトウェア

Xillybusの demo bundle (および Xillybusを含む他の designs ) の implementation に推奨されるツールは、 FPGAのファミリーに応じて以下にリストされています。

**PCIeのXillybus** :

現在、コンピュータにインストールされている Vivado バージョンが、 PCIeで Xillybus を使用するのに適していることはほぼ確実です。それにもかかわらず、これらはより詳細な要件です。

- Virtex-5 FPGAsを使用する場合、 Xilinx ISE 13.1 バージョンが優先されます。段落 4.3.5を参照してください。
- Spartan-6 および Virtex-6の場合は、 Xilinx ISE 13.2 以降を使用してください。
- Gen2 インターフェースを備えた Kintex-7 および Virtex-7 ( XT/HTではないすべての Virtex-7 、および 485T も) では、 Vivado 2014.1 以降が優先ツールです。 ISE リビジョンのうち、バージョン 14.2 以降を推奨します。
- Gen3 インターフェースを持つ Virtex-7 ( 485Tを除く XT/HT ) の場合、 Vivado 2014.1 以降が推奨されます。 ISE を選択した場合、リビジョン 14.6 以降が必要です。
- Artix-7の場合、 Vivado 2014.1 以降を使用する必要があります。 ISE 14.6 以降でも問題ありません。
- Kintex / Virtex Ultrascaleの場合、 Vivado 2015.2 以降を使用する必要があります。これらのデバイスをサポートする ISE リビジョンはありません。
- Ultrascale+ FPGAs には Vivado 2017.3 以降が必要です。
- Versal APAC FPGAs には Vivado 2021.2 以降が必要です。

**XillyUSB** :

- Ultrascale+を除くすべての FPGAs では、 Vivado 2015.2 以降を使用する必要があります。
- Ultrascale+の場合、 Vivado 2018.3 以降を使用する必要があります。

このソフトウェアは、 Xilinxの Web サイト (http://www.xilinx.com) から直接ダウンロードできます。

このソフトウェアのどのエディションも適しています。 FPGA が WebPACK エディションの対象である場合、このエディションは、ライセンス料なしで無期限にダウンロードして使用できるため、好ましい選択肢となる可能性があります。

Xillybus の implementation は、 Xilinxによって提供される一部の IP cores に依存しています。すべてのソフトウェア エディションがこれらの IP coresに対応しており、追加のライセンスは必要ありません。

## 2.4 FPGA designの使用経験

design が demo bundlesのリストに表示されるボード用である場合、 FPGAで demo bundle を動作させるために FPGA design の以前の経験は必要ありません。他のボードを使用する場合、 Xilinxのツールの使用、特にピン配置と clocksの定義に関する知識が必要です。

demo bundleを最大限に活用するには、 logic design の手法を十分に理解し、 HDL 言語 (Verilog または VHDL) を習得する必要があります。それにもかかわらず、 Xillybus demo bundle は、実験するための単純な starter design を提供するため、これらを学習するための良い出発点です。

# 3

# demo bundleの implementation

## 3.1 概要

以下のすべてを読み飛ばしたい Vivado ユーザーの場合、最初の implementationを実行するための手順は次のとおりです。

- demo bundle を作業ディレクトリに解凍します。
- Vivadoを起動するか、 Vivado が既に実行されている場合は開いているプロジェクトを閉じます。
- Tools > Run Tcl Script... を選択し、 verilog/ または vhdl/ ( demo bundle内) で **xillydemo-vivado.tcl** を選択します。
- “Generate Bitstream” (または “Generate Device Image”) をクリックします。
- implementationが成功した後、 vivado/xillydemo.runs/impl_1/で bitstream ファイルを見つけます。

Xillybusの demo bundleの implementation 、および bit stream ファイルの取得には、次の 3 つの方法があります。

- バンドル内のプロジェクト ファイルをそのまま使用します。これは最も簡単な方法で、 ML506 (Virtex-5)を除く demo bundlesのリストに表示されるボードで作業する場合に適しています。
- 別の FPGAに一致するようにファイルを変更します。これは、他のボードや他の FPGAsで作業する場合に適しています。これは、 Virtex-5を使用する場合にも必要です。これについての詳細は、段落 4.3を参照してください。

- Vivado (または ISE) プロジェクトをゼロからセットアップします。 demo bundle を既存の application logicと統合する場合に必要になる可能性があります。段落 4.2の詳細。

このセクションの残りの部分では、最初の作業手順について詳しく説明します。これは、最も単純で最も一般的に選択される手順です。他の 2 つの作業手順は、最初の作業手順に基づいていますが、上記の段落で詳しく説明されている違いがあります。

**重要:**

評価バンドルは、パフォーマンスよりも単純化のために構成されています。特に高帯域幅の場合、持続的かつ継続的なデータ フローを必要とするアプリケーションでは、大幅に優れた結果が得られます。これらのシナリオでは、カスタム *IP core* を簡単に構築し、*Web* アプリケーションでダウンロードできます。

## 3.2 ファイル概要

バンドルは、これらのディレクトリのいくつかで構成されています (どのディレクトリが存在するかは、目的の FPGAによって異なります)。

- core – Xillybus IP core はここに格納されます
- instantiation templates – core 用の instantiation templates が含まれています ( Verilog および VHDL内)。
- verilog – demo bundle のプロジェクト ファイルと Verilog のソースが含まれています ( 'src' サブディレクトリ内)。
- vhdl – demo bundle のプロジェクト ファイルと VHDL のソースが含まれています ( 'src' サブディレクトリ内)。
- blockdesign – このディレクトリは、 Vivadoでサポートされている FPGAs の demo bundles にあります。 Block Design Flowに関連するファイルが含まれています。 Block Design Flow は新しい designsには推奨されないことに注意してください。
- blockplus または pcie_core (またはなし) – このディレクトリは、 ISE が使用されている場合にのみ関連するため、 Virtex および series-7 FPGAsのバンドルにのみ存在します。残りのバンドルを構築する前に、このディレクトリを処理する必要があります。 Xilinxの PCIe core ラッパーは、 FPGAのメイン プ

ロジェクトに Coregen モジュールとして直接含めることができないため、ここでビルドされます。

- vivado-essentials – Vivadoで使用する、 logic用の定義ファイルとビルド ディレクトリ。

demo bundle がダウンロードされたサイトの Web ページにリストされているように、各 demo bundle は特定のボード用であることに注意してください。別のボードが使用されている場合、または特定の構成抵抗がボードに追加または削除されている場合は、それに応じて constraints ファイルを編集する必要があります。

Vivado プロジェクトの場合、このファイルは vivado-essentials/xillydemo.xdcであり、 ISE プロジェクトの場合、 verilog/ または vhdl/の下の選択された 'src' ディレクトリにある UCF ファイルです。

また、 vhdl ディレクトリには Verilog ファイルが含まれていますが、これらのファイルを大幅に変更する必要はないことに注意してください。

Xillybusの IP core と application logic の間のインターフェースは、 xillydemo.v ファイルまたは xillydemo.vhd ファイル (それぞれの 'src' サブディレクトリ内) で行われます。これは、 Xillybus を自分のデータで試すために編集するファイルです。

