

はじめに

このアプリケーション・ノートでは、Arria® II GX FPGA に実装される専用 PCI Express ロジック・ブロックを紹介して、以下について説明します。

- Arria II GX デバイスにおける PCI Express MegaCore® のハード IP の実装
- DDR2 SDRAM 高性能コントローラ
- PCI Express プロトコル
- Quartus® II ソフトウェアによる PCI Express MegaCore の実装
- ModelSim® ソフトウェアによる複数の IP モジュールの検証

PCI Express は、コンポーネントにはるかに高いレベルの効率で通信する能力を提供するポイント・ツー・ポイント高速シリアル I/O インタフェースです。このデザインは *PCI Express Base Specification, Rev 2.0* に準拠しています。

このデザインは以下の間のサンプル・インタフェースを表します。

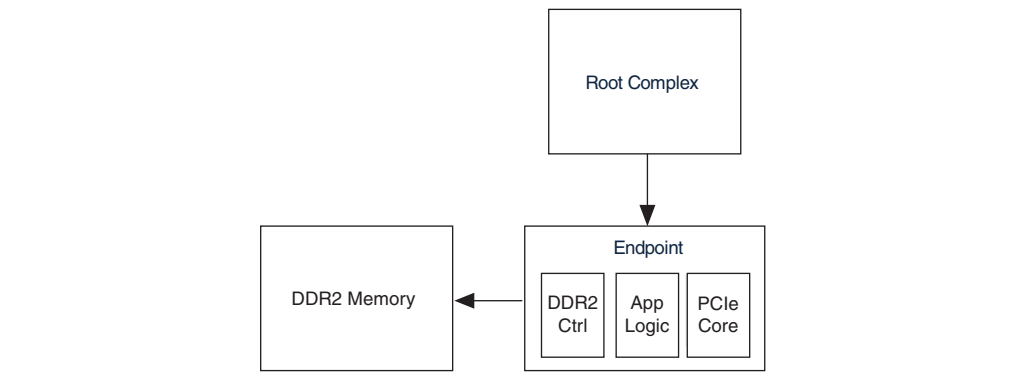
- PCI Express プロトコルを使用して Arria II GX FPGA にコマンドをドライブするデバイス (ルート・コンプレックス)
- Arria II GX FPGA (エンドポイント)
- 外部 DDR2 SDRAM メモリ

Altera® PCI Express-to-DDR2 リファレンス・デザインは Altera PCI Express MegaCore ファクションとインタフェースする一般的なユーザー・アプリケーションの例です。

概要

このデザインでは、ルート・コンプレックスは FPGA エンドポイントとインタフェースします。FPGA エンドポイントは外部 DDR2 SDRAM とインタフェースします (図 1 を参照)。

図 1. ルート・コンプレックス、FPGA エンドポイント、および外部 DDR2 SDRAM



リファレンス・デザインは以下のブロックを含む FPGA 上のエンドポイントを実装します。

- ルート・コンプレックスとインタフェースする PCI Express MegaCore
- DDR2 SDRAM メモリにインタフェースする DDR2 SDRAM 高性能メモリ・コントローラ
- DDR2 SDRAM 高性能メモリ・コントローラと PCI Express MegaCore 間をインタフェースするロジック

ルート・コンプレックスは PCI Express リンクを通してエンドポイントにリードおよびライト・コマンドを発行します。また、エンドポイントはダイレクト・メモリ・アクセス (DMA) リードおよびライトを通してルート・コンプレックスにリードおよびライトを開始することができます。これはリードおよびライト・トランザクションによって行われます。

トランザクション・タイプ

パケットは 1 つのデバイスから別のデバイスに転送される時、トランザクションが発生します。パケットはヘッダーおよびデータで構成されています。ヘッダーではトランザクションの種類、含まれたデータ、およびパケットに関する他の関連詳細について情報が含まれています。データは、コンポーネントの間に転送されるべき情報です。この場合、コンポーネントはルート・コンプレックス、エンドポイント、および DDR2 SDRAM です。この場合のパケットはトランザクション・レイヤ・パケット (TLP) です。

リファレンス・デザインは次の TLP を処理します。

- メモリ・リード要求 (MRd)
- メモリ・ライト要求 (MWr)
- 完了データ (CpID)

始動デバイスは、ターゲット・デバイスに MRd を転送することができます。それによって、ターゲット・デバイスは、要求されるデータを含めて、CpID で応答します。また、始動デバイスはデータを含む MWr を転送することもできます。

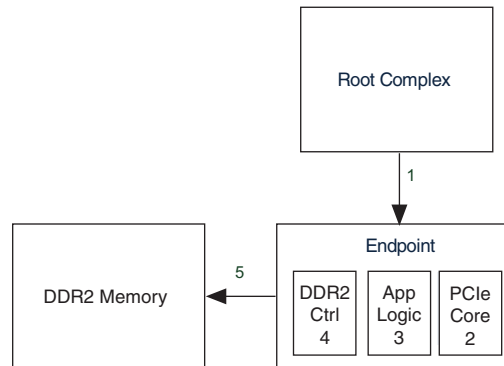
以下の 4 種類のトランザクションはルート・ポートと FPGA エンドポイント間の PCI Express リンクにまたがって開始されます。

- ルート・コンプレックスが開始したメモリのライト
- ルート・コンプレックスが開始したメモリのリード
- エンドポイントが開始した DMA ライト
- エンドポイントが開始した DMA リード

ルート・コンプレックスが開始したメモリのライト

ルート・コンプレックスはライト・トランザクションを開始することによって、データをエンドポイントに転送します。このトランザクションは MWr タイプ・トランザクションおよび転送トランザクションです。これは、リターン通信がないことを意味します (図 2 を参照)。

図 2. ルート・コンプレックスが開始したメモリのライト



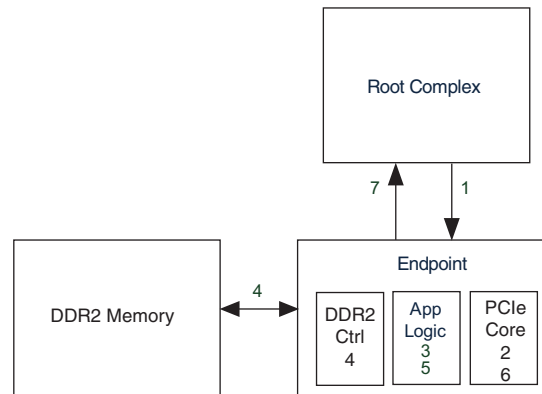
以下のイベントが図 2 に発生します。

1. ルート・コンプレックスは MWr トランザクションを FPGA に転送します。
2. PCI Express のハード IP はパケットを受信して、すべての物理層およびデータ・リンク層サービスを実行します。また、トランザクション層においてフロー・コントロール・サービスを実行します。
3. PCI Express のハード IP は『*PCI Express Compiler User Guide*』に規定されたフォーマットでの TLP のあるアプリケーション・ロジックを提供します。
4. アプリケーション・ロジックは TLP をデコードして、DDR2 SDRAM 高性能メモリ・コントローラにデータを転送します。
5. データが DDR2 SDRAM メモリに書き込まれます。

ルート・コンプレックスが開始したメモリのリード

ルート・コンプレックスはリード要求トランザクションを開始することによって、エンドポイントからデータを要求します。このトランザクションは MRd タイプ・トランザクションおよび非転送トランザクションです。これは、リターン通信が完了データ (CplID) タイプ・トランザクションの形で発生することを意味します。(図 3 を参照)。

図 3. ルート・コンプレックスが開始したメモリのリード



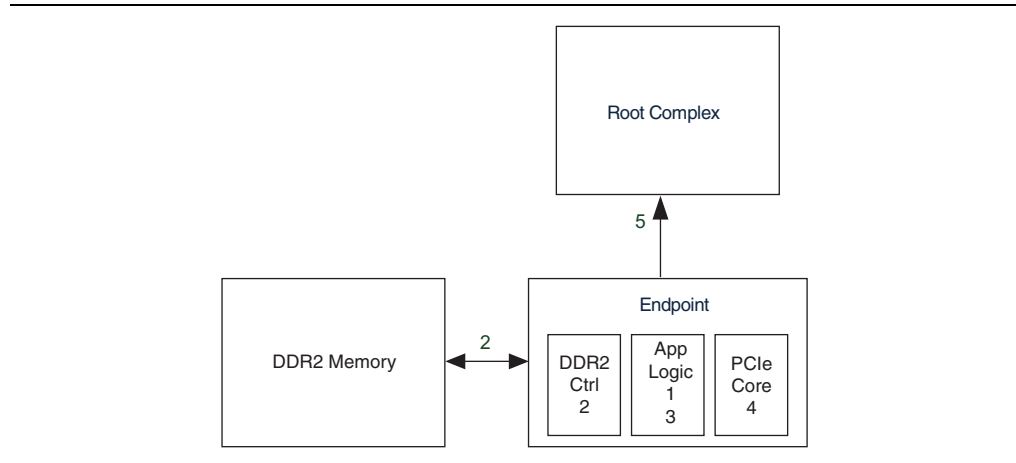
以下のイベントが図 3 に発生します。

1. ルート・コンプレックスは MRd トランザクションを FPGA に転送します。
2. PCI Express のハード IP はパケットを受信して、すべての物理層およびデータ・リンク層サービスを実行します。また、トランザクション層においてフロー・コントロール・サービスを実行して、『*PCI Express Compiler User Guide*』に規定されたフォーマットでアプリケーション・ロジックに TLP を提供します。
3. アプリケーション・ロジックは TLP をデコードして、DDR2 SDRAM 高性能メモリ・コントローラからのメモリ・リード要求を開始します。
4. DDR2 SDRAM メモリは DDR2 SDRAM 高性能メモリ・コントローラでデータを取得します。
5. アプリケーション・ロジックはデータのヘッダーを作成して、完了データ TLP が集合され、転送用に PCI Express のハード IP に転送されます。
6. PCI Express のハード IP は残りすべてのトランザクション・サービスを実行して、すべてのデータ・リンク層および物理層サービスを実行します。
7. FPGA はルート・コンプレックスに CplID パケットを転送します。

エンドポイントが開始した DMA ライト

エンドポイントはライト・トランザクション (DMA ライトとも呼ばれる) を開始することによって、ルート・コンプレックスにデータを転送します。このトランザクションは MWr タイプ・トランザクションおよび転送トランザクションです。これは、リターン通信がないことを意味します (図 4 を参照)。

図 4. エンドポイントが開始した DMA ライト



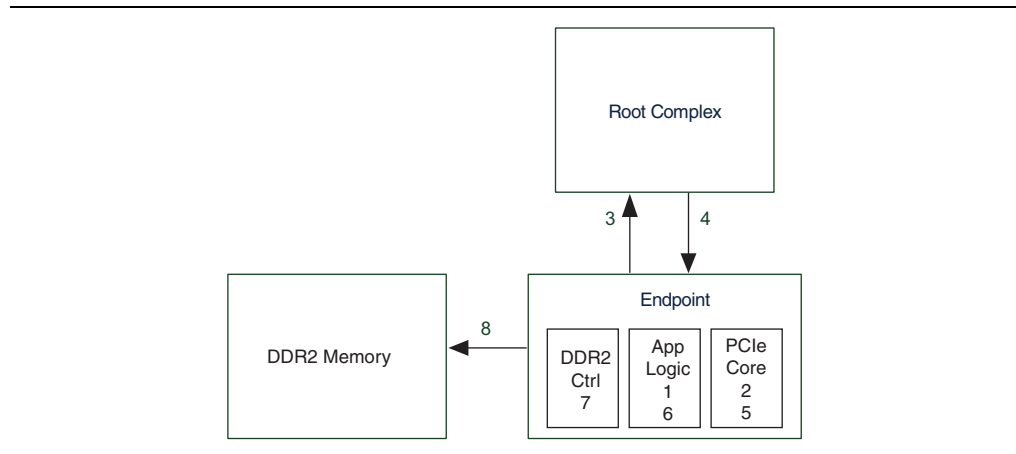
以下のイベントが図 4 に発生します。

1. アプリケーション・ロジックはルート・コンプレックスへのライト要求トランザクションの発生を待っているかどうかを確認するために、DMA レジスタを読み出します。次に、アプリケーション・ロジックはデータを転送するために、DDR2 SDRAM 高性能コントローラにリード要求を転送します。
2. DDR2 SDRAM 高性能メモリ・コントローラを通して DDR2 SDRAM メモリからデータを取得します。
3. データのヘッダーが作成されて、アプリケーション・ロジックでメモリ・ライト TLP が集合され、送信用に PCI Express のハード IP に転送されます。
4. PCI Express のハード IP は残りすべてのトランザクション・サービスを実行して、すべてのデータ・リンク層および物理層サービスを実行します。
5. FPGA はルート・コンプレックスに MWr パケットを転送します。

