(機械で日本語に翻訳)

The guide to defining a custom Xillybus IP core

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 This document may also be slightly outdated in relation to the original.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

1.1 全般的

Xillybus は、さまざまなアプリケーションに対応する多目的プラットフォームです。したがって、各ユーザーは特定の要件 (streams の数、方向、パフォーマンスに関連する属性、およびリソースの消費) を満たすカスタム IP core を簡単に作成およびダウンロードできます。

カスタム IP cores の定義と作成を簡素化するために、IP Core Factory (http://xillybus.com/custom-ip-factory) というオンライン ツールを使用できます。

このツールは、ユーザーが要求された device files とその構成を定義できる単純な Web アプリケーションで構成されています。定義が完了すると、FPGA プロジェクトに含めるファイルが自動プロセスによって生成されます。カスタム IP core は、しばらくすると (通常は数分) zip ファイルとしてダウンロードできるようになります。

ダウンロードしたカスタム IP core は完全に機能します。実際のアプリケーションでこの IP core をテストして使用することに技術的な制限はありません。

Web アプリケーションはこのガイドを読まなくても使用できますが、最初に demo bundle を実行して Xillybus に慣れることをお勧めします。 device files の属性をよりよく理解し、制御したいユーザーは、このガイドでいくつかの背景情報を見つけることができます。

demo bundle に慣れていないユーザーには、次のドキュメントのいくつかを事前に読むことをお勧めします。

- Getting started with the FPGA demo bundle for Xilinx
- Getting started with the FPGA demo bundle for Intel FPGA

- Getting started with Xillinux for Zynq-7000
- Getting started with Xillybus on a Linux host
- Getting started with Xillybus on a Windows host

カスタム IP core の必要性が明らかな場合でも、demo bundle から始めるのが最善です。これにより、IP core を application logic と統合する方法と、プロジェクト全体をセットアップする方法が明確になります。

IP core のカスタム構成に関するすべての情報は、FPGA の IP core 自体に保存されます。 host の driver は、driver の初期化時にこの情報を取得します。したがって、IP core を交換しても host を変更する必要はありません。

よくある間違いは、リソースを節約するために、構成されている streams の数を最小限に抑えようとすることです。セクション 3 および 4.4 では、Xillybus IP core がどのように拡張されるかを示し、streams を十分に割り当てることが理にかなっている理由を説明します。

1.2 カスタム IP core を統合する方法

IP Core Factory からカスタム IP core をダウンロードした後、この IP core を含めるように demo bundle を変更する必要があります。これには、いくつかの簡単な手順が必要です。命令は、IP Core の zip ファイルの一部である README.TXT で書かれています。これらの手順は、便宜上以下にも記載されています。

この README ファイルには、その他の有用な情報も含まれています。

- 5桁の数字であるCore ID。この番号は、IP core の一意の識別子です。価格見積もりをリクエストする際は、Core ID について言及する必要があります。
- IP core の devices files がリストされています。各 device file の技術的な詳細も表示されます。これは、IP core の実際の特性に関する正確な情報です。

custom IP core を demo bundle に統合するには、次の手順に従います。

1. demo bundle の 2 つのファイルを、IP Core の zip のファイル xillybus.v と xillybus_core.v (または xillybus_xl_core.v / xillybus_xxl_core.v) に置き換えます。
2. IP core自体を交換してください。このファイルは、“core/” という名前で demo bundle の subdirectory にあります。置き換えるファイルは、xillybus_core.ngc、xillybus_core.e などです。

3. xillydemo.v (または xillydemo.vhd) を編集して、目的のアプリケーションをこのカスタム IP core と統合します。ガイダンスについては、IP core の zip ファイルの一部である “instantiation templates” という名前のディレクトリを参照してください。 template.v (または template.vhd) という名前のファイルには、従うべき instantiation template が含まれています。

2

カスタム IP cores の定義

2.1 概要

IP Core Factory は、カスタム IP core を最初から定義するか、demo bundle の core の構成を出発点として使用するための、wizard に似た Web アプリケーションです。

ほとんどの場合、“Autoset internals” オプションを有効にしたままにして、IP Core Factory に依存して各 stream の属性を設定することをお勧めします。 stream のパラメーターを微調整するためにこのオプションをオフにすることは非常によくある間違いであり、ほとんどの場合、パフォーマンスの低下につながります。

特に、IP core が期待されるデータ レート パフォーマンスを満たさない場合、問題が別の場所にある可能性が高くなります。この場合、IP core のフル パフォーマンスを実現する方法について説明している次の 2 つのガイドのいずれかを参照することをお勧めします。

- Getting started with Xillybus on a Linux hostのセクション 5
- Getting started with Xillybus on a Windows hostのセクション 5

もう 1 つのよくある間違いは、DMA buffers のサイズを調整するために “Autoset internals” をオフにして、送信するデータ パケットのサイズと一致するようにすることです。これについては、セクション 2.6 で説明します。

これらは、このツールを使用する際に強調する価値のある追加のポイントです。

- IP core は netlist として提供されるため、IP core が意図されている FPGA ファミリを正しく選択する必要があります。

- 各 device file の “use” 属性を、意図した目的に一致する説明に設定することが重要です。これにより、stream の属性が正しく設定されます。
- XillyUSB IP cores の場合、“Expected bandwidth” 属性は、stream が要求する最大帯域幅に正確に設定する必要があります。これは、データ レートがその値に制限されているためです。他のバリアント (PCIe および AXI) の場合、この属性はパフォーマンス チューニングにのみ影響します。要件を誇張してより良い結果を得ようとするのではなく、現実的な数値を適用する必要があります。このような誇張は、特定の限られたリソースを実際に必要とする他の streams のパフォーマンスを低下させる可能性があります。