## 3.3 Vivadoで bitstream ファイルを生成する

ISE ユーザー: 段落 3.4まで飛ばしてください。

Vivado は、比較的複雑な構造で多くの中間ファイルを生成するため、プロジェクトを管理することが困難になります。バンドル内のファイル構造をコンパクトに保つために、 Vivado プロジェクトを作成するために Tcl の script が提供されています。この script は、新しいサブディレクトリ "vivado"を作成し、必要に応じてこのディレクトリにファイルを追加します。

プロジェクトは、 src/ サブディレクトリ内のファイルに依存しています (これらのファイルのコピーは作成されません)。 PCIe ブロックと、 logicで使用される FIFOs は、 vivado-essentials/で定義されています。また、 Vivado は、プロジェクトのimplementation中に、このディレクトリに中間ファイルを設定します。

Vivadoを起動します。プロジェクトを開いていない状態で、好みに応じて、 Tools > Run Tcl Script... を選択し、 verilog/、 vhdl/ 、または blockdesign/ サブディレクトリで **xillydemo-vivado.tcl** を選択します。一連のイベントが 1 分以内に発生します。プロジェクトの展開の成功は、 Vivadoのウィンドウの下部にある "Tcl Console" タブを選択することで確認でき、次のように表示されることを確認します。

```
INFO: Project created: xillydemo
```

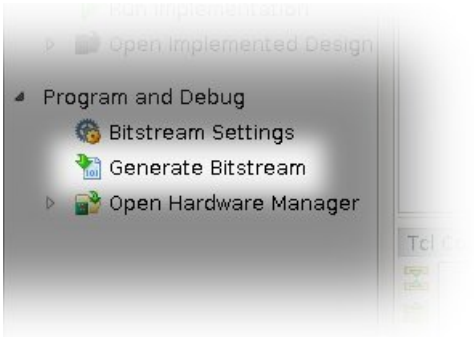
これが Tcl consoleの最後の行でない場合は、 Vivado の間違ったリビジョンが使用されている可能性が高いため、何か問題が発生しています。この段階で

Warnings が表示されますが、エラーはありません。ただし、プロジェクトが既に生成されている (つまり、 script が既に実行されている) 場合、 script を再度実行しようとすると、次のエラーが発生します。

```
ERROR: [Common 17-53] User Exception: Project already exists on disk,
       please use '-force' option to overwrite:
```

新しい “vivado” サブディレクトリが、 Tcl script.1 を含むディレクトリに作成されることに注意してください。

プロジェクトが作成されたら、 implementationを開始します。左側の Flow Navigator バーの “Generate Bitstream” (または “Generate Device Image”) をクリックします。




synthesis と implementation を起動してもよいかどうかを尋ねるポップアップ ウィンドウが表示される可能性があります – “Yes”を選択します。

Vivado は一連のプロセスを実行します。通常、これには数分かかります。いくつかの warnings が発行されましたが、いずれもクリティカルに分類されていません (ただし、一部の critical warnings は、 Tcl scriptの実行によりログに残っている可能性があります)。

bitstream の implementation が正常に完了したことを示すポップアップ ウィンドウが表示されます。次に何をするかの選択肢を与えてくれます。 “Cancel”を含め、どのオプションでも問題ありません。

bitstream ファイル、 xillydemo.bitは vivado/xillydemo.runs/impl_1/にあります。 Versal FPGAsの場合、ファイルは代わりに xillydemo.pdi です。

implementation が故障することは決してありません。ただし、言及する価値のあるエラー状態が 1 つあります。

カスタム logic が統合されている場合、 “Timing constraints weren’t met” を示すエラーが発生する可能性があります。これは、ツールが timingの要件を達成できなかったことを意味します。この場合、 design は構文的に正しいですが、特定の clock レートや I/O 要件に関して特定のパスを十分に高速にするために修正が必要です。 design をより良い timing に修正するプロセスは、しばしば *timing closure*と呼ばれます。

timing constraint の障害は一般に critical warningとして発表されるため、 Vivado は、ユーザーが FPGAの信頼できる動作を保証しない bitstream ファイルを作成することを妨げません。このような bitstreamの作成を防ぐために、 route の実行の最後に自動的に実行される小さな Tcl script、つまり “showstopper.tcl”によって、 timing の障害がエラーに変わります。この安全対策をオフにするには、 Flow Navigatorの “Project Manager” の下にある “Project Settings” をクリックします。 “Implementation” ボタンを選択し、 “route_design”の設定まで下にスクロールします。次に、 tcl.postから showstopper.tcl を取り外します。

Vivado のユーザーは、次のセクションを飛ばして 3.6の段落に直接進むことができます。

## 3.4 Xilinxの PCIe IP coreのセットアップ

この部分は、 Spartan-6以外の FPGAs の ISE ツールチェーンにのみ関連します。 Vivado を使用する場合は、段落 3.3を参照してください。 Spartan-6 FPGAs で作業している人は、パラグラフ 3.5に直接ジャンプできます。

PCIe 用の Coregen IP core のやや特殊な構成では、 implementation プロジェクトに XCO ファイルを含めることはできませんが、代わりに、 core 生成ソフトウェアが含める Verilog ファイルを作成します。これは、 Virtex-5、 Virtex-6 、および series-7を使用している場合にのみ当てはまります。

Virtex-5 FPGAs には、 XCO ファイルの特別な処理が必要です。段落 4.3.5を参照してください。

blockplus ディレクトリまたは pcie_core ディレクトリ (バンドル内にある) でプロジェクト ファイル (.xise) を見つけ、それをダブルクリックして ISEを開きます。 “Design Utilities”の下で、 “Regenerate all cores” をクリックし、プロセスが完了するまで待ちます。次に、 ISEを閉じます。これ以上のアクションは必要ありません。

この手順では、 Xilinx PCIe coreのラッパーである Verilog ファイルのセットを生成します。これらのファイルは、 demo bundleの bitstreamを作成するプロジェクトで使用されます。優先言語として Verilog または VHDL を選択しても、ファイルは

Verilogで生成されます。

これらの Verilog ファイルをメイン プロジェクトに手動で含める必要はありませんが、プロジェクト全体の implementation を試行する前に、これらのファイルを一度生成する必要があります。メイン プロジェクトの implementation を繰り返す前であっても、この手順を繰り返す必要はありません。

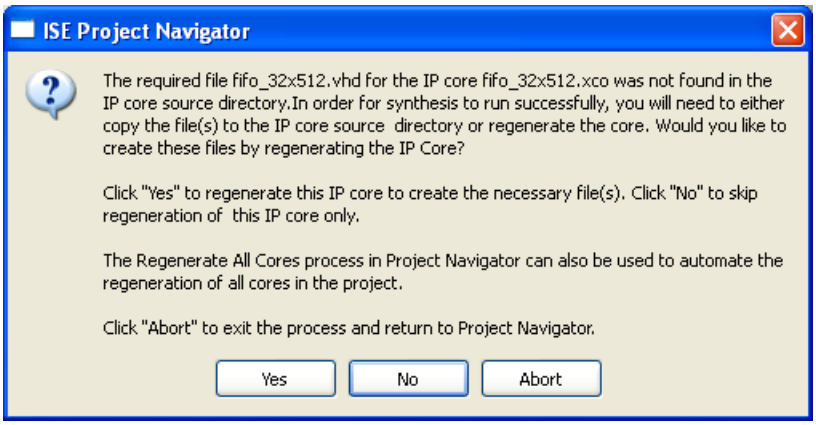
## 3.5 ISE スイートでのビット ファイルの生成

Virtex または series-7 ファミリに属する FPGA を使用する場合は、上記の 3.4 の説明に従って、 PCIe ラッパーを準備したことを確認してください。