エンドポイントが開始した DMA リード

エンドポイントはリード・トランザクション (DMA リードとも呼ばれる) を開始することによって、ルート・コンプレックスからデータを要求します。このトランザクションは MRd タイプ・トランザクションおよび非送信トランザクションです。これは、リターン通信が完了データ (CplD) タイプ・トランザクションの形で発生することを意味します (図 5 を参照)。

図 5. エンドポイントが開始した DMA リード



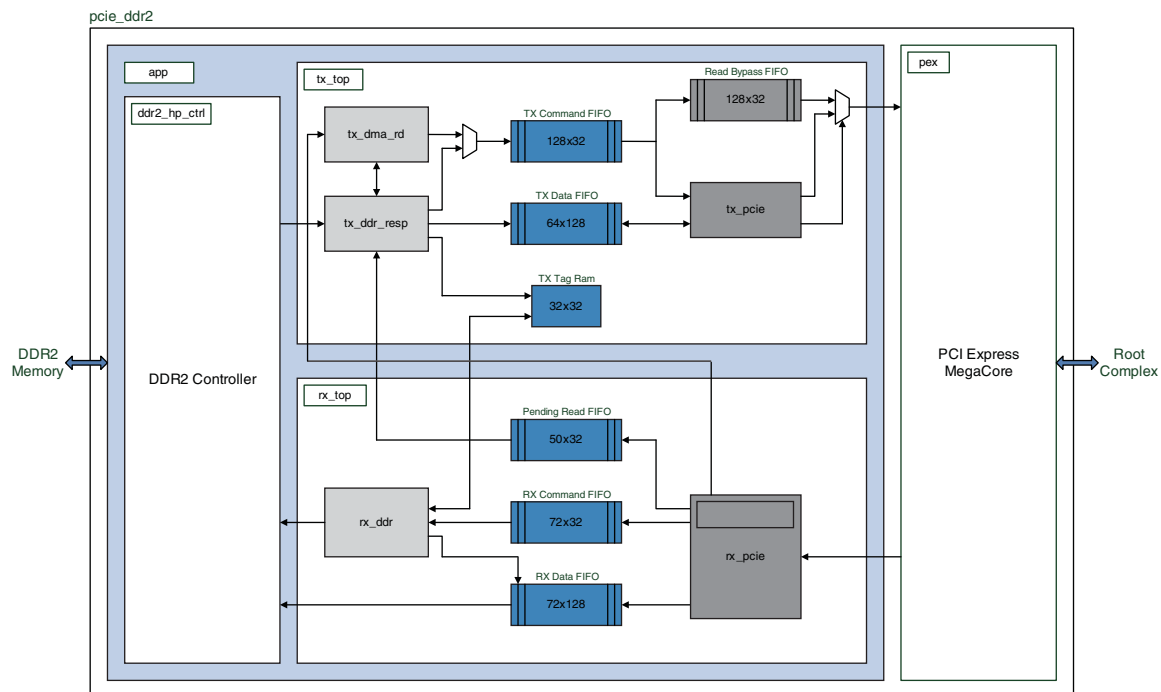
以下のイベントが図 5 に発生します。

1. アプリケーション・ロジックはルート・コンプレックスへのリード要求トランザクションの発生を待っているかどうかを確認するために、DMA レジスタを読み出します。次に、アプリケーション・ロジックはメモリ・リード TLP を集合して、送信用に PCI Express のハード IP に転送されます。
2. PCI Express のハード IP は残りすべてのトランザクション層サービスを実行して、すべてのデータ・リンク層および物理層サービスを実行します。
3. FPGA はルート・コンプレックスに MRd パケットを送信します。
4. ルート・コンプレックスは FPGA に CplD パケットを送信します。
5. PCI Express のハード IP は完了パケットを受け取って、すべてのデータ・リンク層および物理層サービスを実行します。また、トランザクション層においてフロー・コントロール・サービスを実行します。
6. PCI Express のハード IP は *「PCI Express Compiler User Guide」* に規定されたフォーマットでアプリケーション・ロジックに完了データ TLP を提供します。
7. アプリケーション・ロジックは TLP をデコードして、DDR2 SDRAM 高性能メモリ・コントローラにデータを送信します。
8. データが DDR2 SDRAM メモリに書き込まれます。

リファレンス・デザインの概要

図 6 に、リファレンス・デザインの上位レベルのブロック図を示します。リファレンス・デザインには、PCI Express MegaCore ファンクション、DDR2 SDRAM コントローラ、およびその両方間のリファレンス・デザインのアプリケーション・ロジックで構成されます。このアプリケーション・ロジックの例では、TX アプリケーション・ロジック・ブロック、(tx_top)、および RX アプリケーション・ロジック・ブロック、(rx_top) が含まれます。

図 6. リファレンス・デザインの上位レベルの図



この項では、以下のリファレンス・デザイン・コンポーネントの概要を提供します。

- **PCI Express MegaCore** ファンクション — 高速シリアル・リンクを通して、ルート・コンプレックスにデータを送受信します。
- **DDR2 SDRAM** コントローラ — DDR2 SDRAM 外部メモリを通してデータを送受信します。
- **RX** アプリケーション・ロジック — データを受信するために、PCI Express MegaCore および DDR2 SDRAM コントローラとインタフェースします。
- **TX** アプリケーション・ロジック — データを送信するために、PCI Express MegaCore および DDR2 SDRAM コントローラとインタフェースします。
- **DMA** エンジン — トランザクションを開始するために、DMA レジスタをモニタします。ロジックが TX アプリケーション・ロジックおよび RX アプリケーション・ロジックに分散されています。

ロジックは TLP ヘッダーおよび TLP データに対して、異なるデータ・フローを持っています。TLP ヘッダーは、RX アプリケーション・ロジックでデコードされるか、または TX アプリケーション・ロジックで集合されます。ヘッダーからの情報はコマンドに推定されます。一般に、コマンドはロジックに使用されて、128 ビット FIFO で保存されます。これは 4 つのダブル・ワード (DW) TLP ヘッダー (128 ビット) または 3 つの DW ヘッダー (96 ビット) を対応するためです。1 つの DW は 32 ビットです。

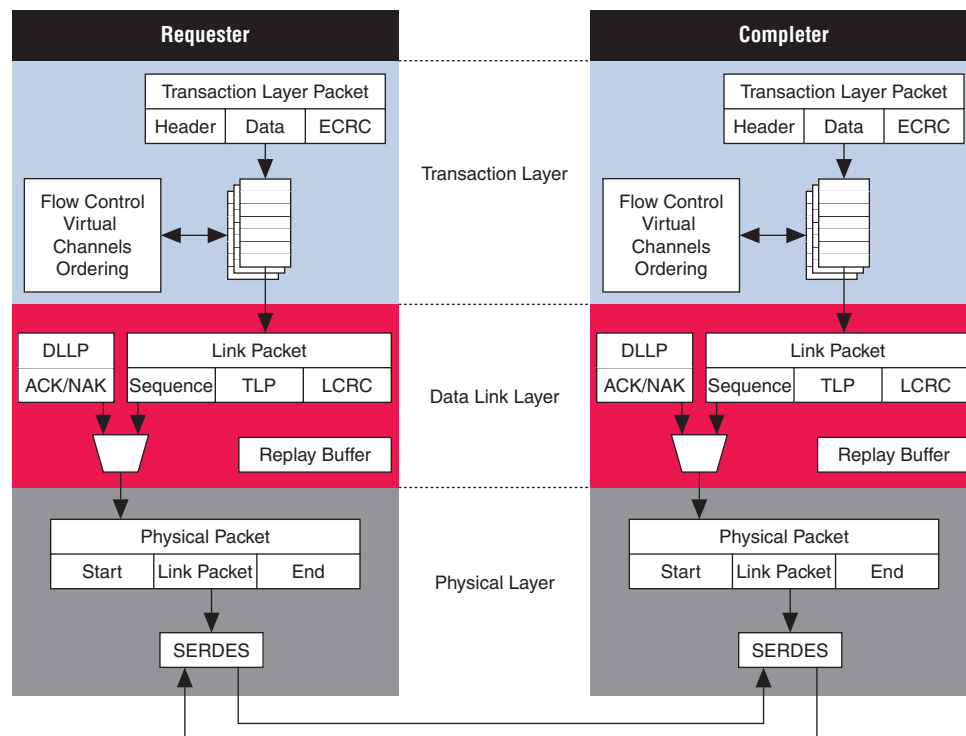
PCI Express MegaCore ファンクション

PCI Express MegaCore ファンクションはシステム・アプリケーション・ロジックの 1 つの側に接続します。反対側では、それが PCIe リンクに接続する PHY に接続します。PCI Express MegaCore ファンクションは *PCI Express Base Specification Rev. 2.0* で指定されているとおり、物理層、データ・リンク層、およびトランザクション層の PCIe ファンクションを実行します。

このリファレンス・デザインはハード IP の実装にある PCI Express MegaCore を Avalon®-ST インタフェースでインスタンス化します。コアが MegaWizard™ Plug-In Manager フローを使用してインスタンス化されます。

ハード IP の実装にある PCI Express MegaCore はトランザクション層の用に物理層、データ・リンク層およびフロー・コントロールに関する仕様を処理します (図 7 を参照)。

図 7. ハード IP の実装における PCI Express MegaCore



トランザクション層の出力は、PCI Express プロトコルに従うパッケージ (TLP と呼ばれる) です。TLP はヘッダーおよびデータ情報に分割されています。TLP はヘッダーまたはデータのどちらの部分であるダブル・ワード (DW) に分割されます (図 8 を参照)。

図 8. TLP ヘッダーおよびデータ情報



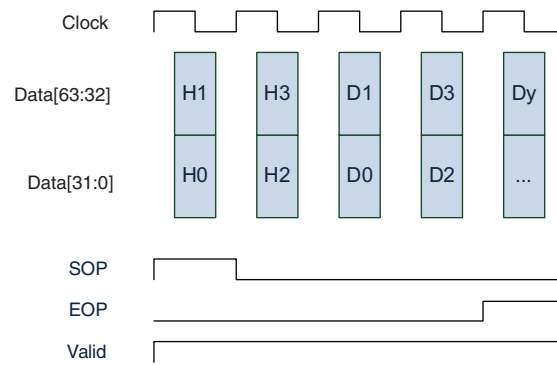
TLP は 3DW ヘッダーまたは 4DW ヘッダーを持つことができます。ペイロード・レングスは DW のレングス・フィールドにある TLP ヘッダーで指定されます (図 9 を参照)。

図 9. ダブル・ワード・ヘッダーおよびペイロード・レングス

	+0								+1								+2								+3								
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
H0	Byte 0	0	Fmt		Type				0	TC		0		0		0		TD	EP	Att		0		0		Length							
H1	Byte 4	Fields depend on type of TLP																															
H2	Byte 8	Fields depend on type of TLP																															
H3	Byte 12	Only required for some TLPs. Fields depend on type of TLP.																															