このセクションの残りの部分では、device files の属性のいくつかについて説明します。

2.2 device fileの名前

各 stream には名前が指定され、host 上で作成される device file の名前として使用されます。

名前は常に xillybus_* の形式を取ります (例: xillybus_mystream)。 XillyUSB の場合、名前は xillyusb_NN_* のようなもので、NN はインデックスです。通常、XillyUSB デバイスが 1 つしか host に接続されていない場合は 2 つのゼロです。

Linux システムでは、stream はプレーン ファイル (/dev/xillybus_mystream など) として開かれます。 Windows では、同じ stream が \\.\xillybus_mystream として表示されます。

device file は、反対方向の 2 つの streams を表すことができます。これは、たまたま device file の名前を共有する 2 つの streams です。これら 2 つの streams は、いずれかの方向に個別に開くことも、読み書き用に開くこともできます。この機能は、混乱を避けるために一般的に避けるべきですが、device file が双方向 pipe を期待するソフトウェアに渡される場合に役立ちます。

2.3 データ幅

データ幅は、FPGA で FIFOs からフェッチまたは FIFOs に書き込まれるワードのビット数です。許可されている選択肢は、32、16、または 8 ビットです。 XillyUSB と同様に、リビジョン B/XL/XXL の Xillybus IP cores (これらについてはセクション 4 で説明します) では、より広いデータ幅が許可されています。

stream で高帯域幅のパフォーマンスが必要な場合、および IP core のリビジョンが PCIe の場合は A、AXI の場合は任意の IP core の場合、データ幅を 32 ビットに設

定する必要があります。基礎となるトランスポート (PCIe bus トランスポートなど) の非効率的な使用に起因します。

その理由は、ワードが bus clock の速度で Xillybus の内部データ パスを介して転送されるためです。その結果、8 ビット ワードの転送には 32 ビット ワードと同じタイム スロットが必要となり、事実上 4 倍遅くなります。

これは、データ パスが低速のデータ要素で占有されるため、特定の時点で基盤となるトランスポートを競合する他の streams にも影響を与えます。

IP core および XillyUSB の新しいリビジョンでは、内部データ パス構造が異なるため、この制限はありません。

とにかく、データ幅に一致する粒度で host application で I/O 操作を実行することをお勧めします。たとえば、データ幅が 32 ビットの場合、関数 read() および write() を 4 の倍数のデータ長で呼び出します。

データ幅を適切に選択しないと、望ましくない動作が発生する可能性があります。たとえば、host から FPGA へのリンクが 32 ビット幅の場合、host で 3 バイトのデータを書き込むと、driver は、FPGA に何かを送信する前に 4 番目のバイトを無期限に待機します。

2.4 同期または非同期 stream

この属性は、“use” 設定の選択に基づいて、“Autoset internals” オプションが選択されたときに自動的に設定されます。

ほとんどの場合、連続データ フローには非同期 streams が適切であり、コマンド、制御データ、およびステータス情報の取得には同期 streams が適切です。

同期 streams の場合、すべての I/O (FPGA のデータ フローを含む) は、read() または write() への関数呼び出しの呼び出しと戻りの間にのみ発生します。これにより、いつ何が起こるかを完全に制御できますが、CPU が他のことをしている間、データ転送リソースは未使用のままになります。次の 2 つのドキュメントのいずれかのセクション 2 で、この件に関する詳細を読むことをお勧めします。

- Xillybus host application programming guide for Linux
- Xillybus host application programming guide for Windows

同期 streams を使用すると、ソフトウェア プログラミングがより直感的になりますが、帯域幅の使用率にマイナスの影響があります。非同期 streams を使用すると、オペレーティング システムが user space で実行されているプロセスから CPU を一定期間離しても、継続的なデータ フローを維持できます。

この主題を要約すると、次のような指針となる質問があります。

- downstreams (host から FPGA) の場合: データが FPGA に到達する前に write() 操作が戻っても問題ありませんか?
- upstreams (FPGA から host) の場合: host での read() 操作が要求する前に、Xillybus IP core が FPGA で user application logic からデータのフェッチを開始しても問題ありませんか?

それぞれの質問に対する答えが「いいえ」の場合、同期 stream が必要です。それ以外の場合は、通常、データ フローの制御が少なく、直感的ではないことを理解した上で、非同期オプションを選択することをお勧めします。

2.5 バッファリング時間

Xillybus は、FPGA と host の間で連続した stream のデータの錯覚を維持します。DMA buffers の存在は、FPGA の user application logic および host のアプリケーション ソフトウェアに対して透過的です。これらは、データ フローの効率と、特に高データ レートでの連続性を維持する能力を制御する場合にのみ重要です。

data acquisition や data playback などのアプリケーションでは、FPGA でデータの継続的なフローが必要です。そうしないと、データが失われます。このフローを維持するために、user space application は、FPGA のアクティビティによって DMA buffers が (それぞれ) フルまたは空になるのを防ぐのに十分な頻度で、read() または write() に関数呼び出しを行う必要があります。

ただし、これらの関数呼び出しが十分に頻繁に行われるようにすることには問題があります。Linux や Windows などの一般的なオペレーティング システムは、理論的に任意の期間、user-space application から CPU を奪う可能性があります。FPGA は、関係なく driver の buffers を満たしたり排出したりし続けます。したがって、DMA buffers は、このような CPU の一時的な剥奪にもかかわらず、継続的なデータ フローを維持するのに十分な大きさである必要があります。

ここでの説明のために、バッファリング時間とは、stream がすべての DMA buffers が空の状態からすべての DMA buffers がいっぱいになった状態に変化するのにかかる時間であり、データが buffers を所定のレートで満たす場合です。 streamが意図されているものです（その間、それらは排出されません）。

“Autoset internals” を有効にして Xillybus stream をセットアップすると (推奨)、“Buffering” というタイトルの選択ボックスが Web アプリケーションに表示されます。ここで、必要なバッファリング時間を選択します。

連続性を維持する必要がある非同期 stream の場合、選択された時間は、CPU が user space application から取り除かれると予想される最大時間を反映する必要があります。