好みに応じて、 'verilog' サブディレクトリまたは 'vhdl' サブディレクトリにある 'xillydemo.xise' ファイルをダブルクリックします。 ISE が実行を開始し、正しい設定でプロジェクトを開きます。 ISE は、ファイルが見つからないと文句を言うべきではありません。そうで、 FPGA ファミリが series-7 または任意の Virtex ファミリの 1 つである場合、 PCIe ラッパーが段落 3.4で述べたように準備されていない可能性があります。

“Generate Programming File” をクリックして、 implementationを起動します。

最初の implementationの間に、2 つまたは 3 つの Xilinx Coregen IP coresを再生成する必要があります。このプロセスには時間がかかりますが、幸いなことに一度だけです。次のようなポップアップ ウィンドウが 2 3 回表示されます。




これらすべてで “Yes” をクリックします。

この手順では、いくつかの warnings が生成されます ( FPGA プロジェクトの implementation は常に生成されます) が、エラーは表示されません。プロセスが終了すると、 bitfile は xillydemo.bitとして検出されます。

log の次の文 (末尾付近) を探して、 **timing constraints が達成されたことを常に確認します。**

```
All constraints were met.
```

これは、特にプロジェクトに変更を加えた後は非常に重要です。 constraints が達成されない場合でも、ツールは bitfileを生成しますが、 FPGA は予期しない方法で動作する可能性があります (おそらく、許容温度範囲内の温度変化の結果として)。

同様のメッセージが、 ISEの design の概要に記載されています。

timing constraints を達成できないのは、 bus_clkの速度に耐えられるほど高速でない logic を追加した結果である可能性があります。しかし、明らかな理由もなく障害が発生した場合は、 logic コンポーネントを FPGAの logic fabricに配置しようとしたときに、 Xilinxのツールが初期の推測を誤っており、後で実行される最適化アルゴリズムがこれを修正できなかった可能性があります。

後者のケースは、 placer cost table figure を変更することで修正できます (これは、最初の配置のランダム性に対する seed です)。 ISE Project Navigator 内の Processes ペインで、 “Map” を右クリックし、 “Process Properties...”を選択します。 Property display level が “Advanced” であることを確認し、 “Starting Placer Cost Table” をまだ試していない他の番号に変更します。この数の大きさには意味がありません。その後、 “Generate Programming File”で再起動します。

**重要:**

一部の *ISE* バージョン、特に *ISE 14.2*では、 *verilog/* ディレクトリでのビルドが失敗し、 *ERROR:HDLCompiler:687 - “C:/try/xillybus-eval-kintex7-1.1/verilog/src/fifo_32x512_synth.v” Line 54: Illegal redeclaration of module fifo_32x512.* (または類似) というエラーが表示される場合があります。これは、 *Xilinx*のツールの *bug* が原因です。これを回避するには、 *src/* ディレクトリ内の *fifo_8x2048_synth.v* と *fifo_32x512_synth.v* を削除し、 *“Generate Programming File”*を再起動します。いくつかの *warnings* は、ツールがこれらのファイルを見つけられないことを示しますが、それでも *implementation* は適切に実行されるはずです。

## 3.6 bitfileのロード

開発の初期段階では、 JTAG経由で FPGA をロードすることをお勧めします。ほとんどのボードでは、 Vivado を実行するコンピュータとボードのパネルの USB コ

ネクタを接続する単純な USB ケーブルで十分です。 ISEを使用している場合は、bitfileのロードに iMPACT を使用します。

JTAG経由で FPGA をロードする方法については、ボードの説明書を参照してください。

XillyUSB プロジェクトの場合、 USB インターフェイスが動作中のコンピューターに接続されている場合でも、 FPGA はいつでもロードおよび再ロードできます。

PCIe プロジェクトでは、コンピューターの電源を入れる前に、 FPGA に bitfile をロードする必要があります。コンピューターは、電源を入れたときに PCIe 周辺機器が適切な状態にあることを期待しており、その後の予期せぬ事態に耐えられない場合があります。

したがって、 host が動作している限り、 FPGA をリロードし**ない**でください。 PCIe 仕様では hotpluggingのサポートが必要ですが、マザーボードは通常、 PCIe カードが消えてから再び現れることを想定していません。したがって、一部のマザーボードは正しく応答しない場合があります。それでも、一部のマザーボードでは、オペレーティング システムの実行中に FPGAをリロードできます。

Xillybusの driver は hotpluggingに正常に応答するように設計されていますが、コンピュータの一般的な安定性を保証するものは何もありません。これについては、次のページで説明しています。

http://xillybus.com/doc/hot-reconfiguration

FPGA の電源を入れ、コンピューターの電源を入れると同時に flash memory からロードする場合、 BIOS が busをスキャンするときに PCIe デバイスが存在するように、 FPGA が十分に速くロードされることを確認することが不可欠です。

# 4

# 変更

このセクションは、このドキュメントで概説されている Verilog / VHDL 手順に関連しています。 (推奨されていませんが) Block Design Flow を選択したユーザーは、The guide to Xillybus Block Design Flow for non-HDL users (deprecated)の指示に従う必要があります。

## 4.1 カスタム logicとの統合

Xillybus demo bundle は、 application logicと簡単に統合できるように構築されています。データを接続する場所は、 xillydemo.v または xillydemo.vhd ファイルです (優先言語によって異なります)。バンドル内の他のすべての HDL ファイルは、 Xillybus IP core を使用して host (Linux または Windows) と FPGAの間でデータを転送する目的で無視できます。

カスタム logic designs を含む追加の HDL ファイルは、段落 3.5 または 3.3で説明されているように準備されたプロジェクトに追加され、 “Generate Programming File” または “Generate Bitstream”をクリックして再構築されます。初期展開の他の手順を繰り返す必要がないため、 logic の開発サイクルは非常に迅速かつ単純です。

Xillybus IP core をカスタム application logicに接続する場合、少なくとも最初の段階では、 FIFOsを介してのみ Xillybus IP core と対話し、 logicでそれらの動作を模倣しようとしないことを強くお勧めします。

これに対する例外は、メモリまたは register arrays を Xillybusに接続する場合です。この場合、 xillydemo モジュールに示されている例に従う必要があります。

xillydemo モジュールでは、 FIFOs を使用して data loopbackを実行します。つまり、 host から到着したデータは host に送り返されます。 FIFOの両側が Xillybus IP coreに接続されているため、 core はデータのソースとコンシューマの両方になりま

す。

実際の使用シナリオでは、 FIFOの一方の端のみが Xillybus IP coreに接続され、もう一方の端はデータを供給または消費する application logicに接続されます。

xillydemo モジュールで使用される FIFOs は、両側が Xillybusのメイン clockによって駆動されるため、両側に 1 つの共通 clock のみで動作します。実際のアプリケーションでは、 bus_clk以外の clock でデータ ソースとデータ コンシューマーを駆動できるように、読み取りと書き込み用に個別の clocks を持つ FIFOs に置き換えることが望ましい場合があります。こうすることで、 FIFOs は仲介者としてだけでなく、適切な clock domain 交差にも役立ちます。

Xillybus IP core は、 FPGA から hostへの streams に対して ( First Word Fall Through, FWFTとは対照的に)単純な FIFO インターフェースを想定していることに注意してください。