TLP が FPGA ロジックに実装されるアプリケーション層に供給されます。TLP は以下のフォームにあり、 図 10 に示すように、コントロール信号によって付属されます。TLP 情報が data にあります。各 TLP の先頭には SOP (Start-Of-Packet) が付属しています。EOP (End-of-Packet) は TLP の終了を表し、そして valid 信号はクロックされる必要があるデータの期間に High になり、TLP 情報を含みます (図 10 を参照)。

図 10. アプリケーション層に提供される TLP



TLP の性質は PCI Express Base Specification, Rev 1.1 のセクション 2.1 および「[PCI Express Compiler User Guide](#)」のセクション 5 に説明されています。

PCI Express コアはそれに参照する最大 8 ベース・アドレス・レジスタ (BAR) を持つシステム・アドレス空間を予約します。各 BAR は参照されるメモリ・セグメントの開始アドレスを格納します。一般に、標準の PCI Express アプリケーションはリンクに関する標準的基準として BAR0 と BAR1 を使用します。代替使用のために追加の BAR を指定することができます。

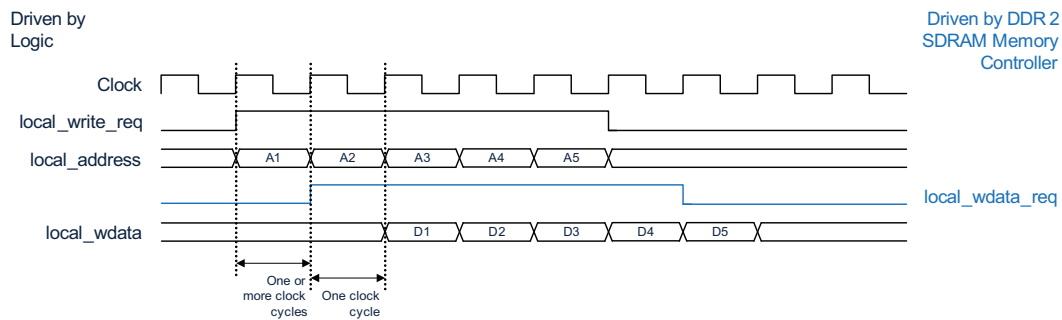
このデザインでは、BAR0 および BAR1 は DDR2 SDRAM メモリを参照するのに使用されます。BAR2 はアプリケーション・ロジックによるモニタされた DMA レジスタを参照するのに使用されます。これで、ルート・コンプレックスは BAR2 に書き込むことによって、DMA レジスタに書き込むことができます。したがって、ルート・コンプレックスからエンドポイント・トランザクションを開始することも可能です。それはこのデザインが DMA レジスタに書き込み方法です。

DDR2 SDRAM コントローラ

DDR2 SDRAM コントローラはローカル・サイドの TX と RX アプリケーション・ロジックかつシステム・サイドのオフチップ DDR2 SDRAM メモリとインタフェースします。コントローラは、RX と TX アプリケーション・ロジックと通信するのにインタフェース信号を使用します。

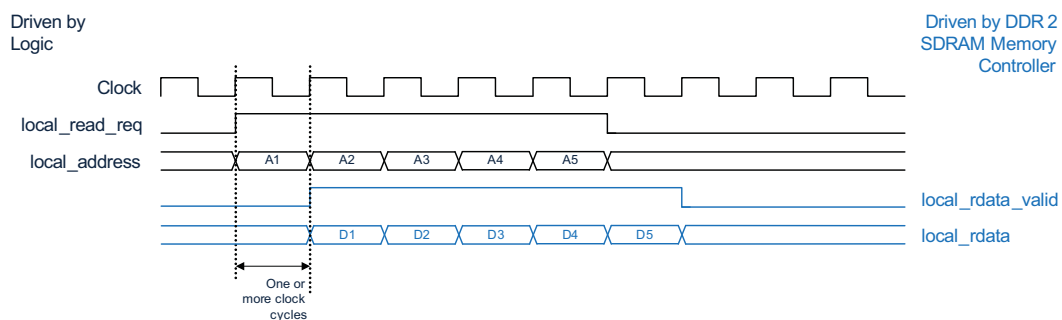
DDR2 SDRAM メモリに書き込んでいるとき、RX アプリケーション・ロジック (rx_ddr) が local_address 信号に書き込まれるアドレスを供給しているのに、local_write_req 信号をアサートします。次に、DDR2 SDRAM 高性能メモリ・コントローラは local_wdata_req をアサートして、rx_ddr モジュールは 1 クロック・サイクル以内に、local_wdata 信号にデータを供給します (図 11 を参照)。

図 11. DDR2 SDRAM メモリへの書き込み



DDR2 SDRAM メモリから読み出しているとき、RX アプリケーション・ロジック (rx_ddr) が local_address 信号で読み出されるアドレスを供給しているのに、local_read_req 信号をアサートします。しばらくすると、DDR2 SDRAM 高性能メモリ・コントローラが local_rdata 信号上の TX アプリケーション・ロジック (tx_ddr_resp) にデータを返送して、local_rdata_valid 信号をアサートします (図 12 を参照)。

図 12. DDR2 SDRAM メモリからの読み出し



RX アプリケーション・ロジック

RX アプリケーション・ロジック・ブロックは PCI Express MegaCore を通してルート・コンプレックスから TLP パケットを受信します。RX アプリケーション・ロジック・ブロックは TLP パケットをデコードし、そしてトランザクションのタイプに基づいて、DDR2 メモリから情報を取り込んで要求を終了することができるように、DDR2 メモリにデータを書き込むか、または TX アプリケーション・ロジック・ブロックにリード要求情報を送信します。また、RX アプリケーション・ロジック・ブロックは TX アプリケーション・ロジックによってモニタされる DMA レジスタを保持します。

RX アプリケーション・ロジック・コンポーネント

RX アプリケーション・ロジックには、以下のコンポーネントで構成されます。

- RX PCIe パケット受信ブロック (rx_pcie)
- RX コマンド FIFO バッファ (72 × 32)
- RX データ FIFO バッファ (72 × 128)
- リード待ちの FIFO バッファ (50 × 32)
- RX DDR2 ライト・ブロック (rx_ddr)

rx_pcie モジュールは PCI Express MegaCore とインタフェースして、rx_ddr モジュールは DDR2 SDRAM 高性能コントローラとインタフェースします。FIFO バッファは、コンポーネントによってモニターされて、転送されたデータをコンポーネントの間に保持します。また、FIFO バッファは PCIe および DDR2 クロック・ドメイン通っているデータを同期します。

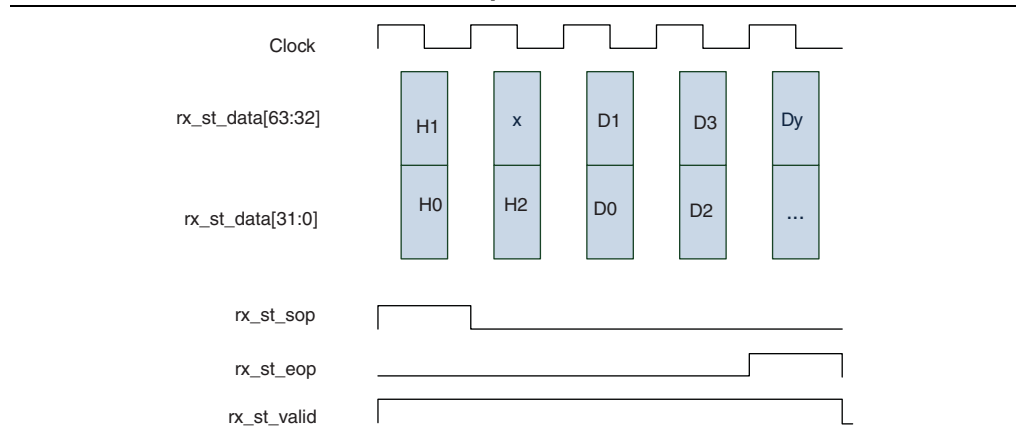
RX PCIe パケット受信ブロック (rx_pcie)

rx_pcie ブロックが PCI Express MegaCore ファンクションから TLP を受信します。それは PCI Express MegaCore ファンクションからシステムサイド信号をデコードします。要求のタイプに基づいて、rx_pcie ブロックは DDR2 SDRAM にデータを書き込むか、リード待ちの FIFO にリード要求を提出するか、または DMA レジスタに書き込むかを決定します。

パケットにはパケット開始 SOP (Start of Packet)、パケット終了 EOP (End of Packet)、およびデータ有効 (valid) 信号で付属されます。ヘッダーが抽出され、そしてデータがある場合、それは RX データ FIFO バッファに書き込まれます。

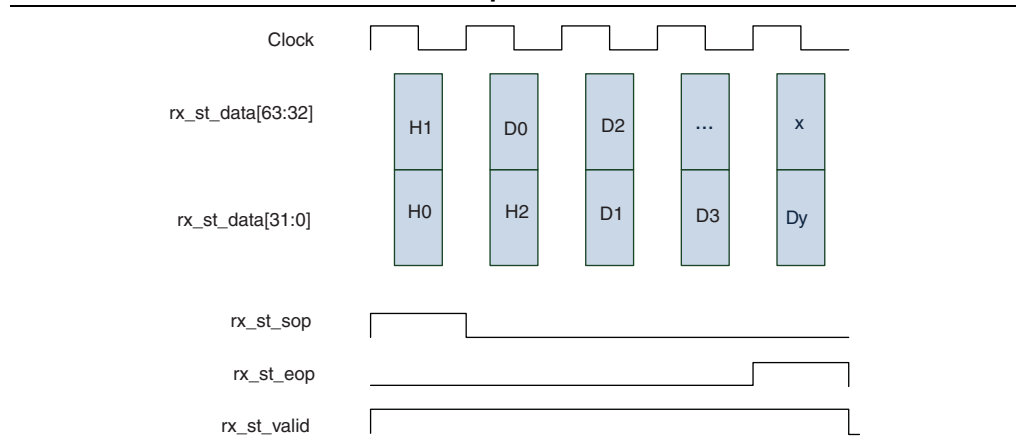
データがアラインメントまたは非アラインメントされることができます。アラインメントされたデータは D0 から始まって、64 ビット・バスの下位 32 ビットから開始することを意味します (図 13 を参照)。

図 13. RX PCIe パケット受信ブロック (rx_pcie)— アラインメント



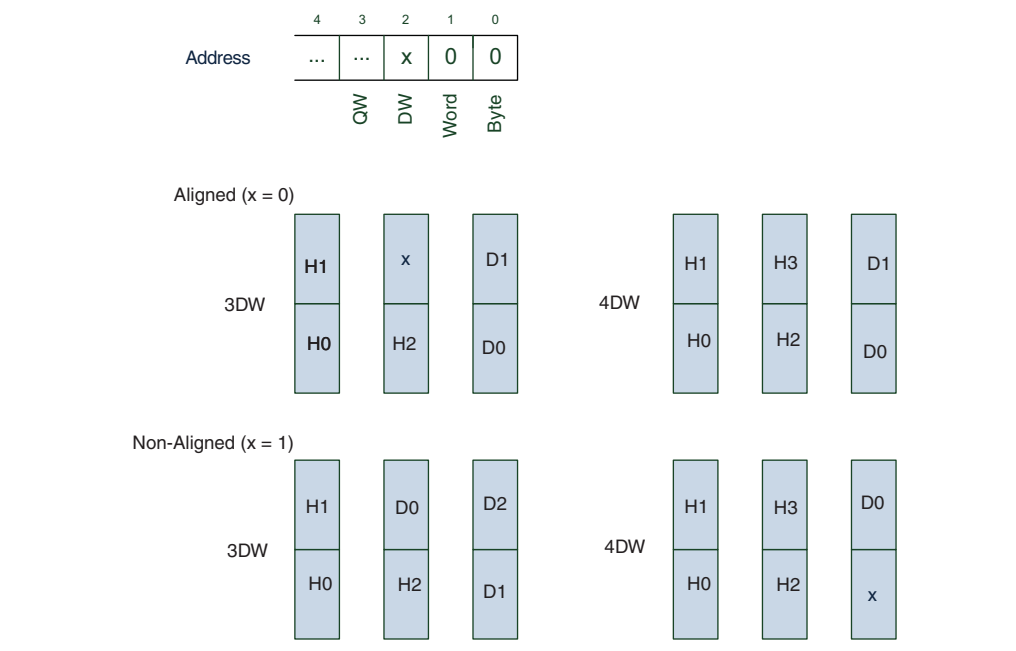
非アラインメントされたデータは D0 が 64 ビット・バスの上位 32 ビットで開始することを意味します (図 14 を参照)。