“Maximum” を選択すると、buffers を割り当てるアルゴリズムは、他の streams を少し考慮して、できるだけ多くの RAM を割り当てようとします。

望ましいバッファリング時間 t と予測される帯域幅 W が与えられると、アルゴリズムは、次の式に基づいて driver の DMA buffers に RAM、M の合計量を割り当てようとします。

$$M = t \times W$$

ただし、実際の buffer サイズは常に 2 のべき乗 (2^N) です。また、必要なバッファリング時間を満たすのに十分なメモリを割り当てることができない場合もあります。

したがって、IP core の README ファイルで割り当てられた buffer サイズを調べ、それが使用できるかどうかを確認することが重要です。 buffer のサイズを手動で設定する (つまり、 “Autoset internals” をオフにする) ことが、streams 間で RAM をより適切に分散させるために必要な場合があります。これは、意図したアプリケーションにより適しています。

2.6 DMA buffersのサイズ

Web アプリケーションで “Autoset internals” を有効にして、ツールが DMA buffers のパラメータを自動的に設定できるようにすることをお勧めします (上記のセクション 2.5 を参照してください)。シナリオによっては、自動設定がアプリケーションに適さない場合があります。その場合、DMA buffers のサイズと数を手動で設定することができます。

非同期 streams の場合、buffers のパラメーターは、次の 2 つのドキュメントの “Continuous I/O at high rate” というセクションで説明されている重要な影響を及ぼします。

- Xillybus host application programming guide for Linux
- Xillybus host application programming guide for Windows

DMA buffers のサイズを、read() および write() の意図した関数呼び出しのサイズに適合させる必要はまったくありません。これら 2 つのガイドで説明されているように、DMA buffers のサイズは無関係であり、read() と write() への関数呼び出しでは

透過的です。特に、read() への関数呼び出しは、十分なデータが FPGA の IP core に到達した場合 (DMA buffer のフィル レベルに関係なく)、すぐに戻ります。これは、FPGA が部分的に満たされた DMA buffer を送信できるようにする、FPGA と host の間のメカニズムのおかげです。このメカニズムは、read() への関数呼び出しをすぐに完了するのに役立つ場合に使用されます。

同様に、host から FPGA へのデータは、明示的な要求によって FPGA にすぐに到達することが保証されます。

DMA buffers のサイズと意図したデータ交換のパターンを関連付けるのはよくある間違いです。 Xillybus では、その必要はありません。これが、“Autoset internals” が DMA buffers のサイズを設定するための好ましい選択である理由です。

CPU がアプリケーションから奪われた場合でも、DMA buffers 内の総容量がデータの流れを継続的に維持するため、継続性のためには、RAM が多い方が優れています。正しい決定を下すには、上記のプログラミング ガイドで詳しく説明されている他の要因が関係します。

ただし、DMA buffers の**合計サイズ**が大きすぎると、大量のデータを保存できる buffering delay のリスクが生じます。その結果、一方が buffers を空にするよりも速く buffers をいっぱいにすると、かなりの時間が経過してからデータがもう一方の端に到着する可能性があります。これは、“Monitoring the amount of buffered data” というセクションの Xillybus FPGA designer’s guideで説明されている手法によって制御できます。

XillyUSB IP cores の場合、stream ごとに 1 つの buffer があり、driver によって管理される大きな FIFO として機能します。他の IP cores (PCIe および AXI) は、各 stream に対して複数の DMA buffers を維持するため、それらのサイズと数の両方が定義されています。したがって、DMA buffers の有効サイズは、各 DMA buffer のサイズにその数を掛けたものになります。

したがって、PCIe / AXI をベースにした IP cores で “Autoset internals” をオフにする場合は、DMA buffers の数とそれぞれのサイズを指定する必要があります。次の点を考慮する必要があります。

- 各 DMA buffer のサイズは、host から FPGA までの streams で独自の意味を持ちます。これらの buffers がいっぱいになると、データは FPGA に送信されます (flush がソフトウェアによって明示的に要求されない限り、または stream が 10 ミリ秒間アイドル状態である場合を除きます)。 ）。したがって、各 DMA buffer のサイズは、流れるデータの典型的な latency の影響を受けます。
- 低速の streams (10 MBytes/s 未満) の場合、DMA buffers の推奨数は 4 です。より高い帯域幅が必要な場合は、適切な全体的な DMA buffer 割り当てを達成

するために buffers の数が選択されます。各 buffer を 128 kBytes 以下にできる場合、高帯域幅 streams の DMA buffers の適切な数は 16 から 64 の間です。

- host で拡張 driver が使用されていない限り、すべての streams に対する DMA buffers の合計割り当ては 512 MBytes を超えてはなりません。そうしないと、オペレーティング システムがこれ以上の割り当てを拒否し、driver の初期化に失敗する可能性があります。
- buffer がいっぱいになるたびに、hardware interrupt が host に送信されます。予想されるデータ レートを考慮して、interrupts のレートを計算し、processor にとって適切なレベル (1 秒あたり数千以下) に維持する必要があります。
- DMA Buffers の数を増やして合計サイズに達する場合、各 DMA Buffers のサイズは 128 kBytes を超えてはなりません。

driver の DMA buffers の問題は、同期 streams ではそれほど重要ではありません。そのような場合、stream に代わって buffers に割り当てられる RAM の合計は、read() および write() の意図された関数呼び出しのデータ長の大きさのオーダーである必要があります。すでに上で述べたように、buffers のサイズをこれらの関数呼び出しに適合させる必要はありませんが、kernel RAM をそれよりも大きくして無駄にすることにはほとんど意味がありません。