次のドキュメントは、カスタム logicの統合に関連しています。

- logic design用の API ： Xillybus FPGA designer's guide
- Getting started with Xillybus on a Linux host
- Getting started with Xillybus on a Windows host
- Xillybus host application programming guide for Linux
- Xillybus host application programming guide for Windows
- The guide to defining a custom Xillybus IP core

## 4.2 カスタム プロジェクトへの組み込み

必要に応じて、 Xillybus IP core を既存の Vivado / ISE プロジェクトに含めるか、新しいプロジェクトを最初から作成することができます。

プロジェクトがまだ存在しない場合は、新しいプロジェクトを開始し、好みの HDL 言語と目的の FPGAに基づいて設定します。

Xillybus IP core を Vivado プロジェクトに含めるには、カスタム プロジェクトのソース ファイルと設定を反映するように xillydemo-vivado.tcl を編集し、この scriptを実行して新しいプロジェクトを作成することをお勧めします。

ISE プロジェクトに Xillybus IP core を含めるには:

- Virtex-5/6 または series-7 ファミリーに属する FPGAs を使用する場合、 PCIe ラッパー ファイルは、段落 3.4 (および Virtex-5 FPGAsの段落 4.3.5 ) で説明

されているように、個別に生成する必要があります。生成された Verilog ファイルは、カスタム プロジェクトに追加する必要があります ( XCO ファイルではありません)。

- 2 つの src/ サブディレクトリ (言語設定に応じて) のいずれかにあるすべてのファイルをプロジェクトに追加します。
- Macro search pathにディレクトリを追加します。 process メニューの “Implementation”で、 “Translate” を右クリックし、 “Process Properties...”を選択します。 ’core’ サブディレクトリを Macro Search Path プロパティに追加します (右端のボタンで参照)。このプロパティを設定しないと、 Translate ステージが implementation中に失敗します。これは、 xillybus_core.ngc ファイルが見つからないためです。
- xillydemo モジュールがプロジェクトの top level module でない場合は、そのポートを top levelに接続します。
- Xillybus IP core をカスタム application logicに接続するには、 xillydemo モジュールを編集して、既存の application logic を目的のものに置き換えます。

## 4.3 他のボードを使用する

### 4.3.1 全般的

demo bundlesのリストにないボードを使用する場合は、バンドルに若干の変更が必要です。

core はいくつかの GPIO LED 出力を生成します。空きがあればボード上の LEDs に接続することをお勧めします。

### 4.3.2 PCIeに Xillybus を使用する

購入したほとんどのボードには、 FPGA designの独自の例があり、そのボードで PCIe インターフェイスがどのように使用されているかを示しています。多くの場合、目的のボードの XDC / UCF ファイルで関連するピン割り当てを見つけ、ピンの名前を Xillybusの XDC / UCF ファイルで使用されている名前に変更するのが最も簡単です。次に、 Xillybusのプロジェクトで使用される XDC / UCF ファイルの関連する行を置き換えることができます。

ピンの配置方法の詳細は以下のとおりです。

最も一般的な間違いは、 PCIe busの reference clock であることに注意してください。同じ周波数の clock を接続しても機能しません: マザーボードの clock と他の clock のわずかな周波数の違いにより、 transceiver が散発的にロックを失い、信頼性の低い通信が発生し、 FPGA を PCIe デバイスとして検出できない可能性があります。

Xillybus core は Xilinxの PCIe coreに基づいているため、 Xilinxのユーザー ガイドは、 PCIeの physical layerに関連する考慮事項の有効な情報源です。

### 4.3.3 Spartan-6 PCIe ボードの操作

Spartan-6の場合、 Xillybus core は PCIe bus ポートを介して host と接続します。このポートは、次の 7 本の物理ワイヤで構成されています。

- PCIE_250M_P と PCIE_250M_Nという名前の reference clock用の differential wires のペア: PCIe bus clockから派生した 125 MHz の周波数を持つ clock ( netの名前にもかかわらず) は、これらのワイヤで期待されます。別の clock が適用される場合、 Xilinx PCIe Coregen core (バンドル内の pcie.xco によって定義される) は、実際の clock 周波数を期待するように再構成する必要があります。さらに、 TS_PCIE_CLK 仕様に変更が反映されるように、 timing constraint を更新する必要があります。
- PCIE_PERST_B_LS上の hostの master bus reset
- シリアルデータ入力ペア、 PCIE_RX0_P および PCIE_RX0_N
- シリアルデータ出力ペア、 PCIE_TX0_P および PCIE_TX0_N

これらのピンの割り当ては、ボードの配線に従って設定されます。

### 4.3.4 Virtex-6 PCIe ボードの操作

Virtex-6 の場合、配線は似ています。

- PCIE_REFCLK_P および PCIE_REFCLK_Nという名前の、 reference clock用の differential wires のペア: PCIe bus clockから派生した 250 MHzの周波数を持つ clock がこれらのワイヤで期待されます。別の clock が適用される場合、 Xilinx PCIe Coregen core (バンドル内の pcie_v6_4x.xco によって定義される) は、実際の clock 周波数を期待するように再構成する必要があります。このような変更には、 constraintsの変更も含まれる場合があります。 Coregenによって生成された UCF ファイルの例を参照してください。

- PCIE_PERST_B_LS上の hostの master bus reset
- シリアル データ入力ベクトル ペア、 PCIE_RX_P および PCIE_RX_N (各 4 ワイヤ)
- シリアル データ出力ベクトル ペア、 PCIE_TX_P および PCIE_TX_N (各 4 ワイヤ)

ピンの割り当ては、 transceiver logicを配置することによって暗黙的に行われます。 UCF ファイルで GTX の配置を定義する constraints は、特定の pinoutを強制します。同様に、 reference clockのピンの配置は、 clock buffer (pcieclk_ibuf) の位置を制約することによって暗黙的に設定されます。 Xillybusのバンドル内の UCF ファイルにはガイド コメントが含まれています。

UCF ファイルは、これらのピン配置が目的のボードのものと一致するように編集する必要があります。

### 4.3.5 Virtex-5 PCIe ボードの操作

Virtex-5 ファミリには 2 つのデバイス グループがあり、それぞれにわずかに異なる PCIe インターフェイスが必要です。これを簡単に処理するには、 'blockplus' サブディレクトリに 2 つの異なる XCO ファイルがあり、そのうちの 1 つだけを使用する必要があります。

したがって、 PCIe coreをビルドする前に、そのサブディレクトリ内のファイルの名前を次のように変更する必要があります。

- Virtex-5 LX または Virtex-5 SXの場合: pcie_v5_gtp.xco を pcie_v5.xco に名前変更
- Virtex-5 FX または Virtex-5 TXの場合: pcie_v5_gtx.xco を pcie_v5.xco に名前変更

**重要:**

*PCIe Block Plus generator* のバージョンは *1.14*である必要があり、 *1.15*ではありません。 *ISE 13.1* にはこの目的に適したバージョンがありますが、 *ISE 13.2* に含まれているバージョンでは問題のあるコードが生成されます。

implementation全体に ISE 13.1 以外のバージョンが必要な場合は、正しいバージョンの PCIe Block Plus ( ISE 13.1に含まれる) を使用して Verilog ファイルを生成する