図 14. RX PCIe パケット受信ブロック (rx_pcie)— 非アラインメント



データがアラインメントまたは非アラインメントされたかを確認するには、アドレスのビット 2 を検査します。address[2] は 0 の場合、データがアラインメントされます。address[2] は 1 の場合、データが非アラインメントされます (図 15 を参照)。

図 15. アラインメントおよび非アラインメントされるデータ



rx_pcie モジュールが DMA レジスタへのアクセスを管理して、DMA レジスタをモニタします。BAR2 をターゲットとする場合、受信されたトランザクションが DMA レジスタをターゲットとします。dma_start 信号によって命令されるとき、または DMA トランザクションを開始する準備ができているときには、rx_pcie ブロックは DMA レジスタからの情報を使用して、1 つ以上の要求を生成します。これらの要求は次に、RX コマンド FIFO バッファおよび RX リード待ちの FIFO バッファにロードされます。

RX コマンド FIFO バッファ (72 × 32)

RX コマンド FIFO バッファは rx_pcie ブロックからリフォーマットされたコマンドを受信します。rx_ddr ブロックは RX コマンド FIFO バッファをモニタして、空ではない場合、コマンドを読み出します。また、RX コマンド FIFO バッファはコマンドを PCI Express クロック・ドメインから DDR2 クロック・ドメインに変換して、コマンドを rx_ddr ブロックに転送します。

RX データ FIFO バッファ (72 × 128)

RX データ FIFO バッファは DDR2 SDRAM メモリに転送するために rx_pcie ブロックからデータを受け取ります。RX DDR2 ブロック、rx_ddr は、RX データ FIFO バッファからのデータを読み出します。

FIFO バッファからの上位 8 ビットは 64 ビット・データ・バスのバイト・イネーブルです。RX データ FIFO バッファからのリード動作は常に 72 ビットのトランクで行われます (DDR2 SDRAM メモリに 64 ビットのみ格納される)。

リード待ちの FIFO バッファ (50 × 32)

PCI Express MegaCore ファンクションからリード要求を受信しているとき、RX アプリケーション・ロジックは DDR2 SDRAM メモリからどのくらいの完了データを予期するか、および準備ができており、リード完了データをどこに送信するかに関する詳細も受け取ります。RX アプリケーション・ロジックは RX コマンド FIFO バッファによって受信される要求の詳細をリード待ちの FIFO バッファに格納します。tx_dds_resp ブロックはリード待ちの FIFO バッファから読み出して、情報を使用して DDR2 SDRAM コントローラから戻るリード完了データを追跡します。

RX DDR2 インタフェース・ロジック (rx_dds)

RX DDR2 インタフェース・ロジックは RX コマンド FIFO からコマンドを取り出します。RX コマンド FIFO 情報に基づいて、rx_dds block は DDR2 コントローラ信号を生成して、DDR2 リードおよびライト要求を要求します。DDR2 ライト要求に対して、rx_dds ブロックは TX リード・タグ RAM で動作して、どこに DDR2 SDRAM メモリのデータを書き込むかを決定します。

TX アプリケーション・ロジック

TX アプリケーション・ロジック・ブロックは、PCI Express を使用することでルート・コンプレックスに送信するように TLP パケットを生成する役割を持っています。また、DDR2 SDRAM コントローラ・インタフェースを制御して、DDR2 SDRAM からデータを受信します。さらに、それは DMA レジスタと交信して、TX DMA リードおよびライトを生成します。そして、PCI Express を使用することでルート・コンプレックスに送信します。

TX アプリケーション・ロジック・コンポーネント

TX アプリケーション・ロジックは tx_top で以下のコンポーネントで構成されます。

- TX DDR2 リード・ブロック (tx_dds_resp)
- TX DMA 要求生成ブロック (tx_dma_rd)
- TX コマンド FIFO バッファ (128 × 32)
- TX データ FIFO バッファ (64 × 128)
- TX リード・タグ RAM
- TX PCIe パケット生成ブロック (tx_pcie)
- リード・バイパス FIFO バッファ (128 × 32)

tx_pcie モジュールは PCI Express MegaCore とインタフェースします;tx_dds_resp モジュールは DDR2 SDRAM 高性能コントローラとインタフェースして、tx_dma_rd モジュールは DMA レジスタをモニタします。FIFO バッファは、コンポーネントによってモニターされて、転送されたデータをコンポーネントの間に保持します。また、FIFO バッファは PCIe および DDR2 クロック・ドメインを横断するデータを同期します。

TX DDR2 リード・ブロック (tx_ddr_resp)

TX DDR2 リード応答ステート・マシン (tx_ddr_resp) はリード待ちの要求のための RX リード待ちの FIFO バッファをチェックします。リード待ちの要求がある場合、tx_ddr_resp は RX リード待ちの FIFO からの情報を使用して DDR2 SDRAM コントローラから予期するデータ量を決定します。また、使用可能なとき、DDR2 データの送信する時間および場所を決定します。さらに、tx_ddr_resp モジュールは DDR2 SDRAM コントローラ・システムサイド・データ・バスおよびデータ有効信号をモニタします。

TX DMA 要求生成ブロック (tx_dma_rd)

TX DMA 要求生成ブロック (tx_dma_rd) は RX アプリケーション・ロジックにある DMA レジスタをモニタします。dma_start によって管理され、および DMA トランザクションを開始する準備ができており、tx_dma_rd ブロックは DMA レジスタからの情報を使用して、1 つ以上の要求を生成します。そして、それらを TX コマンド FIFO に配置します。

TX コマンド FIFO バッファ (128 × 32)

TX コマンド FIFO バッファは tx_dma_rd ブロックおよび tx_ddr_resp ブロックによって生成されるコマンドを格納します。コマンドが PCIe TLP ヘッダー・フォーマットであり、最大 4DW のヘッダーまで持つように、128 ビットの幅があります。ヘッダーが 3DW のヘッダーである場合、最後の DW は予約されていると見なされません。

DDR2 クロック・ドメインから PCIe クロック・ドメインまで横断した後に、tx_pcie ブロックは、TX コマンド FIFO バッファから PCIe コマンドを受け取ります。

TX データ FIFO バッファ (64 × 128)

TX データ FIFO バッファは DDR2 SDRAM コントローラから PCI Express MegaCore ファンクションまで転送されるデータにバッファ・スペースを提供します。ルート・コンプレクスが開始したりリード完了またはエンドポイントが開始したライト要求の時に、DDR2 SDRAM コントローラが要求されるデータを返送します (データは FIFO バッファに入ります)。

TX DDR2 応答コントロール・ブロック (tx_ddr_resp) がこのバッファにライト動作を制御します。TX PCIe コントロール・ブロック (tx_pcie) がこのバッファからリード動作を制御します。

パケットを tx_pcie PCIe コントローラに送信する前に、必要な場合、TLP にすべての DW を格納できるのを確認するために、TX データ FIFO バッファのサイズは 64 × 128 ビットです。RX データ FIFO バッファとは異なり、FIFO バッファは常に 64 ビット幅バッファとして動作します。この FIFO バッファからのリードまたはライトは常に 1 度に 64 ビットが発生します。

TX リード・タグ RAM

TX リード・タグ RAM は TX リード要求情報を格納します。保存された情報に基づいて、それが rx_ddr ブロックで動作して、DDR2 SDRAM に書き込まれる準備ができているとき、リード完了データの送信先を決定します。

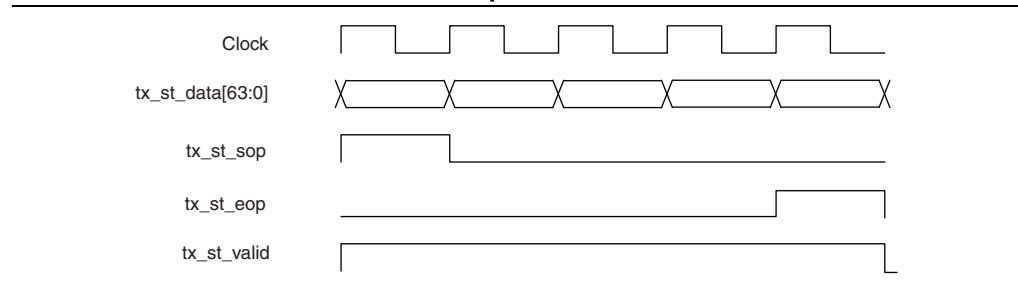
TX PCIe パケット生成ブロック (tx_pcie)

TX PCIe パケット生成ブロックは TX コマンド FIFO および TX リード・バイパス FIFO をモニタします。FIFO が空ではない場合、tx_pcie ブロックはそこからコマンドを取り出します。コマンド情報に基づいて、tx_pcie ブロックは、tx_pcie ブロックは、適切な TLP を発生して、トランザクションを開始します。そして、TLP を PCI Express MegaCore に送信します。

PCI Express Base Specification, Rev 2.0 で規定されるように、コマンド FIFO およびリード・バイパス FIFO のコマンドが TLP ヘッダーの形式です。

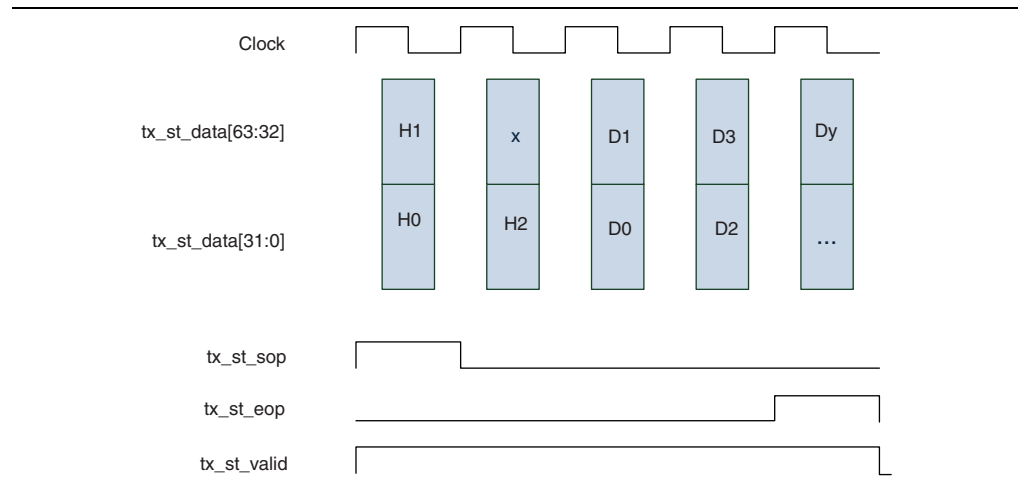
TLP の生成に加えて、tx_pcie ブロックは TLP が付属するパケット開始「start of packet」(tx_st_sop)、パケット終了「end of packet」(tx_st_eop)、およびデータ有効 (tx_st_valid) 信号も生成します (図 16 を参照)。

図 16. TX PCIe パケット生成ブロック (tx_pcie)



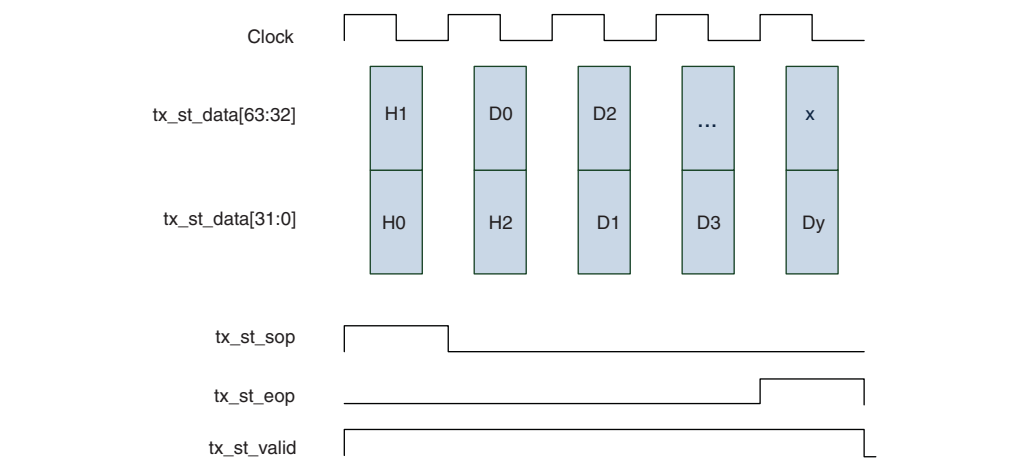
このアプリケーションでは、TLP はメモリ・リード、メモリ・ライト、および完了パケットを生成します。データがアラインメントまたは非アラインメントされることがあります。アラインメントされるデータは D0 から始まって、64 ビット・バスの下位 32 ビットで開始することを意味します (図 17 を参照)。