2.7 DMA acceleration

PCIe をベースにした IP cores では、host から FPGA への streams では、DMA データ転送の高速化が必要になる場合があります。

この方向でデータ交換を行うために、PCIe bus protocol は、FPGA が host からデータの要求を発行し、データが到着するのを待つ必要があると述べています。要求が bus を移動し、host によってキューに入れられて処理され、データが戻ってくるときに固有の遅延が発生します。このターンアラウンド タイムのギャップにより、bus の効率がいくらか低下し、単一の stream の帯域幅が 40% まで低下することがあります。

この問題を回避するには、複数のデータ要求を送信して、連続送信中に host のキューに常に要求があるようにします。さまざまな要求からのデータがランダムな順序で到着する可能性があるため、FPGA 上の RAM buffers に格納して、順序付けられたデータの流れを application logic に提示する必要があります。

FPGA 内の各 buffer は、要求されたデータのセグメントを格納するために使用されます。 DMA accelerations で現在可能な設定は次のとおりです。

- なし。 FPGA にはデータは保存されません。データの各要求は、前のデータからすべてのデータが到着した場合にのみ送信されます。
- それぞれ 512 バイトの 4 つのセグメント。 FPGA では、block RAM の 2048 バイトが割り当てられます。任意の時点で最大 4 つのデータ要求をアクティブにすることができます。
- それぞれ 512 バイトの 8 つのセグメント。 block RAM の 4096 バイトが FPGA に割り当てられます。任意の時点で最大 8 つのデータ要求をアクティブにすることができます。
- リビジョン B 以降の IP cores には、それぞれ 512 バイトの 16 セグメントのオプションもあります。

リクエストからデータ到着までの所要時間は、host のハードウェアによって異なります。したがって、実際の帯域幅パフォーマンスは異なる場合があります。

IP Core Factory で “Autoset internals” を使用する場合、アクセラレーション リソースの自動割り当ては、一般的な PC コンピュータ ハードウェアでの測定結果に基づいており、まれに手動での調整が必要になる場合があります。

3

スケーラビリティと logic のリソース消費

3.1 全般的

Xillybus は、スケーラビリティを考慮して設計されました。カスタム IP core を単一の stream として構成することは完全に理にかなっていますが、多数の streams にスケールアップしても、Xillybus core によって消費される logic の量に与える影響は比較的小さくなります。

logic の消費量を測定するために、streams の数を増やして Xillybus IP core (ベースライン、リビジョン A) の連続ビルドを作成しました。すべてのテストで、FPGA から host までの streams の数は、他の方向と同じでした。 streams の数は、2 (各方向に 1 つ) から 64 (各方向に 32) の範囲でした。

このセクションでは、FPGAs の 3 つのファミリでの IP core 自体による logic の消費量の概要を、ツールによって報告されています。これらの FPGAs (Xilinx による) はかなり時代遅れですが、Xilinx と Intel によって、より最近の FPGAs でも同様の結果が得られます。

リビジョン B、XL、および XXL を使用した IP cores の同様の分析については、セクション 4 を参照してください。

XillyUSB は、これらの分析のいずれにも含まれていません。

3.2 Block RAMs

Xillybus core で使用される block RAMs の数は、ゼロから数個の間で変化します (64 個の streams に対して 3 block RAMs)。各 stream の Xillybus core 内に buffers はありません。むしろ、Xillybus core はそれに接続された FIFOs に依存してデータを収集します。内部的には、core には、すべての streams で使用される単一のメモリ

プールがあります。

streams の数が増えると、DMA buffers のアドレスを格納するために block RAMs が使用されます。

core の README ファイルに詳述されているように、host から FPGA への streams の DMA acceleration には、追加の block RAMs が割り当てられます。

3.3 logic fabricのリソース

以下のグラフは、streams の数が 2 から 64 になるにつれて、LUTs と registers (flip-flops) の消費量を示しています。これらのグラフの各ドットは、synthesis レポートに示されているように、事実上の使用量です。これらのグラフから明らかなのは、logic の消費量がほぼ直線的に増加していることです。 FPGA アーキテクチャに関係なく、stream ごとに、平均で約 110 個の LUTs と 82 個の registers が追加されます。

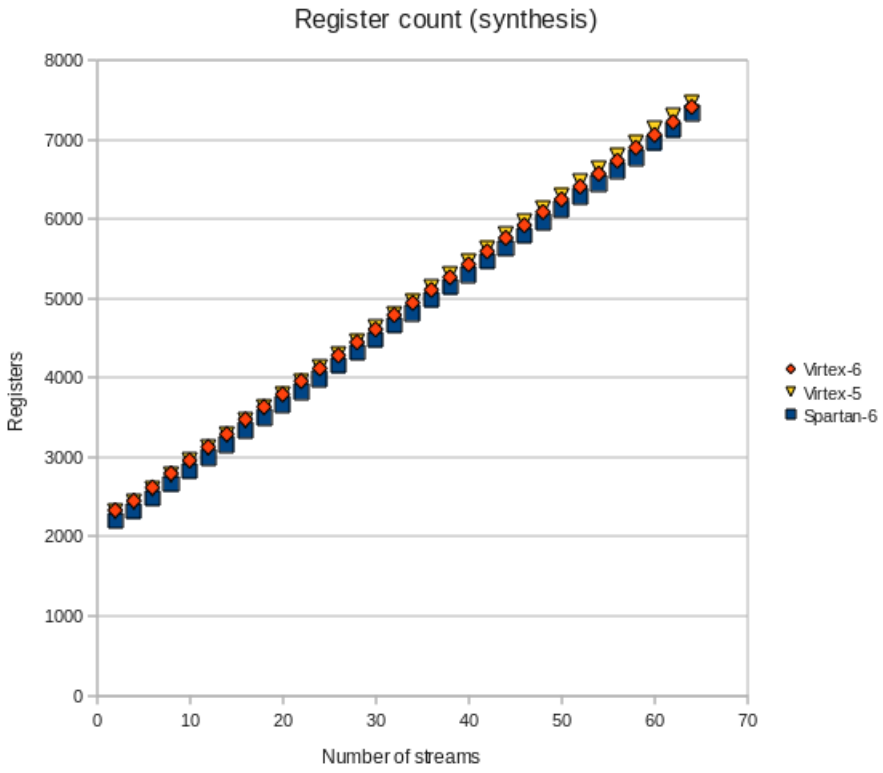
FPGA で実際に消費される slices の数は、logic の要素がそれらにどのようにパックされているかによって異なります。 Spartan-6 または Virtex-6 ファミリでは、各 slice に最大 8 つの LUTs および 8 つの registers を含めることができます。したがって、非常に楽観的なアプローチは、registers が完全にパックされていると仮定することです。したがって、各 stream は、消費されるリソースに 110/8 = 14 slices のみを追加します。一方、その半分の効率でパッキングすることは、かなりの労力を必要とせずに達成できるものです。したがって、stream の slices での予想コストは、14-28 slices の範囲で見積もることができます。