ことができます。プロジェクト全体の implementation は、 ISEの優先バージョンで実行できます。

UCF ファイルには、ピンの設定方法に関するガイド コメントがあります。 PCIe ピンの配置は暗黙的であり、 GTP/GTX コンポーネントの位置で constraint によって強制されます。

100 MHz の周波数を持つ clock は、 PCIE_REFCLK ワイヤ ペアで期待されます。別の clock が適用される場合、 Xilinx PCIe Coregen core ( pcie_v5.xcoで定義) は、実際の clock 周波数を期待するように再構成する必要があります。さらに、 TS_MGTCLK 仕様が変更を反映するように、 timing constraint を更新する必要があります。

### 4.3.6 Kintex-7、 Virtex-7 、および Artix-7 ボードでの作業 (PCIe)

series-7 ファミリのすべての FPGAs は、同じ PCIe インターフェイスを備えています。

- PCIE_REFCLK_P および PCIE_REFCLK_Nという名前の、 reference clock用の differential wires のペア: PCIe bus clock から派生した (または直接接続された) 100 MHzの周波数を持つ clock は、これらのワイヤで予期されます。

  別の clock が適用される場合、 PCIe ブロック ( pcie_k7_vivado.xci または demo bundleで同様に定義される) は、実際の clock 周波数を期待するように再構成する必要があります。このファイルは、プロジェクトのソース リストに表示されます。このような変更には、 timing constraintsの変更も含まれる場合があります。 Xilinxのツールによって生成された XCF ファイルの例を参照してください。

  ISE を使用する場合、 Xilinxの PCIe core は pcie_k7_8x.xcoなどで定義されます。 XDCではなく、 UCF ファイルを調整する必要がある場合があります。

- PCIE_PERST_B_LS 上の hostの master bus reset

- シリアル データ入力ベクトル ペア、 PCIE_RX_P および PCIE_RX_N (それぞれ 8 または 4 ワイヤ)

- シリアル データ出力ベクトル ペア、 PCIE_TX_P および PCIE_TX_N (それぞれ 8 または 4 ワイヤ)

ピンの割り当ては、 transceiver logicを配置することによって暗黙的に行われます。 UCF/XDC ファイルで GTX の配置を定義する constraints は、特定の pinoutを強制します。同様に、 reference clockのピンの配置は、 clock buffer (pcieclk_ibuf) の位

置を制約することによって暗黙的に設定されます。 Xillybusの demo bundle の UCF / XDC ファイルには、ガイド コメントが含まれています。

UCF/ XDC ファイルは、これらのピン配置が目的のボードのものと一致するように編集する必要があります。

### 4.3.7 Ultrascale および Ultrascale+ ボードの操作 (PCIe)

これらの FPGAs はすべて、同じ PCIe インターフェイスを備えています。

- PCIE_REFCLK_P と PCIE_REFCLK_Nという名前の reference clock用の differential wires のペア: PCIe busの clockに直接接続されている 100 MHzの周波数を持つ clock 。

  別の clock が適用される場合、 PCIe ブロック ( pcie_ku_vivado.xci または demo bundleで同様に定義される) は、実際の clock 周波数を期待するように再構成する必要があり、この clock の timing constraint は xillydemo.xdcで更新する必要があります。これらのファイルは、プロジェクトのソース リストに表示されます。

- PCIE_PERST_B_LS 上の hostの master bus reset

- シリアル データ入力ベクトル ペア、 PCIE_RX_P および PCIE_RX_N (それぞれ 8 または 4 ワイヤ)

- シリアル データ出力ベクトル ペア、 PCIE_TX_P および PCIE_TX_N (それぞれ 8 または 4 ワイヤ)

ピンの割り当ては、 transceiver logicを配置することによって暗黙的に行われます。 XDC ファイルで GTX の配置を定義する constraints は、特定の pinoutを強制します。同様に、 reference clockのピンの配置は、 clock buffer (pcieclk_ibuf) の位置を制約することによって暗黙的に設定されます。 Xillybusの demo bundle の XDC ファイルには、ガイド コメントが含まれています。

XDC ファイルは、これらのピン配置が目的のボードのものと一致するように編集する必要があります。

### 4.3.8 Versal ACAP ボードの操作 (PCIe)

Versal FPGAsでの PCIe の実装については、 Xillybusの tutorial page を一読することをお勧めします。

Xillybusの PCIe インターフェイスの実装は、 CIPS内にある CPM ブロックではなく、 Versal ACAP Integrated Block for PCI Express IP に依存しています。

これらの FPGAs には、次の PCIe インターフェイスがあります。

- PCIE_REFCLK_P と PCIE_REFCLK_Nという名前の reference clock用の differential wires のペア: PCIe busの clockに直接接続されている 100 MHzの周波数を持つ clock 。

  別の clock が適用される場合、 pcie_versal_xxl block design内にある pcie_block ブロック内の PCIe controller は、実際の clock 周波数を予期するように再構成する必要があります。

- シリアル データ入力ベクトル ペア、 PCIE_RX_P および PCIE_RX_N (各 8 ワイヤ)

- シリアル データ出力ベクトル ペア、 PCIE_TX_P および PCIE_TX_N (各 8 ワイヤ)

hostの bus reset は、 PMC MIO 38に直接接続されています。この信号に別の MIO pin を使用する場合、 CIPS IP をそれに応じて構成する必要があります (詳細については、 Xillybusの tutorial page を参照してください)。

他のピン割り当ては、 transceiver logicを配置することによって暗黙的に行われます。 XDC ファイルで GTX の配置を定義する constraints は、特定の pinoutを強制します。同様に、 reference clockのピンの配置は、 clock buffer (IBUFDS_GTE5) の位置を制約することによって暗黙的に設定されます。

XDC ファイルは、これらのピン配置が目的のボードのものと一致するように編集する必要があります。

### 4.3.9 XillyUSBの操作

XillyUSB は、 SFP+ インターフェイスを持つ他のボードで使用できます。その場合、 designの constraints を、 SFP+ コネクタに配線された MGT を使用するように設定するだけです。

ボードは、 MGT用に低 jitter を備えた 125 MHz reference clock も提供する必要があります。 USB 仕様の要件にもかかわらず、 Spread Spectrum Clocking (SSC) を有効にしないでください (そのようなオプションが存在する場合)。 SSC reference clock が使用されている場合、 MGT は受信信号を適切にロックしません。

カスタム ボードの場合は、 SFP+ コネクタのピンが FPGAの MGTに直接接続されているため、 sfp2usb モジュールの回路図を参照することをお勧めします。

sfp2usb モジュールで行った SSRX ワイヤの交換はオプションであり、 PCB designを簡素化する場合にのみ推奨されます。

必要に応じて SSTX ワイヤを交換することもできますが、 *_frontend.v ファイルを編集して送信ビットの極性を逆にし、ワイヤ ペアの交換を補正する必要があります。 USB の仕様では、極性を入れ替えても link partners が正しく動作する必要があるため、この編集がなくても USB 接続が正しく動作する可能性が高いことに注意してください。ただし、これに依存することはお勧めしません。

## 4.4 PCIe lanesの数を示すためのPRSNT ピン

PCIe 仕様によると、 PCIe コネクタには 1 つまたは複数のピンがあり、 PCIe slotの周辺機器の存在と lanesの数を示しています。これらは PRSNT ピンです。ほとんどの開発ボードには、これらのピンのおかげで、 host が通知される lanes の数を調整するための DIP switches があります。