図 17. TX PCIe パケット生成ブロック — アラインメントされるデータ



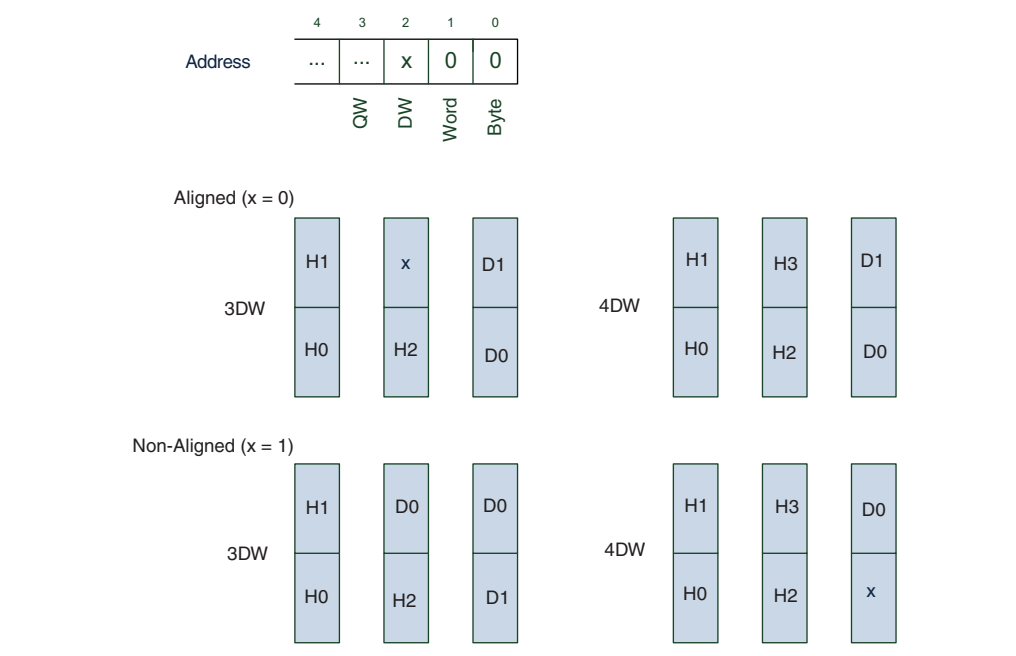
非アラインメントされるデータは D0 が 64 ビット・バスの上位 32 ビットで開始することを意味します (図 18 を参照)。

図 18. TX PCIe パケット生成ブロック — 非アラインメントされるデータ



データがアラインメントまたは非アラインメントされるかを確認するには、アドレスのビット 2 を検査します。address [2] は 0 の場合、データがアラインメントされます。address [2] は 1 の場合、データが非アラインメントされます (図 19 を参照)。

図 19. アラインメントおよび非アラインメントされるデータ



リード・バイパス FIFO バッファ (128 × 32)

送信クレジットの欠如が TX リード要求から PCI Express MegaCore ファンクションまで tx_pcie ブロックを防止するとき、リード・バイパス FIFO バッファは TX リード要求用にバッファ・スペースを提供します。PCI Express MegaCore ファンクションが TX リード要求を受け取る準備が整っていない場合、リード・バイパス FIFO バッファはリード要求を一時的に格納できて、その間も tx_pcie ブロックが他の要求を送信することができます。



PCI Express トランザクション層がパケットのフロー・コントロールを処理するので、これは要件ではなく最適化です。これはリード要求が受け入れられない場合、アプリケーション・ロジックが TLP の他のタイプを送信することにより、システムを最適化するためのだけです。

DMA エンジン

DMA エンジンの目的は、エンドポイントがルート・コンプレックスにトランザクションをドライブするアクティブ・コンポーネントであることを許容することです。このデザインでは、ルート・コンプレックスが BAR2 をターゲットとする PCI Express リンクの上にトランザクションを送信することによって DMA レジスタに書き込みます。レジスタは RX アプリケーション・ロジックに配置されます。DMA がトランザクションをドライブするとき、それは DDR2 SDRAM メモリからおよび DDR2 SDRAM メモリへの座標トランザクションです。

DMA エンジンにおける DMA レジスタを設定することにより、DMA トランザクションが開始されます。DMA レジスタは rx_pcie モジュールに配置されます。これらのレジスタは PCI Express MegaCore ファンクションでベース・アドレス・レジスタ BAR2 へのメモリ・マップドです。ターゲット・トランザクションがメモリ・マップド・アドレスを通してレジスタにアクセスできます。DMA レジスタはそれらのコンテンツを供給するのにルート・コンプレックスを必要として、以下の要素で構成されます。

- PCIe 下位アドレス・レジスタ (0x0)
- PCIe 上位アドレス・レジスタ (0x4)
- バイト・カウンタ・レジスタ (0x8)
- DMA コントロールおよびステータス・レジスタ (0xC)
- DMA ローカル・アドレス・レジスタ (0x10)

機能の説明

この項では、PCI Express-to-DDR2 SDRAM リファレンス・デザインにあるさまざまなモジュールの動作について説明して、以下の項目に関する情報を提供します。

- ルート・コンプレックスから開始されるトランザクション
- エンドポイントから開始されるトランザクション
- メモリ・マッピング
- DMA レジスタ・アクセス

ルート・コンプレックスで開始されるトランザクション

PCI Express MegaCore ファンクションがエラー無しでパケットを正常に受け取ると、パケットを PCIe RX コントロール・ロジック、`rx_pcie` に送信します。`rx_pcie` ロジックは PCIe パケットをデコードして、任意の関連するデータを受け取ります。そして、要求情報を RX コマンド FIFO バッファに転送して、データを RX データ FIFO バッファに転送します。

ページ7の図6に示すとおり、DDR2 サイド上の RX コントロール・ロジック、`rx_ddr` は RX FIFO バッファから要求およびデータを受信し、要求を複数の DDR2 要求に分割して、DDR2 SDRAM コントローラに要求を送信します。

この項では、以下の内容について説明します。

- ルート・コンプレックスが開始したメモリのリード
- ルート・コンプレックスが開始したメモリのライト

ルート・コンプレックスが開始したメモリのリード

メモリ・リード要求が RX アプリケーション・ロジックに達するとき、次の動作のシーケンスを実行します。

1. PCI Express MegaCore ファンクションは `rx_st_sop` をアサートして、ヘッダー H0 および H1 が `rx_st_data` で使用可能です。RX アプリケーション・ロジックはフォーマット (Fmt) およびタイプ (Type) フィールドを調べます。トランザクション・タイプはメモリ・リードであると決定できます。
2. `rx_sop` 信号がデアサートされて、`rx_st_data` で使用可能な H2 および H3 にはアドレス情報が含まれます。
3. `rx_pcie` モジュールは必要な情報を抽出して、要求をリフォーマットして、要求をコマンド FIFO に配置します。
4. DDR2 クロック・ドメインで、`rx_ddr` ロジックはコマンド FIFO からリード要求を受け入れて、それを1つ以上の DDR2 リード要求に変換します。`rx_ddr` ロジックは DDR2 SDRAM コントローラにリード要求を順番に送信します。
5. DDR2 SDRAM コントローラはリード要求をリフォーマットして、それらを DDR2 SDRAM メモリに送信します。
6. しばらくすると、DDR2 SDRAM メモリがリード要求データを返送します。
7. TX データ FIFO はデータを累算します。`tx_ddr_resp` ロジックはこの処理をモニタして、リード完了パケットを PCI Express MegaCore ファンクションに返送する時間を決定します。
8. 十分なデータが TX データ FIFO で累算するとき、`tx_ddr_resp` は PCIe リード完了パケットを生成して、TX コマンド FIFO に配置します。
9. TX データ FIFO を通じて、パケットはクロック・ドメインを越えて、`tx_pcie` ロジックは PCIe TX パケットとして読み出されます。
10. PCI Express MegaCore ファンクションはリード完了パケットを受け取る準備ができているとき、`tx_pcie` は生成された TLP を送信して、`tx_st_sop` および `tx_st_valid` 信号をアサートします。
11. 最後の DW のヘッダーまたはデータが送信されたとき、`tx_pcie` は `tx_st_eop` 信号をアサートします。

ルート・コンプレクスが開始したメモリのライト

ターゲットのメモリ・ライト要求が RX アプリケーション・ロジックに達するとき、次の動作のシーケンスを実行します。

1. PCI Express MegaCore ファンクションは `rx_st_sop` をアサートして、ヘッダー H0 および H1 が `rx_st_data` で使用可能です。RX アプリケーション・ロジックはフォーマット (Fmt) およびタイプ (Type) フィールドを調べます。トランザクション・タイプはメモリ・リードであると決定できます。
2. `rx_sop` 信号がデアサートされて、`rx_st_data` で使用可能な H2 および H3 にはアドレス情報が含まれます。
3. アプリケーション・ロジックは必要な情報を抽出して、要求をリフォーマットして、要求をコマンド FIFO に配置します。
4. DDR2 クロック・ドメインで、`rx_ddr` ロジックはコマンド FIFO からライト要求を受け入れて、それを 1 つ以上の DDR2 ライト要求に変換します。`rx_ddr` ロジックは DDR2 SDRAM コントローラにライト要求を順番に送信します。
5. DDR2 SDRAM コントローラはライト・データを受け取る準備ができたとき、`local_wdata_req` 信号がアサートされます。RX アプリケーション・ロジックは、`local_wdata` バスにライト・データを提供することによって応答します。
6. DDR2 SDRAM コントローラはライト要求をリフォーマットして、それらを DDR2 SDRAM メモリに送信します。

エンドポイントから開始されるトランザクション

PCI Express MegaCore ファンクションは DMA リードまたはライトの指定を意味し、BAR2 をターゲットとするパケットを正常に受け取ると、パケットを PCIe RX コントローラ・ロジック、`rx_pcie` に送信します。`rx_pcie` ロジックは PCIe パケットをデコードして、要求を DMA レジスタに書き込みます。

RX コントロール・ロジックは `tx_dma_rd` ブロックによって読み出される `dma_start` 信号をアサートします。

この項では、以下の内容について説明します。

- エンドポイントが開始した DMA のリード
- エンドポイントが開始した DMA のライト

エンドポイントが開始した DMA のリード

DMA リード要求を開始するには、最初にルート・ポートがすべての DMA レジスタを順番にプログラムします。DMA コントロール・レジスタが書き込まれ、デコードされた後に、`dma_start` 信号が DMA リードを開始するために 1 クロック・サイクルの間アサートされます。

PCIe DMA リード要求を処理しているとき、次の動作のシーケンスを実行します。

1. `dma_start` 信号がアサートされると、TX アプリケーション・ロジックにある `tx_dma_rd` 信号は DMA リード・トランザクションを開始します。DMA レジスタから得られた情報に基づく DMA リード要求を生成させて、それらを TX コマンド FIFO バッファにロードします。
2. TX PCIe パケットは TX コマンド FIFO バッファでのクロック・ドメイン境界を越えると、`tx_pcie` ブロックがそれらを読み出します。