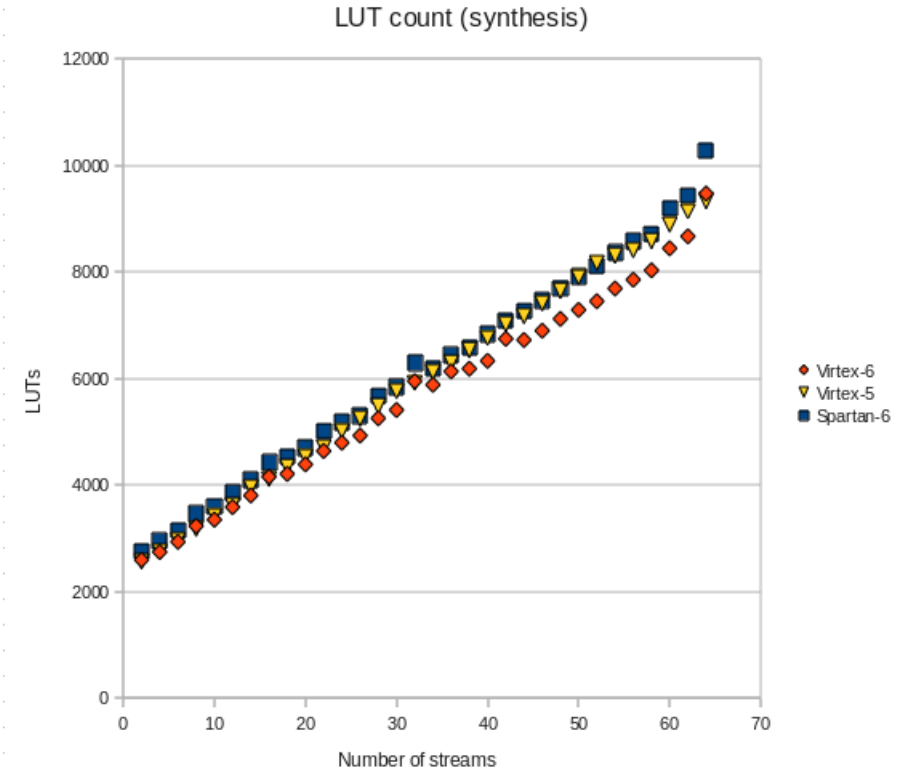
FPGA がほぼ満杯でない場合、implementation を実行するツールは logic を slices に効率的にパックしようとしないため、slices の数の増加が大幅に急勾配になる可能性があることに注意することが重要です。この場合、リソースがたくさんあるため、ツールはリソースを浪費します。

ベンチマーク テストで選択した設定は、upstreams では 50%、downstreams では同じです。実際の IP cores は、通常、方向の 1 つに重点を置いていますが、以下の結果は、何を期待すべきかを示しています。

グラフが続きます。傾斜が急に見えるかもしれませんが、streams の数が最小限の IP core (2 streams) からかなり重いもの (64 streams) になることに注意してください。

肝心なのは、slices の数に対する streams の貢献度がかなり低いため、最も些細なタスクであっても、IP core に追加の streams を割り当てることは理にかなっているということです。


Register count (synthesis)
Registers
Number of streams
0
1000
2000
3000
4000
5000
6000
7000
8000
0
10
20
30
40
50
60
70
Virtex-6
Virtex-5
Spartan-6


LUT count (synthesis)
12000
10000
8000
6000
4000
2000
0
LUTs
0
10
20
30
40
50
60
70
Number of streams
Virtex-6
Virtex-5
Spartan-6

4

リビジョン B、XL、および XXL の IP cores

4.1 全般的

この時点まで、このドキュメントは、2010 年以降に利用可能な IP cores のベースライン リビジョン (リビジョン A) に関連しています。 リビジョン B および XL は、Xillybus のユーザーのデータ帯域幅のニーズに合わせて 2015 年に導入されました。これらの cores は、リビジョン A を徐々に置き換えます。

リビジョン XXL は 2019 年に導入されました。

新しいリビジョン (B、XL および XXL) は、リビジョン A と比較して機能のスーパーセットを提供しますが、同じ属性で定義された場合、機能的に同等です (パフォーマンスが向上する可能性があります)。

最も顕著な違いは次のとおりです。

- データ帯域幅の増加: リビジョン B、XL、および XXL の FPGA、IP cores では、それぞれリビジョン A の帯域幅の約 2 倍、4 倍、および 8 倍の総帯域幅が可能です。

  Xillybus IP cores の帯域幅機能を実現する方法については、Getting started with Xillybus on a Linux host または Getting started with Xillybus on a Windows host のセクション 5 を参照してください。

- 8、16、および 32 ビットの既存のオプションに加えて、64、128、および 256 ビットのユーザー インターフェイス データ幅が許可されます。これらの幅は、使用中の Xillybus の IP core と PCIe ブロック間のデータ パスの幅に関係なく許容されます。