これらのピンの典型的なデフォルト設定は、ボードで可能な lanes の最大数です。この設定は、実際に使用されている lanes の数が少ない場合でも、通常は機能します。これは、 host とペリフェラル間の最初のネゴシエーションにより、 lanesの実際の数が正しく検出されるためです。このネゴシエーションは、 PCIe 仕様で必要です。

これらの DIP switchesの設定方法については、ボードのリファレンス マニュアルを参照してください。一部の hosts は誤った設定の結果として lanes を無視する可能性があるため、これらの DIP switches を実際に使用されるよりも少ない lanes に設定しないことが重要です。

## 4.5 PCIe lanes および/または link speedの数の変更

### 4.5.1 序章

**重要:**

*link*のパラメータを変更すると、 *timing constraints*の調整が必要になる場合があります。この問題に注意を払わないと、 *PCIe link* が動作する可能性がありますが、信頼性が低くなります。
変更を行った後は、必要に応じて *timing constraints*を適切に調整してください。このトピックについては、以下で詳しく説明します。

Xillybusの FPGA demo bundles は通常、目的のボードで使用可能な lanes の最大数

と、 2.5 GT/s (Gen1) の link speed に設定されます。

これは、 FPGA ボードがマザーボードの PCIe コネクタに適合する場合、すべての lanes が hostとの接続に使用されることが期待できるからです。一方、ほとんどの場合、これらの lanes によって達成される帯域幅は、 2.5 GT/sを使用した場合でも、 Xillybus IP core が使用する可能性がある帯域幅よりも高いため、より高い link speedを設定しても意味がありません。

PCIe 仕様では、関連するすべての bus コンポーネントの速度を下げるためのフォールバック機能が必要であるため、 2.5 GT/s を選択すると、すべてのマザーボードで均一な動作が保証されます。

ただし、 lanes とその link speedの数を変更したいことがよくあります。特に、カスタム ボードで Xillybus IP core を使用している場合はそうです。 lanes を減らして link speed を増やすことは、一般的な要件です。

Xillybus IP core は、 PCIe busとの低レベル インターフェイスを Xilinxの PCIe ブロックに依存しています。したがって、 Xilinxの PCIe ブロックが正常に動作する限り、 lanes または link speedの数に関係なく、 IP core は正常に動作します。

PCIe ブロックが少数の lanesで構成され、 link speedと組み合わされて、その帯域幅能力が Xillybus IP coreよりも低くなる場合でも、適切に動作します。この場合、Xillybusの streams によって提供される総帯域幅は、 PCIe ブロックの設定によって課される帯域幅制限とほぼ等しくなります。何がボトルネックになるのかという問題です。

### 4.5.2 作業手順

Versal FPGAsについては、セクション 4.5.6を参照してください。以下の情報は、他のすべての FPGAsに関連しています。

原則として、 lanes および/または link speed の数を変更するには、必要に応じて PCIe ブロックの構成を変更する必要があります。

ただし、注意すべきいくつかの問題があります。

- この変更は、 PCIe ブロックの他のパラメーターに影響を与え、正しく動作しなくなる可能性があります。特に、 Xilinxの GUI ツールには注意すべきバグがあります。詳しくは以下をご覧ください。
- この変更により、 PCIe ブロック ( PIPE clocks) を駆動する clocks の周波数が変更される可能性があるため、 timing constraintsの変更が必要になります。

- 上記の clock 周波数の変更には、 PCIe ブロック (該当する場合は pipe_clock モジュールの instantiation ) をサポートする Verilog コードの変更も必要になる場合があります。そうしないと、 PCIe ブロックが機能しなくなります。

したがって、段階は次のようになります。

1. アクティブなプロジェクトで XCO または XCI ファイルのコピーを作成します (つまり、 Vivado または ISE が必要に応じて IP をアップグレードした後)。これにより、後で変更を diff ツールと比較し、不要な変更が発生した場合にそれを見つけることができます。
2. 構成のために Vivado (または ISE) の PCIe ブロックの IP を開きます。
3. (AXI) interface widthを変更しないように注意しながら、 lanes および/または Maximum Link Speed の数を必要に応じて変更します。タスクに適した lanes と link speed の組み合わせを選択することで、 (AXI) Interface Frequency の変更を回避できる場合は、それが望ましいです。
4. 必要な変更を行った後、 Vendor ID と Device ID が変更されていないことを確認します ( Subsystem のどちらも変更されていません)。 Vivado の一部のリビジョンでは、関連のない変更の結果として一部のパラメーターがデフォルトにリセットされる場合があります (これはバグです)。
5. 変更を確認します (通常、ダイアログ ボックスの下部にある “OK” をクリックします)。確認後に提案された場合は、出力ファイルを生成する必要はありません。
6. 更新された XCO または XCI ファイルをテキストの diff ツールと比較し、関連するパラメーターのみが変更されていることを確認します。これについては、以下で詳しく説明します。
7. xillybus.v と xillydemo.v/.vhdの PCIE_* の信号ベクトル幅を調整して、 lanesの新しい数を反映するようにします。
8. 以下の 4.5.3 の説明に従って、必要に応じて PIPE clock モジュールの instantiation を調整します。
9. 以下の 4.5.4 で説明されているように、必要に応じて timing constraints を調整します。
10. 段落 4.5.5で説明されているように、 PIPE clock モジュールを更新します。

Ultrascale 以降で作業する場合、最後の 3 つの手順は必要ありません。

新しい XCI ファイルと古い XCI ファイルを比較する場合、 `PARAM_VALUE.Device_ID` は誤って変更されることが多いため、特に注意する必要があります。

XCI ファイルのパラメータの違いは、必要なものと一致する必要があります。これは、 Vivadoで行われた変更に従って変更が許容される可能性のあるパラメーターの短いリストです。 Vivado の異なるリビジョン (したがって、 PCIe ブロックの異なるリビジョン) は、異なる XML パラメータを持つ PCIe ブロックの属性を表している可能性があるため、パラメータの名前には多少の注意を払う必要があります。

- lanesの数に関連:
  - `PARAM_VALUE.Maximum_Link_Width`
  - `MODELPARAM_VALUE.max_lnk_wdt`
- link speedに関連:
  - `PARAM_VALUE.Link_Speed`
  - `PARAM_VALUE.Trgt_Link_Speed`
  - `MODELPARAM_VALUE.c_gen1`
  - `MODELPARAM_VALUE.max_lnk_spd`
- インターフェイス周波数に関連しています。これらのパラメータの変更は、段落 4.5.3 および 4.5.4 の段階が必要であることを強く示しています。
  - `PARAM_VALUE.User_Clk_Freq`
  - `MODELPARAM_VALUE.pci_exp_int_freq`

### 4.5.3 PIPE 周波数は変更されましたか?

Ultrascale FPGAs 以降を使用する場合、 PCIe ブロックが IP 自体の不可欠な部分として timing constraints を提供するため、以下の考慮事項とアクションは不要です。 PIPE モジュールについても同様です。

他の FPGA ファミリについては、 PIPE clock の設定が正しいことを次のように確認することが重要です。

**変更後に** PCIe ブロックのサンプル プロジェクトを生成し、そのプロジェクトの synthesis を実行します。 Vivadoでは、これは通常、プロジェクトのソース階層で PCIe ブロックを右クリックし、 “Open IP Example Design...”を選択することによって行われます。 designの場所を選択し、それが生成されたら、左側の列で “Run Synthesis” をクリックして synthesis を起動します。