3. PCI Express MegaCore ファンクションはリード完了パケットを受け取る準備ができているとき、`tx_pcie` が生成された TLP を送信して、`tx_st_sop` および `tx_st_valid` 信号をアサートします。PCI Express MegaCore ファンクションはメモリ・リード要求を受け付ける準備ができなく、または TX リード要求のための使用可能な送信クレジットがない場合、TX リード要求をリード・バイパス FIFO に迂回します。その代わりに、`tx_pcie` ブロックは以降の非リード要求を送信することができます。
4. 最後の DW のヘッダーまたはデータが送信されたとき、`tx_pcie` は `tx_st_eop` 信号をアサートします。
5. PCI Express MegaCore ファンクションが DMA リード要求を受け付けると、それをリフォーマットして、PCIe リンクに送信します。
6. しばらくすると、PCIe リンクはリード完了要求で返送します。
7. PCI Express MegaCore ファンクションは `rx_st_sop` をアサートして、ヘッダー H0 および H1 が `rx_st_data` で使用可能です。RX アプリケーション・ロジックはフォーマット (Fmt) およびタイプ (Type) フィールドを調べます。トランザクション・タイプはリード完了要求であると決定できます。
8. `rx_sop` 信号がディアサートされて、`rx_st_data` で使用可能な H2 にはアドレス情報が含まれます。
9. アプリケーション・ロジックは必要な情報を抽出して、要求をリフォーマットして、要求を RX コマンド FIFO に配置します。
10. データは RX データ FIFO に入って、累積します。
11. クロック・ドメイン境界を越えると、RX リード完了要求で `rx_ddr` ブロックを読み出します。
12. 十分なデータが存在しているとき、`rx_ddr` ブロックは DDR リード・コマンドを生成して、DDR2 SDRAM コントローラにデータを送信します。
13. DDR2 SDRAM コントローラはデータを保存するために、ライト・コマンドと共に DDR2 SDRAM メモリにデータを転送します。

エンドポイントが開始した DMA のライト

DMA ライト要求を開始するには、最初にルート・ポートがすべての DMA レジスタを順番にプログラムします。DMA コントロール・レジスタが書き込まれ、デコードされた後に、`dma_start` 信号が DMA ライトを開始するために 1 クロック・サイクルの間アサートされます。

PCIe DMA ライト要求を処理しているとき、次の動作のシーケンスを実行します。

1. RX アプリケーション・ロジック・ブロック (`rx_pcie`) は `dma_start` 信号を検出したとき、DMA レジスタから得られた情報に基づく RX PCIe パケットを生成させます。`rx_pcie` ブロックは RX PCIe パケットを RX コマンド FIFO に書き込みます。
2. DDR2 クロック・ドメインで、`rx_ddr` ブロックはコマンド FIFO からリード要求を受け入れて、それを 1 つ以上の DDR2 リード要求に変換します。`rx_ddr` ブロックは DDR2 SDRAM コントローラにリード要求を順番に送信します。
3. DDR2 SDRAM コントローラはリード要求をリフォーマットして、それらを DDR2 SDRAM メモリに送信します。
4. しばらくすると、DDR2 SDRAM メモリがリード要求データを返送します。

5. TX データ FIFO でデータを累算します。同時に、TX アプリケーション・ロジックでの `tx_dds_resp` ブロックは、この処理をモニタして、リード・データを PCI Express MegaCore ファンクションに送信する時間を決定します。
6. 十分なデータが TX データ FIFO で累算するとき、TX アプリケーション・ロジックでの `tx_dds_resp` ブロックは PCIe ライト・パケットを生成して、TX コマンド FIFO に配置します。
7. 十分なデータが TX データ FIFO で累算するとき、`tx_dds_resp` は PCIe メモリ・ライト・パケットを生成して、TX コマンド FIFO に配置します。
8. TX データ FIFO を通じて、パケットはクロック・ドメインを越えて、`tx_pcie` ロジックは PCIe TX パケットとして読み出されます。
9. PCI Express MegaCore ファンクションはリード完了パケットを受け取る準備ができているとき、`tx_pcie` は生成された TLP を送信して、`tx_st_sop` および `tx_st_valid` 信号をアサートします。
10. 最後の DW のヘッダーまたはデータが送信されたとき、`tx_pcie` は `tx_st_eop` 信号をアサートします。

メモリ・マッピング

このファレンス・デザインでは、PCI Express MegaCore ファンクションは 2 つのベース・アドレス・レジスタ BAR0 と BAR1 を通じてアクセスされた 16 M バイトのシステム・アドレス空間 (表 1 を参照) および BAR2 を通じて DMA レジスタの 4 K バイトのシステム・アドレス空間を予約します。

表 1. メモリ・アドレス空間

メモリ領域	ブロック・サイズ	メモリ・タイプ	説明
BAR0 および BAR1	16 M バイト	64 ビット、プリフェッチ可能	24 ビットのアドレス・バスがサポート可能な 16 M バイト DDR2 メモリ範囲。(1)
BAR2	4 K バイト	32 ビット、プリフェッチ不可能	内部リファレンス・デザイン DMA コンフィギュレーション・レジスタ。

表 1 の注:

- (1) PCIe アドレスは 32 または 64 ビット幅があっても、アプリケーション・ロジックは DDR2 SDRAM コントローラに 24 ビットのみを表示します。

BAR0 および BAR1 は DDR2 メモリ・アドレス範囲を指定します。要求ディスクリプタで規定されるとおり、32 ビットまたは 64 ビット・アドレスに基づく BAR0 および / または BAR1 に一致する BAR がある場合、PCI Express ファンクションはターゲット・トランザクションを要求します。

BAR2 は DMA レジスタをマップします (表 2 を参照)。

表 2. 内部レジスタ・メモリ・マップド・アドレス

予約された範囲	リード/ライト	ニーモニック	レジスタ名
0x00000 - 0x00003	ライト	HPAR [31:0]	DMA PCIe 上位アドレス・レジスタ
0x00004 - 0x00007	ライト	LPAR [31:0]	DMA PCIe 下位アドレス・レジスタ
0x00008 - 0x0000B	ライト	BCR [31:0]	DMA バイト・カウンタ・レジスタ
0x0000C - 0x0000F	リード/ライト	CSR [31:0]	DMA コントロール/ステータス・レジスタ
0x00010 - 0x00013	ライト	LAR [31:0]	DMA ローカル・アドレス・レジスタ

DMA レジスタ

この項では、DMA エンジンの DMA レジスタについて説明します。このリファレンス・デザインには、次のレジスタを実装します。

- PCIe アドレス・レジスタ (PAR)
- DMA バイト・カウンタ・レジスタ (BCR)
- コントロール・ステータス・レジスタ (CSR)
- ローカル・アドレス・レジスタ (LAR)

これらのレジスタは PCIe ファンクションの BAR2 にメモリ・マップドされ、FPGA とインタフェースする FPGA の任意のロジック・モジュールまたはコンポーネントを可能にして、転送を開始する前に、それらの内容をコンフィギュレーションします。

DMA レジスタをアクティブにするために、DMA は以下のシーケンスで DMA レジスタに書き込むのを必要とします。

1. 現在の DMA トランザクションのための PCIe 開始アドレスで PAR に書き込みます。
2. 転送するバイト数を BCR に書き込みます。
3. トランザクションの DDR2 SDRAM メモリ開始アドレスを LAR に書き込みます。
4. 適切な値を CSR に書き込みます。

CSR のビット 7 に logical 1 を書き込むと、DMA ステート・マシンが開始します。次のクロック・サイクルは、DMA 転送が進行しているのを示すために、CSR レジスタの dma_busy ビットを設定します。DMA エンジンは、対応するブロックに要求を送信して、DMA 転送を開始します。レシーバ・ブロックは、DDR2 SDRAM メモリとシステム・メモリでデータを転送します。DMA 転送が完了し、新しい DMA トランザクションが始まるとき、DMA エンジンは dma_busy 信号をデアサートします。

PCIe アドレス・レジスタ (PAR)

PCIe アドレスは 64 ビット長ですが、PAR は 32 ビット・レジスタです。したがって、PCIe アドレス・レジスタは 32 ビット・レジスタで構成されています。PCIe 上位アドレス・レジスタ (HPAR) および PCIe 下位アドレス・レジスタ (LPAR) です。HPAR と LPAR を結合すると、PCIe アドレスが形成されます。32 ビット PCIe 要求アドレッシングに対して、LPAR ビットのみが有効です。64 ビット PCIe 要求アドレッシングに対して、結合した 64 ビット・アドレスのみが有効です。

PAR には現在の DMA 転送用に PCIe バス・アドレスが含まれて、再プログラムされるまで変更されません。DMA エンジンによって開始された PCIe バス DMA 転送は、QWORD 境界で始まる必要があります。表 3 に、PAR フォーマットを示します。

表 3. DMA PCI アドレス・レジスタ・フォーマット

データ・ビット	名称	リード/ライト	定義
31..0	PAR	ライト	32 ビット・アドレス

DMA バイト・カウンタ・レジスタ (BCR)

DMA バイト・カウンタ・レジスタは 13 ビット・レジスタです。13 ビット・レジスタは BCR を実装します。BCR は現在の DMA メモリ転送のためのバイト・カウンタを保持します。DMA エンジンによって開始された PCIe バス DMA 転送は、QWORD 転送を必要とします。表 4 に、BCR フォーマットを示します。

表 4. DMA バイト・カウンタ・レジスタ・フォーマット

データ・ビット	名称	リード/ライト	定義
21..0	BCR	ライト	13 ビット・カウンタ
31..13	—	—	予約済み

コントロール・ステータス・レジスタ (CSR)

32 ビット CSR は DMA エンジンをコンフィギュレーションします。CSR は DMA 動作を指示して、現在の DMA 転送のステータスを提供します。表 5 は、CSR フォーマットについて説明しています。

表 5. DMA バイト・カウンタ・レジスタ・フォーマット

データ・ビット	ニックネーム	リード/ライト	定義
5..0	—	—	予約済み
6	リード/ライト	ライト	DMA リード・ライト。DMA ライトに対して、High にアサートされます。DMA リードに対して、デアサートされます。
7	DMA 開始	ライト	DMA 開始。High にアサートされることはすべての DMA レジスタがプログラムされ、DMA トランザクションを開始する準備ができていることを示します。

表 5. DMA バイト・カウンタ・レジスタ・フォーマット

データ・ビット	ニーマニック	リード/ライト	定義
30..8	—	—	予約済み
31	DMA ビジー	リード	DMA ビジー。High にアサートされることは現在の DMA がまだ進行していて、DMA レジスタが新しいプログラミングの準備ができていないことを示します。

ローカル・アドレス・レジスタ (LAR)

32 ビット LAR は SDRAM に転送するデータおよび SDRAM から転送されるデータの SDRAM アドレスを保持します。23 ビット・カウンタで実装されて、LAR が書き込み専用レジスタです。表 6 に LAR のフォーマットを示します。


表 6. DMA ローカル・アドレス・レジスタ・フォーマット

データ・ビット	名称	リード/ライト	定義
22..0	LAR	ライト	23 ビット・アドレス
31..23	—	—	予約済み

インタフェース信号の説明

この項では、リファレンス・デザインのアプリケーション・ロジックの信号の 2 つのカテゴリで説明します。

- PCI Express MegaCore ファンクションとインタフェースする信号
- DDR2 SDRAM コントローラとインタフェースする信号

 PCI Express MegaCore ファンクションのローカル信号については、[「PCI Express Compiler User Guide」](#) を参照してください。DDR2 SDRAM 高性能コントローラのローカル信号については詳しくは、[「DDR & DDR2 SDRAM High-Performance Controller v9.0 User Guide」](#) を参照してください。

PCI Express MegaCore とインタフェースする信号

表 7 に、*applogic.v* からの PCI Express MegaCore ファンクションとインタフェースするデータ信号を説明します。

表 7. PCI Express MegaCore とインタフェースする信号 (その 1)

信号	I/O	説明
Rstn_i	入力	アプリケーション・ロジックの信号をリセットします。信号はアクティブ Low です。
PCIEClk_i	入力	PCI Express MegaCore の coreclk_out から供給される 250 MHz です。
Rx_St_Data_i [63:0]	入力	PCI Express MegaCore からストリーミング・データ信号を受信します。信号は 64 ビット幅で、この信号にすべての TLP が受信されます。 この信号については詳しくは、 「PCI Express Compiler User Guide」 を参照してください。

表7. PCI Express MegaCore とインタフェースする信号 (その2)