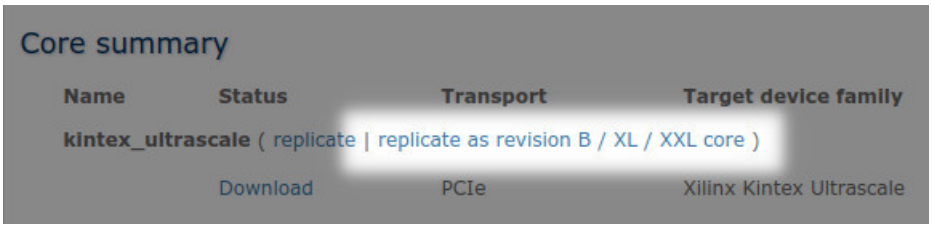
- logic design はより高速で (timing constraints に到達するのがより簡単です)、最も遅い timing path での遅延が約 1 ns 少なくなります。

- ほとんどの場合、logic の消費量は少なくなります (セクション 4.4 を参照)。
- IP core と application logic 間の信号のデータ幅に関係なく、PCIe ブロックの帯域幅が効率的に利用されます。これは、リビジョン A での 8 ビットおよび 16 ビット ワードの streams の効率の低下とは対照的です。
- Xilinx プラットフォームでは、リビジョン B、XL、および XXL は、Vivado でのみ使用できます。

4.2 リビジョン B/XL/XXL での作業

リビジョン B/XL/XXL の IP core は、リビジョン A の IP core を複製することによって IP Core Factory で作成されます。

これは、IP Core Factory からのこのスクリーンショットのように、“replicate as revision B / XL / XXL core” をクリックすることによって行われます。



リビジョン A にダウングレードすることはできません。

B/XL/XXL へのアップグレードの可能性は、これらの高度な IP cores へのアクセスを要求したユーザーに対してのみ有効になります。このようなリクエストは、ウェブサイトで宣伝されている連絡先情報を使用して、プレーンな電子メールで行われます。

このアクセスを取得するための特別な要件はありません。このリクエストの目的は、単にハイエンド ユーザーとの連絡を密にすることです。

リビジョン B の IP cores は、リビジョン A の drop-in replacements です。したがって、目的の FPGA のベースライン demo bundle を出発点として使用する必要があります。この demo bundle はリビジョン A の IP core と一緒に出荷されるため、リビジョン B で作業したい場合は、IP Core Factory から構成してダウンロードする必要があります。

一方、リビジョン XL および XXL を使用するには、専用の demo bundle が必要です。これらの demo bundles は、電子メールで要求する必要があります。

4.3 データワードの幅

リビジョン A の IP cores では、8、16、および 32 ビットのアプリケーション データ幅のみが許可されますが、リビジョン B/ XL /XXL では、64、128、および 256 ビット幅のインターフェイスも許可されます。主な動機は、単一の stream で全帯域幅容量を利用できるようにすることです。

それにもかかわらず、複数の streams (おそらく 8、16、または 32 ビット幅) を使用して帯域幅を分割することも可能です。これにより、総帯域幅が全帯域幅機能を利用できるようになります (おそらく 5-10% の低下を伴います)。

application logic で自然に動作するように、データ幅を選択する必要があります。

帯域幅能力の向上に役立つかどうかに関係なく、より広い単語インターフェイスが許可されます。たとえば、IP core の帯域幅を利用するには 64 ビットで十分ですが、リビジョン B の IP cores では 256 ビットのワード幅が許可されます。これらのデータ幅は、PCIe ブロックとのインターフェイス信号とは無関係です。

32 ビットを超えるワード幅を使用する場合、PCIe bus の本来のデータ要素は 32 ビットであるため、streams の誤った使用を防止する driver のいくつかのセーフガードは、データ幅がそれを超える場合には適用されないことに注意することが重要です。 32 ビット。たとえば、ワード幅が 64 ビットの stream に対する read() または write() への関数呼び出しには、8 の倍数の長さが必要です。 同様に、意味のある結果を得るには、64 ビット幅の stream で seek 操作によって要求される位置が 8 の倍数である必要があります。ただし、ソフトウェアはそれが 4 の倍数であることのみを強制します。

結論として、データ幅が 32 ビットを超える場合、アプリケーション ソフトウェアは、ワード幅に合わせた I/O を実行する責任がより大きくなります。ワード アライメントのルールはすべてのワード幅で同じですが、ワード幅が 32 ビットおよび 16 ビットの streams とは異なり、driver は必ずしもこれらのルールを適用しません。

4.4 Logic のリソース消費

リビジョン B/XL/XXL の IP cores は、streams の数が増加するにつれて logic の消費がわずかに急上昇する代わりに、速度と logic のわずかに低い消費のために最適化されています。

logic リソースの使用量を定量化するために、streams の数が増加する Kintex-7 用の cores が生成されました。 cores は synthesis を受け、logic の要素がカウントされました。セクション 3.3 と同様に、upstreams のベンチマーク テストは 50% であ

り、downstreams の場合も同じでした。

次の 3 つのグラフは logic の消費量を示しており、リビジョン A、B、および XL の IP cores を同じ設定で比較しています。テストされたすべての streams は 32 ビット幅でした。