次に、 synthesis report で PIPE clock モジュールの instantiation parameters を取得します ( Vivadoでは、 pcie_example/pcie_example.runs/synth_1/runme.logのようなものとして見つかります)。このレポートで、次のようなセグメントを検索します。

```
INFO: [Synth 8-638] synthesizing module 'example_pipe_clock' [...]
Parameter PCIE_ASYNC_EN bound to: FALSE - type: string
Parameter PCIE_TXBUF_EN bound to: FALSE - type: string
Parameter PCIE_CLK_SHARING_EN bound to: FALSE - type: string
Parameter PCIE_LANE bound to: 4 - type: integer
Parameter PCIE_LINK_SPEED bound to: 3 - type: integer
Parameter PCIE_REFCLK_FREQ bound to: 0 - type: integer
Parameter PCIE_USERCLK1_FREQ bound to: 4 - type: integer
Parameter PCIE_USERCLK2_FREQ bound to: 4 - type: integer
Parameter PCIE_OOBCLK_MODE bound to: 1 - type: integer
Parameter PCIE_DEBUG_MODE bound to: 0 - type: integer
```

このレポートのパラメーターは、 xillybus.vに表示される pipe_clockの instantiation のパラメーターと一致する必要があります。これは次の形式です。

```
pcie_[...]_pipe_clock #
   (
    .PCIE_ASYNC_EN                  ( "FALSE" ),
    .PCIE_TXBUF_EN                  ( "FALSE" ),
    .PCIE_LANE                      ( 6'h08 ),
    .PCIE_LINK_SPEED               ( 3 ),
    .PCIE_REFCLK_FREQ               ( 0 ),
    .PCIE_USERCLK1_FREQ             ( 4 ),
    .PCIE_USERCLK2_FREQ             ( 4 ),
    .PCIE_DEBUG_MODE                ( 0 )
    )
   pipe_clock
     (
       [ ... ]
      );
```

比較する 3 つのパラメーターは `PCIE_LINK_SPEED`、 `PCIE_USERCLK1_FREQ` 、および `PCIE_USERCLK2_FREQ`であり、一致する必要があります。一致する場合 (例に示すように)、 timing constraintsを含め、すべてが正しく設定されています。そうでない場合は、次の 2 つのアクションを実行する必要があります。

- xillybus.v の instantiation parameters は、サンプル プロジェクトの synthesis reportと一致するように更新する必要があります。

- timing constraints はサンプル プロジェクトに合わせて調整する必要があります。これを正しく行わないとすぐに問題が発生するわけではありませんが、designの信頼性に影響を与える可能性があるため、これはより困難です。

### 4.5.4 timing constraintsの適応

PCIe ブロックの clocks の変更が発生した場合は、それを反映するように timing constraints を調整する必要があります。

constraints は、選択した FPGA と Vivadoのリビジョンに依存するため、これを正しく行うのは多少難しい場合があります。この調整を回避することが、 link speed と lanes の数の組み合わせを (可能な場合) 選択することによって、 PIPE clockの周波数を変更しないようにしようとする主な動機です。ただし、 PIPE clockの周波数が変わらない場合でも、 constraints の更新が必要な場合があります。

繰り返しになりますが、 Ultrascale ファミリ (およびそれ以降) の FPGA を使用する場合、 PCIe ブロックの IPs がこれを内部で処理するため、 timing constraintsを処理する必要はありません。

timing constraintsを調整するには、まずサンプル プロジェクトの constraints を見つけます。 Vivadoでは、通常、`example.srcs/constrs_1/imports/example_design/xilinx_*.xdc`形式のファイルです。

lanes および/または link speedの数が変更される前の PCIe ブロックの設定で 1 つのサンプル プロジェクトを生成し、これらの変更後に 1 つのサンプル プロジェクトを生成することを強くお勧めします。 2 つのサンプル プロジェクトの constraint ファイル間の単純な diff は、 constraints の適応が必要かどうか、必要な場合はどのような方法であるかについて明確な答えを与えます。

constraint ファイルの “Timing constraints” セクションと xillydemo.xdcのセクションを比較します。サンプル プロジェクトでは、 logic 階層内の絶対位置によって logic 要素を選択するため、編集が必要です。たとえば、サンプル プロジェクトで次のような timing constraint があるとします。

```
set_false_path -to [get_pins {pcie_vivado_support_i/pipe_clock_i/
pclk_i1_bufgctrl.pclk_i1/S0}]
```

xillydemo.xdc では、次のように記述します。

```
set_false_path -to [get_pins -match_style ucf */pipe_clock/
pclk_i1_bufgctrl.pclk_i1/S0]
```

主な違いは、 Xillybusの constraintsで使用される相対パスにあります。 Xilinxのツールの以前のリビジョンでは一部の constraints が必要であり、それ以降のリビジョンでは不要になるため、他にもわずかな違いがある可能性があります。

timing constraintsで変更を行った後、 designの implementationの後に timing report を確認して、変更が logic に反映されたことを確認することが重要です。

最後に、一部の demo bundlesの xillydemo.xdc に存在する次の 2 つの constraintsについて説明する価値があります。

```
set_case_analysis 1 [get_pins -match_style ucf */pipe_clock/
pclk_i1_bufgctrl.pclk_i1/S0]
set_case_analysis 0 [get_pins -match_style ucf */pipe_clock/
pclk_i1_bufgctrl.pclk_i1/S1]
```

これらの constraints は、 Xilinx AR #62296で説明されているように、 Vivadoのかなり古いリビジョンを使用する場合、 Gen1 PCIe ブロックに必要です。したがって、最近の Xilinx ツールでは省略される場合があります。

### 4.5.5 PIPE clock モジュールの更新

前述のとおり、この手順は Ultrascale FPGAs 以降では必要ありません。

場合によっては、特に link speed を 2.5 GT/s (Gen1)から、または 2.5 GT/s (Gen1)に増やす場合、 vivado-essentials/ ディレクトリにある pcie_*_vivado_pipe_clock.v ファイルを更新する必要があります。

このファイルは、 xillydemo-vivado.tclを実行することにより、初期プロジェクトセットアップの一部として自動的に生成されます。 PCIe ブロックの構成によって、特に 2.5 GT/s に限定されているかどうかによって、わずかに変化する可能性があります。

推奨される方法は、 Vivado プロジェクトを xillydemo-vivado.tcl scriptで再生成することです。つまり、新しい demo bundleから開始し、前の段階で変更されたファイルをそこにコピーします。

- PCIe ブロックの XCI ファイル
- xillydemo.xdc
- lanes および link speedの新しい数に適応するために編集された Verilog / VHDL ファイル。

これらのファイルを新しい demo bundleにコピーして、 xillydemo-vivado.tcl scriptでプロジェクトを生成すると、 PIPE clock module が PCIe ブロックの設定に従っていること、およびプロジェクトが変更前の残りのファイルに依存しないことが保証されます。

または、段落 4.5.3で作成されたサンプル プロジェクトから pcie_*_vivado_pipe_clock.vファイルを更新します。使用するファイルは、 vivado-essentials/のファイルとまったく同じ名前で、通常、サンプル プロジェクトのファイル階層の奥深くにあります。このファイルを vivado-essentials/ にコピーします (既存のファイルを上書きします)。