信号	I/O	説明
Rx_St_Valid_i	入力	PCI Express MegaCore からストリーミング・データ有効信号を受信します。ロジックが Rx_St_Data_i 信号に有効信号をクロックすると、この信号がアサートされます。 この信号について詳しくは、 「PCI Express Compiler User Guide」 を参照してください。
Rx_St_Sop_i	入力	PCI Express MegaCore からストリーミング SOP 信号を受信します。この信号は Rx_St_Data_i における TLP 伝送の開始の1クロック・サイクルの間アサートされます。 この信号について詳しくは、 「PCI Express Compiler User Guide」 を参照してください。
Rx_St_Eop_i	入力	PCI Express MegaCore からストリーミング EOP 信号を受信します。この信号は Rx_St_Data_i における TLP 伝送の終点の1クロック・サイクルの間アサートされます。 この信号について詳しくは、 「PCI Express Compiler User Guide」 を参照してください。
Rx_St_Bardec_i [7:0]	入力	PCI Express MegaCore からストリーミング BAR デコード信号を受信します。この信号は、BAR が Rx_St_Data_i における TLP 伝送のためにターゲットされていることを表します。このデザインはターゲットとする BAR0 または BAR2 を持つことができます。 この信号について詳しくは、 「PCI Express Compiler User Guide」 を参照してください。
Rx_St_Be_i [7:0]	入力	PCI Express MegaCore からストリーミング・バイト・イネーブル信号を受信します。各ビットは Rx_St_Data_i で対応するバイトを参照します。バイトが有効である場合、対応するビットが High になります。 この信号について詳しくは、 「PCI Express Compiler User Guide」 を参照してください。
Rx_St_Ready_o	出力	PCI Express MegaCore からストリーミング・レディ信号を受信します。アプリケーションはデータを受け取る準備ができたとき、この信号がアサートされます。 この信号について詳しくは、 「PCI Express Compiler User Guide」 を参照してください。
Rx_St_Mask_o	出力	PCI Express MegaCore からストリーミング・マスク信号を受信します。アプリケーションは非ポスト要求を受け取る準備が整っていない場合、この信号がアサートされます。 この信号について詳しくは、 「PCI Express Compiler User Guide」 を参照してください。
Tx_St_Data_o [63:0]	出力	PCI Express MegaCore からストリーミング・データ信号を送信します。信号は 64 ビット幅で、この信号にすべての TLP が送信されます。 この信号について詳しくは、 「PCI Express Compiler User Guide」 を参照してください。

表 7. PCI Express MegaCore とインタフェースする信号 (その 3)

信号	I/O	説明
Tx_St_Valid_o	出力	PCI Express MegaCore ヘストリーミング・データ有効信号を送信します。TLP データが Tx_St_Data_i 信号に有効であるとき、この信号がアサートされます。 この信号について詳しくは、 「PCI Express Compiler User Guide」 を参照してください。
Tx_St_Sop_o	出力	PCI Express MegaCore ヘストリーミング SOP 信号を送信します。この信号は Tx_St_Data_i における TLP 伝送の開始の 1 クロック・サイクルの間アサートされます。 この信号について詳しくは、 「PCI Express Compiler User Guide」 を参照してください。
Tx_St_Eop_o	出力	PCI Express MegaCore ヘストリーミング EOP 信号を送信します。この信号は Tx_St_Data_i における TLP 伝送の終点の 1 クロック・サイクルの間アサートされます。 この信号について詳しくは、 「PCI Express Compiler User Guide」 を参照してください。
Tx_St_Ready_i	入力	PCI Express MegaCore からストリーミング・レディ信号を送信します。PCI Express MegaCore はデータを受け取る準備ができたとき、この信号がアサートされます。 この信号について詳しくは、 「PCI Express Compiler User Guide」 を参照してください。
TxCred_i [35:0]	入力	PCI Express MegaCore からストリーミング・クレジット信号を送信します。この信号は、PCI Express MegaCore がどのトランザクションのタイプを受け入れることができるかを説明します。 この信号について詳しくは、 「PCI Express Compiler User Guide」 を参照してください。
DevCsr_i [31:0]	入力	PCI Express MegaCore をコンフィギュレーションするために使用されたコンフィギュレーション・モジュールからです。最大リード・サイズと述べます。
BusDev_i [12:0]	入力	PCI Express MegaCore をコンフィギュレーションするために使用されたコンフィギュレーション・モジュールからです。コンプリータとリクエスタ ID を決定するのに使用されます。

DDR2 SDRAM 高性能コントローラとインタフェースする信号

表 8 に、DDR2 SDRAM 高性能コントローラとインタフェースするデータ信号について説明します。すべての信号は rx_top または tx_top とインタフェースします。簡単にするために、信号名は DDR2 SDRAM 高性能コントローラの命名規則に対応します。

表 8. DDR2 SDRAM 高性能コントローラとインタフェースする信号 (その1)

信号	I/O	説明
local_ready	入力	rx_top (DdrReady_i) への入力。コントローラが要求を受け付ける準備ができており、この信号がアサートされます。 詳細については、「 DDR & DDR2 SDRAM High-Performance Controller v9.0 User Guide 」を参照してください。
local_address [24:0]	出力	rx_top (DdrAddress_o) からの出力。この信号は書き込まれるまたは読み出されるメモリのアドレスを保持します。 詳細については、「 DDR & DDR2 SDRAM High-Performance Controller v9.0 User Guide 」を参照してください。
local_write_req	出力	rx_top (DdrWrite_o) からの出力。ライトが local_address におけるアドレスに要求される時、この信号がアサートされます。 詳細については、「 DDR & DDR2 SDRAM High-Performance Controller v9.0 User Guide 」を参照してください。
local_read_req	出力	rx_top (DdrRead_o) からの出力。リードが local_address におけるアドレスから要求される時、この信号がアサートされます。 詳細については、「 DDR & DDR2 SDRAM High-Performance Controller v9.0 User Guide 」を参照してください。
local_wdata_req	出力	rx_top (DdrWrDataReq_i) への入力。コントローラが次のクロック・サイクルでデータを受け付ける準備ができた時、この信号がコントローラでアサートされます。 詳細については、「 DDR & DDR2 SDRAM High-Performance Controller v9.0 User Guide 」を参照してください。
local_wdata [127:0]	出力	rx_top (DdrWriteData_o) からの出力。この信号にはメモリに書き込まれるデータが含まれています。データは 128 ビット幅ですが、64 ビットのみが使用されます。これは、選択した特定のメモリ・コンポーネントは 128 ビット幅であるためです。 詳細については、「 DDR & DDR2 SDRAM High-Performance Controller v9.0 User Guide 」を参照してください。
local_be	出力	rx_top (DdrByteEnable_o) からの出力。この信号は書き込みの時に local_wdata の個々のバイトをマスクするバイト・イネーブル信号です。 詳細については、「 DDR & DDR2 SDRAM High-Performance Controller v9.0 User Guide 」を参照してください。
local_size	出力	rx_top (DdrBurstCount_o) からの出力。この信号は要求されたアクセスのバースト・サイズであり、2 進数としてエンコードされます。 詳細については、「 DDR & DDR2 SDRAM High-Performance Controller v9.0 User Guide 」を参照してください。

表 8. DDR2 SDRAM 高性能コントローラとインタフェースする信号 (その2)

信号	I/O	説明
local_rdata[63:0]	入力	tx_top (TxReadData_i)への入力。この信号にはリード要求が local_read_req から開始された後に、メモリから返送されるデータが含まれます。 詳細については、「 DDR & DDR2 SDRAM High-Performance Controller v9.0 User Guide 」を参照してください。
local_rdata_valid	入力	tx_top (TxReadDataValid_i)への入力。local_rdata におけるデータが有効であると、この信号がアサートされます。 詳細については、「 DDR & DDR2 SDRAM High-Performance Controller v9.0 User Guide 」を参照してください。

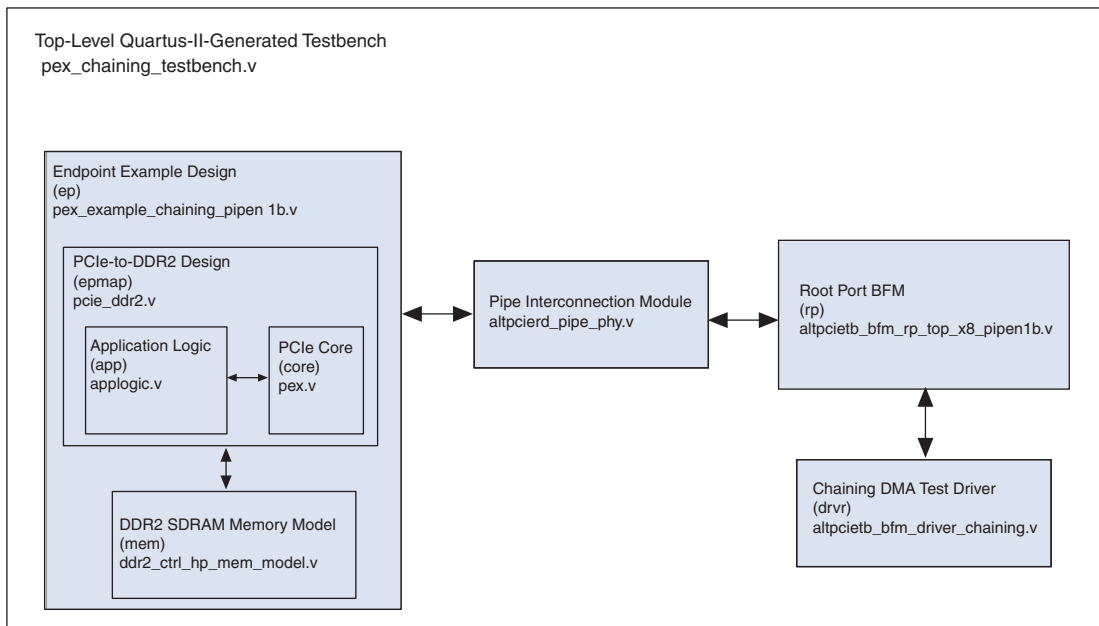
リファレンス・デザインのシミュレーション

リファレンス・デザインのシミュレーション方法については、以下の項で説明しています。MegaWizard Plug-In Manager を使用して PCI Express コアを生成するとき、Quartus II ソフトウェアは、テストベンチ・モデルを生成します。このデザインはそのシミュレーション・モデルを内蔵しています。テストベンチは ModelSim-Altera Edition で使用するものです。

シミュレーション・デザイン階層

Quartus II ソフトウェアから生成されたファイルでは、**pex_example_chaining_pipen1b.v** で PCI Express コア (**pex.v**) がインスタンス化されます。このデザインは PCI Express コアのインスタンスをトップレベル・ファイル (**pcie_ddr2.v**) に置き換えます。簡単にするために、DDR2 SDRAM 高性能コントローラ用に生成されるメモリ・モデルはエンドポイント・デザイン例 (**pex_example_chaining_pipen1b.v**) 内で参照されます ([図 20](#) を参照)。

図 20. トップ・レベル Quartus II の生成されたテストベンチ



以下を除いて、PCI Express MegaCore が生成される時、テストベンチにあるすべてのファイルが Quartus II ソフトウェアによって生成されます。

- **pcie_ddr2.v** — トップ・レベル・デザイン・ファイル
- **applogic.v** および参照されるすべてのファイル — DDR2 SDRAM 高性能メモリ・コントローラのインスタンス化を含むアプリケーション・ロジック
- **ddr2_ctrl_hp_mem_model.v** — DDR2 SDRAM 高性能メモリ・コントローラが生成される時、Quartus II ソフトウェアによって生成される

PCI Express MegaCore が再生成される場合、ファイルは上書きされて、変更する必要があります。

- **pex_example_chaining_pipen1b.v** — トップ・レベル・デザイン・ファイル (**pcie_ddr2.v**) およびメモリ・モジュールをインスタンス化します。PCI Express MegaCore 上のポートが変更する場合、**pcie_ddr2.v** ファイルも変更する必要があります。
- **altpcierrb_bfm_driver_chaining.v** — チェーン DMA テスト・ドライバにはルート・コンプレックスによって送信されるコマンドが含まれます。

 チェーン DMA テスト・ドライバを使用して、PCI Express コアのシリアル・サイドへの入力を提供します。「[PCI Express Compiler User Guide](#)」の第 7 章で参照されるように、多くの指示が使用可能です。