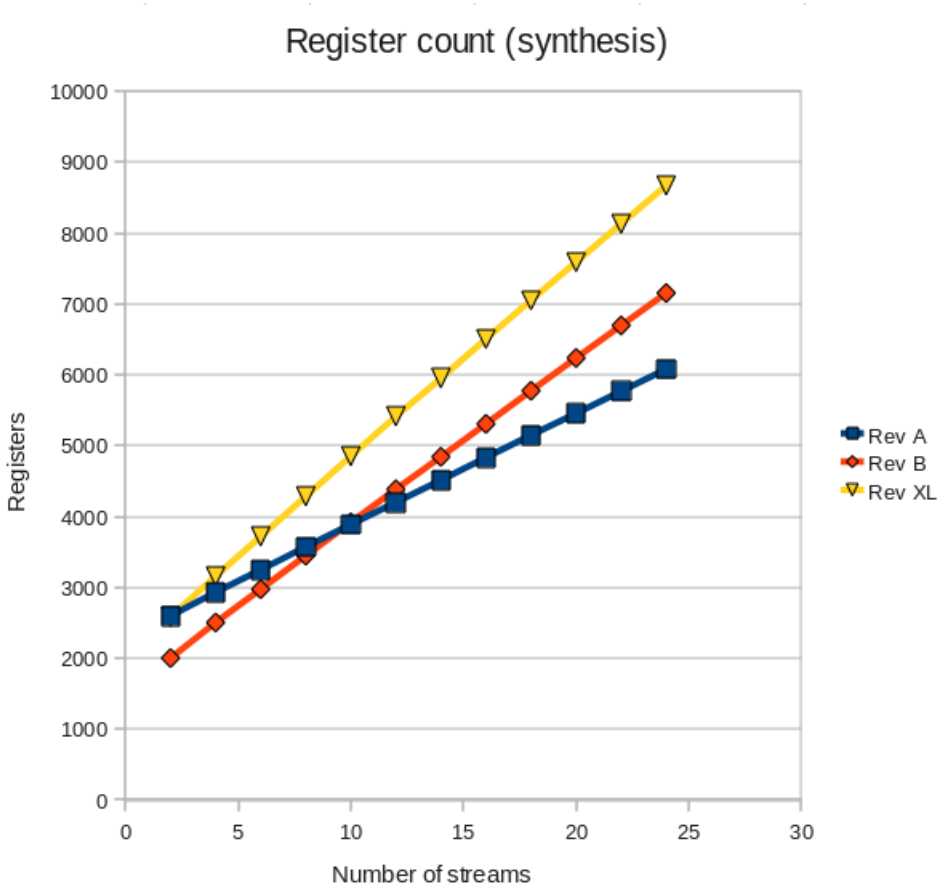
registers と LUTs の数を比較すると、streams の数が少ない場合はリビジョン B がリビジョン A よりも優れていますが、streams の数が増えるとこの利点は失われます。

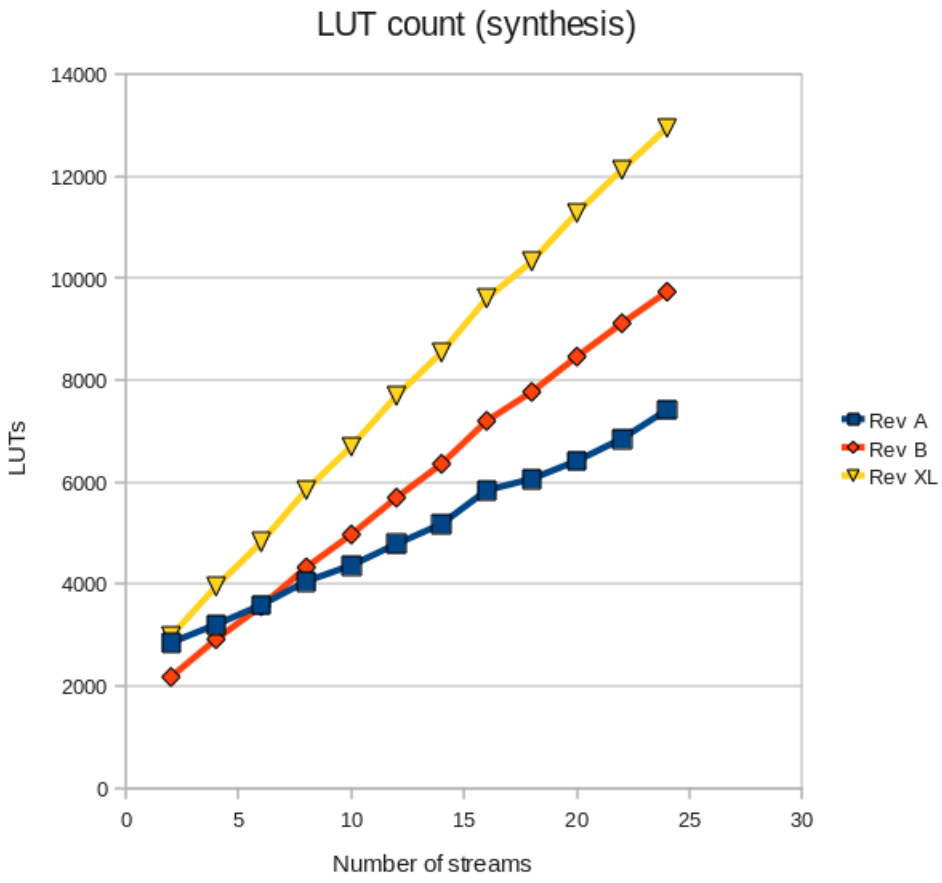
リビジョン XL と XXL は、すべてのシナリオで他の両方のリビジョンよりも多くの logic を消費します。