### 4.5.6 Versal FPGAs

このセクションは、 Versal FPGAsのみに関連しています。

他の FPGAsとは異なり、 Versal FPGAs の PCIe ブロックは block designで構成されています。それに伴い、 PCIe lanes または link speed の番号変更の手順が異なります。特に、 block design (pcie_versal_xxl) の一部のブロックの内容を削除して、再度作成する必要があります。 Vivado は、 Block Automationと呼ばれる機能を使用してこれらのブロックを自動的に作成します。これを行う手順は次のとおりです。

- 既存の pcie_versal_xxl ブロックの instantiation template を取得します。sourcesのリストで pcie_versal_xxl エントリを右クリックし、 “View Instantiation Template”を選択します。 “Save As...” を使用して、後で参照できるようにこのコピーを保持します。
- pcie_block_support ブロックを削除します ( CIPS ブロックは削除しません)。
- すべての外部 ports とそれらへのワイヤを削除します。
- pcie_block ブロックを開き、必要な変更を行います。
- 予期しない変更がないことを確認します (特に Product ID が変更されていないことを確認します。これは偶然に発生する可能性があります)。
- CIPS ブロックと pcie_block ブロックの間の reset signal を切断します (そうしないと、 Block Automation は有効になりません)。
- Block Automation を実行し、 Vivadoによって提供されるデフォルトを受け入れます。これにより、以前に削除されたブロックが再作成されます。
- 外部 portから sys_reset 信号を削除し、代わりに pl_pcie0_resetnという名前の CIPS ブロックの port に接続します。切断された外部 portを削除します。

ワイヤは両方のブロックの sys_reset input (pcie_block と pcie_block_support) に到達することに注意してください。

- pcie_block ブロックのすべての未接続の ports を外部 portsに接続します。 CTRL が押されたブロックの ports をクリックし、どこかを右クリックして “Make External”を選択します。 pcie_block_supportの ports はすでに外部になっているはずです。
- 前と同じように instantiation template againを取得し、保存します。前作と比べてみてください。違いはまったくないはずです（作成時を除く）。

## 4.6 FPGA 部品番号の変更

ある FPGA ファミリから別のファミリに移行する場合、別の demo bundleから開始する必要があります。 PCIe ブロック (および XillyUSBを使用した MGT ブロック) に関連する違い (微妙な場合もあります) があります。これらの違いには、異なる Xillybus IP core と異なる wrapper modulesが必要です。

Vivado / ISE でプロジェクトの一部を変更しようとすると、 implementationでプロジェクトにエラーが発生する可能性があります。 implementation が成功したとしても、 logic が動作しないか、信頼性が低い可能性があります。

ただし、同じ FPGA ファミリ内にとどまる場合は、多くの場合、部品番号を変更するだけで十分です (ピン配置と constraintsに関する上記の考慮事項に加えて)。

一部の FPGA ファミリ (特にUltrascale ) では、 logic fabric 内の PCIe ブロックの位置 (site) は PCIe ブロック自体の属性であるため、変更が必要になる場合があることに注意してください。さらに、それぞれの特定の FPGAとそれぞれの特定の packageには、有効な sitesの独自のセットがあります。したがって、 PCIe ブロック用に選択された site が新しい FPGAに存在しない場合、 FPGA の変更は PCIe IPの属性をリセットする可能性があります。

PCIe ブロックの位置 (site) にある無効なサイトに対するVivadoの反応は、非常に破壊的です。この場合、 PCIe ブロックの “upgrade” ( FPGAを変更した後に IP のロックを解除するために常に必要な操作) により、 PCIe ブロックのいくつかの属性が任意の値にリセットされます。そうしている間に、次のような Critical Warning が生成されます。

```
CRITICAL WARNING: [IP_Flow 19-3419] Update of 'pcie_ku' to current project
options has resulted in an incomplete parameterization. Please review the
message log, and recustomize this instance before continuing with your design.
```

この問題に注意を払わずにプロジェクトの implementation を試みることは完全に無意味であり、誤解を招くような多数の warnings、 Critical Warnings 、およびおそらくエラーにつながるだけでなく、その結果があったとしても、機能することにはほど遠いでしょう。

解決策は、プロジェクトの部品番号属性を変更する前に、新しい FPGAで有効な PCIe ブロック サイトを割り当てることです。変更前後の FPGA 部分に共通のサイトがない場合は、 XCI ファイルを手動で編集する必要がある場合があります (この場合、 GUIでこの変更を行うことができず、許可されたサイトに設定が制限されるためです)。現在の FPGA パーツ上)。

# 5

# トラブルシューティング

## 5.1 implementation中のエラー

Xilinxのツールのリリース間のわずかな違いにより、 bitfileを作成するための implementation の実行に失敗することがあります。

問題がすぐに解決しない場合は、 Xillybusの Web サイトにある電子メール アドレスから支援を求めてください。失敗したプロセスの出力ログを添付してください。特に、ツールによって報告された最初のエラーの前後に注意してください。また、design でカスタム変更が行われた場合 (つまり、 demo bundleからの流用)、これらの変更について詳しく説明してください。また、 Vivado (または ISE) のどのバージョンが使用されたかを記載してください。

## 5.2 PCIe ハードウェアの問題

通常、 PCIe カードは hostの BIOS および/またはオペレーティング システムによって適切に検出され、 hostの driver は正常に起動します。

ほとんどの PC コンピューターでは、 BIOS は boot プロセスの早い段階で、検出された周辺機器のリストを短時間表示します。 Xillybus インターフェイスが正常に検出されると、 vendor ID 10EE および device ID EBEB を含む周辺機器がリストに表示されます。

オペレーティング システムによるカードの検出については、次の 2 つのドキュメントのいずれか該当する方を参照してください。

- Getting started with Xillybus on a Linux host
- Getting started with Xillybus on a Windows host

カードの検出の失敗 (またはコンピューターの boot プロセスの失敗) は、 busとのインターフェイスを Xilinxの PCIe IP core に依存している Xillybus IP coreとは関係ありません。

最初に、次のことを確認することをお勧めします。

- bitstream は、コンピューターの電源を入れたとき (または PCI-SIG 仕様に関しては、電源を入れた直後) に既に FPGA にロードされていました。
- reference clock を含む PCIe ワイヤの pinouts は正しいです (これは placement reportで確認できます)。
- ボードは、正しい reference clock を FPGAに提供します。

問題がすぐに見つからない場合は、ボードに付属の PCIe のサンプル プロジェクトを試すことをお勧めします。これにより、不適切なジャンパ設定や、ハードウェアの欠陥が明らかになる場合があります。

カードがこのサンプルで検出され、 Xillybusでは検出されない場合は、2 つのdesignsの pinouts を比較すると役立つ場合があります。それらが等しい場合、次のステップは、 IP GUI を呼び出して Xilinxの PCIe cores の属性を比較することです (各プロジェクトを開いた状態で Project Manager の XCI / XCO 要素をダブルクリックします)。

次の構成要素は、調整が必要な場合があります。

- reference clock の周波数 ( GUIでは “Interface Frequency” と表示される場合があります)。
- base class および sub class (可能性は低いですが、 class が不明であると見なされた場合、一部の比較的古い PC コンピューターは boot プロセスに失敗しました)。
- ベース アドレスの register 設定、 vendor ID、 device ID 、および割り込み設定を除き、変更しないでください。

問題が解決しない場合は、 Xillybusの Web サイトにある電子メール アドレスから支援を求めてください。