チェーン DMA テスト・ドライバはルート・ポートに指定を提供し、指定は理論的に PCI Express コアの rx_in ポートを提供させるシリアル・フォーマットに変換されます。シミュレーションを簡素化するために、パイプ・インタコネクト・モジュール (PIPE) を通して指定を平行に送信します。

Quartus II ソフトウェアによって生成された PCI Express MegaCore シミュレーション・モデルが *PCI Express Base Specification, Rev 2.0* に準拠したため、使用されます。コアのコンフィギュレーションの複雑さおよび物理、データ・リンクとトランザクション層内の相互作用は、Quartus II ソフトウェアによって生成されたモデルで処理されるほうがよいです。

チェーン DMA テスト・ドライバのコマンド

ebfm_barwr および ebfm_barrd の手順を使用してください。表 9 および表 10 に、これらの手順を示します。

表 9. ebfm_barwr の手順

位置:	altpcieth_bfm_rdwr.v または altpcieth_bfm_rdwr.vhd	
構文:	ebfm_barwr (bar_table, bar_num, pcie_offset, lcladdr, byte_len, tclass)	
引数:	bar_table	BFM 共有メモリのエンドポイント bar_table 構造のアドレス。ドライバ・コードは BAR からのアプリケーション特定のオフセットのみを注意して、実際の割り当てアドレスを注意する必要はないので、bar_table 構造は各 BAR に割り当てアドレスを格納します。
	bar_num	PCI Express アドレスを決定するために pcie_offset で使用される BAR の番号。
	pcie_offset	BAR ベースからのアドレス・オフセット。
	lcladdr	書き込まれるデータの BFM 共有メモリ・アドレス。
	byte_len	書き込まれるデータの長さ (バイト数)。BAR 領域または BFM 共有メモリ内の残りの最小バイト数は 1 からになります。
	tclass	PCI Express トランザクションに使用するトラフィック・クラス。

表 10. ebfm_barrd の手順

位置:	altpcieth_bfm_rdwr.v または altpcieth_bfm_rdwr.vhd	
構文:	ebfm_barrd_wait (bar_table, bar_num, pcie_offset, lcladdr, byte_len, tclass)	
引数:	bar_table	BFM 共有メモリのエンドポイント bar_table 構造のアドレス。ドライバ・コードは BAR からのアプリケーション特定のオフセットのみを注意して、実際の割り当てアドレスを注意する必要はないので、bar_table 構造は各 BAR に割り当てアドレスを格納します。
	bar_num	PCI Express アドレスを決定するために pcie_offset セットで使用される BAR の番号。
	pcie_offset	BAR ベースからのアドレス・オフセット。
	lcladdr	書き込まれるデータの BFM 共有メモリ・アドレス。
	byte_len	書き込まれるデータの長さ (バイト数)。BAR 領域または BFM 共有メモリ内の残りの最小バイト数は 1 からになります。
	tclass	PCI Express トランザクションに使用するトラフィック・クラス。

これらの手順を使用する前に、ルート・ポート BFM にデータを一時的に書き込む必要があります。そして lcladdr 引数でメモリの書き込む場所を指定します。図 21 に、ルート・ポート・アドレス空間を示します。

図 21. ルート・ポート・アドレス空間



0x0000 0000 ~ 0x001F FF80 のアドレス空間は手順を使用して、送信することができるデータを格納可能です。アドレスの値は `lcladdr` 引数で指定されます。データの長は `byte_len` 引数で指定されます。

ルート・ポート・アドレス空間にデータを書き込むには、`shmem_fill` の手順を使用します (表 11 を参照)。`addr` 引数は以前に説明されたリードおよびライト手順における `lcladdr` 引数に対応します。

表 11. `shmem_fill` の手順

位置:	<code>altpcieth_bfm_shmem.v</code> または <code>altpcieth_bfm_shmem.vhd</code>	
構文:	<code>shmem_fill (addr, mode, leng, init)</code>	
引数:	<code>addr</code>	データを入れ込む用の BFM 共有メモリ開始アドレス。
	<code>mode</code>	データを入れ込む用のデータ・パターン。
	<code>leng</code>	書き込まれるデータの長さ (バイト数)。長さが増加するデータ・パターン幅の倍数ではない場合、最後のデータ・パターンを適合させるように切り捨てられます。
	<code>init</code>	VHDL で増加しているデータ・パターン・モードに使用される初期データ値です。この引数はタイプ <code>Std_logic_vector [63 down to 0]</code> です。Verilog HDL では、この引数は <code>reg [63:0]</code> です。両言語とも、必要な最下位ビットは 64 ビットより小さいデータ・パターンに使用されます。

shmem_fill 手順のモード引数は表 12 で定義された定数によって指定されます。

表 12. shmem_fill 手順のモード引数

定数	説明
SHMEM_FILL_ZEROS	すべての 0 のデータ・パターンを指定します。
SHMEM_FILL_BYTE_INC	増加する 8 ビット・バイトのデータ・パターンを指定します (0x00、0x01、0x02、など。)
SHMEM_FILL_WORD_INC	増加する 16 ビット・ワードのデータ・パターンを指定します (0x0000、0x0001、0x0002、など。)
SHMEM_FILL_DWORD_INC	増加する 32 ビット dword のデータ・パターンを指定します (0x00000000、0x00000001、0x00000002、など。)
SHMEM_FILL_QWORD_INC	増加する 64 ビット qword のデータ・パターンを指定します (0x0000000000、0x0000000001、0x0000000002、など。)
SHMEM_FILL_ONE	すべての 1 のデータ・パターンを指定します。

テストベンチ・スティミュラスの生成

主な手順では、`altpcieth_bfm_driver_chaining.v` の下部の近くで、`/pex_examples/chaining_dma/testbench` ディレクトリのファイルにおける以下のコマンドが使用されます。

最初に PCI Express コアをコンフィギュレーションするために、例 1 にある `ebfm_cfg_rp_ep` 手順を使用します。

例 1.

```
ebfm_cfg_rp_ep(  
bar_table,           // BAR Size/Address info for Endpoint  
1,                   // Bus Number for Endpoint Under Test  
1,                   // Device Number for Endpoint Under Test  
512,                 // Maximum Read Request Size for Root Port  
1,                   // Display EP Config Space after setup  
addr_map_4GB_limit // Limit the BAR assignments to 4GB address map  
);
```

FPGA に書き込まれるデータを指定するには、`shmem_fill` コマンドはバーチャル・ルート・コンプレックスの 0x0 アドレス空間にデータを書き込みます (例 2 を参照)。

例 2.

```
// Shared Memory Fill: Write data to Root Complex memory to hold  
//      Memory      Data Pattern      Length of      Starting Data  
//      Address      Type      Bytes to Fill      Pattern Value  
shmem_fill(      0, SHMEM_FILL_DWORD_INC,      64,      4'hAAAAAA00BBBBB00);
```

DDR2 SDRAM メモリ (BAR0 と BAR1) および DMA レジスタ (BAR2) の両方に対して、`pcie_offset` を 0 であると考えます。テストベンチは、`ebfm` 手順で指定されたアドレスに追加されるオフセットを参照することによって、エンドポイントのためのアドレスを生成します (例 3 を参照)。

例 3.

```
pcie_offset = 32'h0;
```

現在 `ebfm_barwr` ライト手順がバーチャル・ルート・コンプレックスのアドレス空間で保持されるデータを含むメモリ・ライト TLP を DDR2 SDRAM に送信します (例 4 を参照)。

例 4.

```
// ebfm_barwr(bar_table, bar_num, pcie_offset, lcladdr, byte_len, tclass)
ebfm_barwr(bar_table, 0, pcie_offset, 0, 64, 0);
```

現在 `ebfm_barrd` リード手順が DDR2 SDRAM メモリに書き込まれたデータを要求するメモリ・リード TLP を送信します (例 5 を参照)。

例 5.

```
// ebfm_barrd_wait(bar_table, bar_num, pcie_offset, lcladdr, byte_len, tclass)
ebfm_barrd_wait(bar_table, 0, pcie_offset, 32'hF0, 64, 0);
```

シミュレーションの実行

シミュレーションを実行するのに約 382,000 ns をかかります。これは PCI Express のハード IP と DDR2 SDRAM 高性能コントローラ IP の両方にはかなり時間がかかる初期化を必要とするシミュレーション・モデルがあるからです。シミュレーションを実行するには、以下のステップに従います。


Archived Project および Open Project の復元

`.qar` には Quartus II インパイルおよび ModelSim シミュレーションのすべてのファイルが含まれます。

1. **Project** を選択し、次に **Restore Archived Project** を選択します。
2. `pcie_ddr2.qar` に移動します。
3. 確認するには、**OK** をクリックします。
4. **Yes** をクリックして、新しいディレクトリを作成します。
5. **Open restored project** がチェックされていることを確認します。
6. 確認するには、**OK** をクリックします。

プロジェクトのコンパイル

このステップはトップレベル・デザイン・ファイル `pcie_ddr2.v` をコンパイルします。

1. プロジェクトをコンパイルするには、 をクリックします。
2. コンパイルの結果を確認します。

ModelSim でのデザインのシミュレーション

ModelSim ソフトウェアを使用してデザインをシミュレートするには、これらのステップを実行します。ModelSim でシミュレートするために、Quartus II ソフトウェアでのデザインをコンパイルする必要はありません。

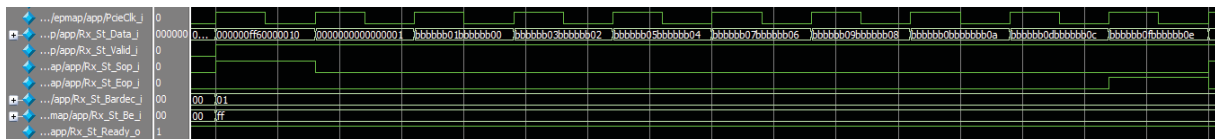
1. ModelSim-Altera を開きます。
2. **File** を選択し、次に **Change Directory...** を選択します。
3. `<top level directory>\pex_examples\changing_dma\testbench` に移動します。

4. **OK** をクリックします。
5. Modelsim> コマンド・プロンプトで、do runt.b.do を入力します。

シミュレーション結果の表示

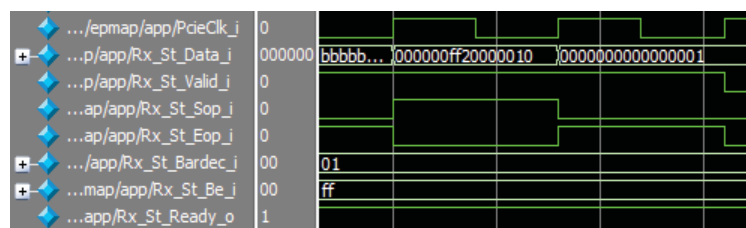
ルート・コンプレックスは MemWr TLP をエンドポイントへの書き込むデータの次に DDR2 SDRAM メモリに送信します (図 22 を参照)。

図 22. MemWr TLP



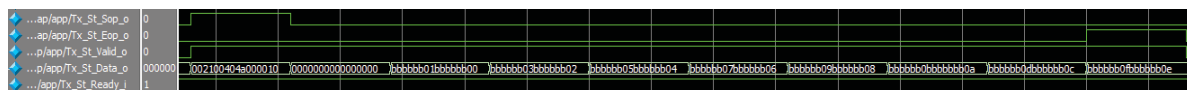
そして、ルート・コンプレックスは MemRd TLP をエンドポイントに送信して、データを要求します (図 23 を参照)。

図 23. MemRd TLP



次に、エンドポイントはデータ付き CpID TLP をルート・コンプレックスに返送します (図 24 を参照)。

図 24. CpID TLP



参考資料

このアプリケーション・ノートでは、以下のドキュメントを参照しています。

- 「PCI Express Compiler User Guide」
- 「DDR & DDR2 SDRAM High-Performance Controller v9.0 User Guide」

改訂履歴

表 13 に、このアプリケーション・ノートの改訂履歴を示します。

表 13. 改訂履歴

日付およびドキュメント・バージョン	変更内容	概要
2009年4月、v1.0	初版。	—



101 Innovation Drive
San Jose, CA 95134
www.altera.com
Technical Support
www.altera.com/support

Copyright © 2009 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



I.S. EN ISO 9001