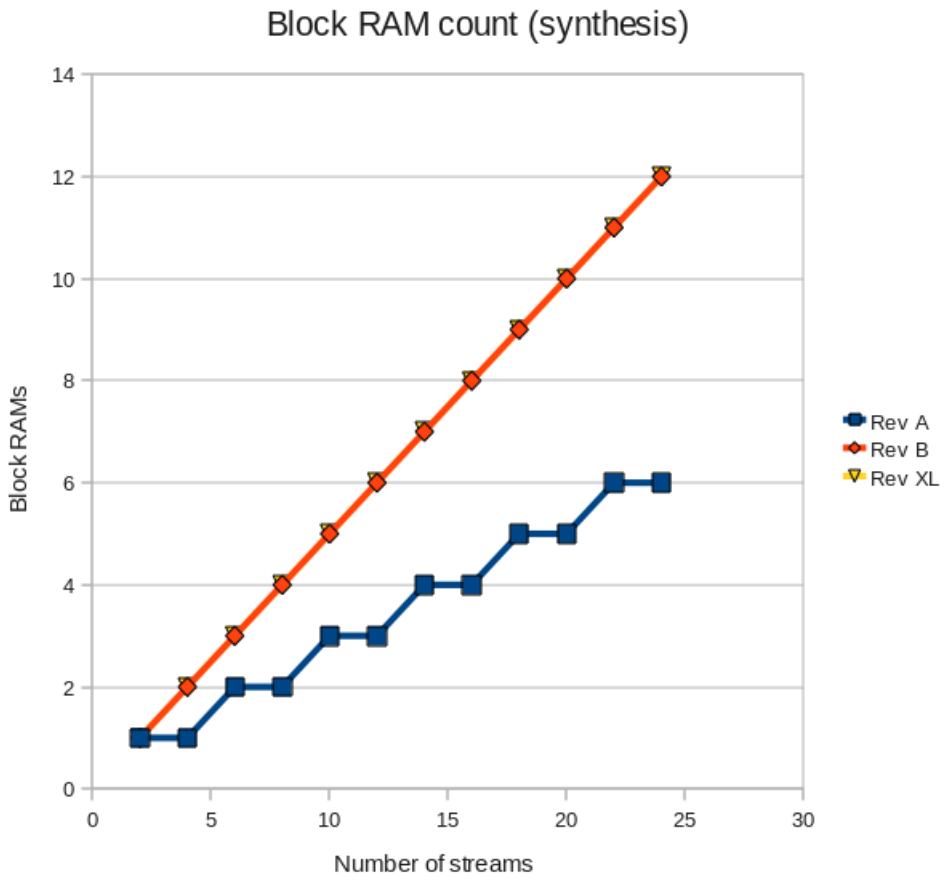
block RAMs のグラフは、リビジョン B と XL の両方が、リビジョン A と比較して 2 倍の block RAMs を消費することを示しています。

推奨される結論は、リビジョン B がほとんど常にリビジョン A より優先されるべきであるということです。 logic の消費量が反対の場合でも、リビジョン B の改良された timing はこの差を上回ります。これは、FPGA の容量と比較して、IP core の logic 消費が無視できるほとんどの実際的なシナリオに当てはまります。

一方、リビジョン XL および XXL は、logic をより多く消費し、timing constraints に関してはより困難であるため、帯域幅容量を必要とするアプリケーションにのみ選択する必要があります。


Register count (synthesis)
Registers
Number of streams
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
0
5
10
15
20
25
30
Rev A
Rev B
Rev XL


LUT count (synthesis)
14000
12000
10000
8000
6000
4000
2000
0
LUTs
0
5
10
15
20
25
30
Number of streams
Rev A
Rev B
Rev XL



4.5 host から FPGA への stream の最適な帯域幅の調整

リビジョン B、XL、および XXL を備えた IP cores では、より高いデータ レートが可能になるため、PCIe ブロックのパラメーターの適切な調整がますます重要になります。

demo bundles の PCIe ブロックは、最適なパフォーマンスが得られるようにすでにセットアップされています。ただし、PCIe bus (host のハードウェアの一部) が通常よりも長い latency でパケットをリレーする場合は、わずかな調整が必要になる場合があります。

Xilinx による FPGAs のうち、これは、Gen2 に限定された PCIe ブロックを持つ Kintex-7 または Virtex-7 で使用される、リビジョン B または XL の IP cores にのみ適用されます。リビジョン A は、改善が見られるデータ レートには達していません。

この限られたケースでは、host から FPGA への streams で意図した帯域幅を実現するために、PCIe ブロックのパラメーターを調整する必要がある場合があります。

これは、FPGA によって発行される DMA 転送の要求により、データが host から FPGA 方向に流れるために必要になる場合があります。 host は、データを送信することでこれらの要求を満たします。要求とそれらを満たすデータ送信 (completions) の間の遅延は、そのような要求に対する host の応答性に依存し、PCIe bus ごとに異なります。

PCIe bus が提供する帯域幅を有効に利用するために、複数の DMA 要求が FPGA によって並行して発行されるため、host が常に処理する要求を持つことが保証されます。ただし、アクティブなリクエストの数には制限があり、PCIe protocol のフロー制御によって課せられます。フロー制御メカニズムによって割り当てられる限られたリソースは、*completion credits* と呼ばれ、PCIe endpoint 用に構成されます。一般的に言えば、これらの数が多いほど、より多くのアクティブなリクエストが許可され、PCIe ブロックを実装するために FPGA でより多くのリソースが必要になることを意味します。

FPGA から host への方向は、FPGA が DMA 要求と一緒にデータを送信するため、credits の割り当てによって影響を受けることはほとんどありません。したがって、credits の設定を変更することによる改善の可能性はほとんどありません。影響はほとんどないためです。

demo bundles の PCIe ブロックは、x86 アーキテクチャに基づく processor を使用して、一般的なデスクトップ コンピュータで帯域幅を最適に利用できるように構成されています。非常にまれですが、host から FPGA 方向でアドバタイズされた帯域幅を達成するために、構成を変更する必要がある場合があります。

completion credits (header と data の両方) の数を増やすことで改善される場合があります。これは、PCIe block IP の構成を呼び出し、そのパラメーターを変更することによって、Vivado または Quartus で行われます。たとえば、Vivado で構成された Kintex-7 の場合、これは “Core Capabilities” タブを選択して “BRAM Configuration Options” を設定することによって行われます。 “Perf level” はすでに可能な限り最高に設定されているため、これを改善する余地はありません。ただし、“Buffering Optimized for Bus Mastering Application” を有効にすると、他のタイプの credits を犠牲にして completion credits が増加します。これにより、反対方向に悪影響を与えることなく、host から FPGA 方向の帯域幅パフォーマンスが向上する場合があります。