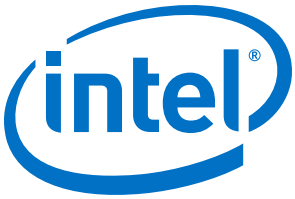


インテル® アクセラレーション・スタック (インテル® Xeon® CPU&FPGA 対応) コア・キャッシュ・インターフェイス (CCI-P) リファレンス・マニュアル

更新情報
フィードバック

MNL-1092 | 2019.11.04
最新版をウェブからダウンロード: PDF | HTML

目次

1. アクセラレーション・スタック (インテル® Xeon® CPU & FPGA 対応) コア・キャッシュ・インターフェイス (CCI-P) リファレンス・マニュアル

1.1. 本リファレンス・マニュアルについて

1.1.1. 本リファレンス・マニュアルのご利用にあたり

このリファレンス・マニュアルは、システムエンジニア、プラットフォーム設計者、ハードウェアおよびソフトウェア・デベロッパーを対象にしています。

ハードウェア AFU は、CCI-P 仕様に準拠するようにデザインする必要があります。

1.1.2. 規則

表 1. 表記規則

規則	説明
#	コマンドに先行し、そのコマンドがルートとして入力されることを示します。
$	ユーザーとして入力されるコマンドを示します。
このフォント (This font) で書かれている文字	ファイル名、コマンド、およびキーワードはこのフォントで表示されます。長いコマンドラインもこのフォントで表示され、次の行に折り返されている場合がありますが、改行はコマンドの一部ではありません。Enter キーを押さないでください。
<variable_name>	プレースホルダー・テキストを示し、山括弧の間に表示されます。このテキストは適切な値に置き換える必要があります。山括弧は入力しないでください。

1.1.3. 関連資料

表 2. 関連資料

資料	説明
Intel® Software Developers Manual	この資料には、*Intel 64 and IA-32 Architecture Software Development Manual* の 3 つの Volume すべてが含まれます (Basic Architecture: 注文番号 253665、Instruction Set Reference A-Z: 注文番号 325383、System Programming Guide: 注文番号 325384)。デザインニーズを評価する際は、この 3 つの Volume をすべて参照ください。
Intel Virtualization Technology for Directed I/O Architecture Specification	このドキュメントは、ダイレクト I/O 向けインテル・バーチャライゼーション・テクノロジー (ダイレクト I/O 向けインテル VT) について説明しています。このテクノロジーは、インテルのプラットフォーム仕様に準拠するインテル・プロセッサーおよびコア・ロジック・チップセットを使用するプラットフォームに適用されるため、I/O バーチャライゼーションをサポートするコンポーネントに関して具体的に説明します。

1.1.4. アクセラレーション・スタック (インテル Xeon® CPU & FPGA 対応) コア・キャッシュ・インターフェイス (CCI-P) リファレンス・マニュアルの頭字語一覧

表 3. 頭字語

「A/B」の欄は、その用語が適用されるパッケージを示します。

- **A:** FPGA を統合したインテル® Xeon® スケーラブル・プラットフォームを指します。このリファレンス・マニュアルでは、FPGA 統合プラットフォームと表されます。
- **B:** インテル®FPGA プログラマブル・アクセラレーション・カード (インテル FPGA PAC) を指します。このリファレンス・マニュアルでは、インテル FPGA PAC と表されます。
- **A、B:** 両方のパッケージに適用されます。

頭字語	展開	A/B	説明
AF	アクセラレーター・ファンクション	A、B	FPGA ロジックに実装されるコンパイル済みのハードウェア・アクセラレーター・イメージで、アプリケーションを高速化します。
AFU	アクセラレーター・ファンクショナル・ユニット	A、B	FPGA ロジックに実装されるハードウェア・アクセラレーターで、CPU からアプリケーションの演算動作をオフロードし、パフォーマンスを向上させます。
BBB	インテル FPGA ベーシック・ビルディング・ブロック	A、B	インテル FPGA ベーシック・ビルディング・ブロックは、CCI-P ブリッジと接続可能なコンポーネントとして定義されます。 詳細は、Basic Building Blocks (BBB) for OPAE-managed Intel FPGAs の Web ページを参照ください。
CA	キャッシュ・エージェント	A	キャッシュ・エージェント (CA) は、システム内のコヒーレント・メモリーに対して読み出しおよび書き込みのリクエストを行います。また、システム内のほかのインテル・ウルトラ・パス・インターコネクト (インテル UPI) エージェントが生成したスヌーピングにも対応します。
CCI-P	コア・キャッシュ・インターフェイス	A、B	CCI-P は、AFU がホストと通信するために使用する標準インターフェイスです。
CL	キャッシュライン	A、B	64 バイトのキャッシュライン。
DFL	デバイス・フィーチャー・リスト	A、B	DFL は、機能などのグループ化と、それらを列挙するための構造を定義します。
FIM	FPGA インターフェイス・マネージャー	A、B	FPGA インターフェイス・ユニット (FIU) および、メモリーやネットワークなどで使用する外部インターフェイスを含む FPGA ハードウェアです。 アクセラレーター・ファンクション (AF) は、ランタイムに FIM と接続します。
FIU	FPGA インターフェイス・ユニット	A、B	FIU はプラットフォーム・インターフェイス層であり、PCIe、UPI などのプラットフォーム・インターフェイスと、CCI-P などの AFU 側のインターフェイスの間のブリッジとして機能します。
KiB	1024 バイト	A、B	KiB という用語は 1024 バイトを表し、KB は 1000 バイトを表します。メモリーについて述べる場合は KB が一般的に使用され、KiB は暗黙的に示されます。クロック周波数について述べる場合は kHz が使用され、その場合の K は 1000 です。
Mdata	メタデータ	A、B	これはユーザー定義のフィールドであり、Tx ヘッダーから Rx ヘッダーに渡されます。トランザクション ID またはチャネル ID で、リクエストにタグを付けるために使用されます。
RdLine_I	無効な読み出しライン	A、B	FPGA のキャッシュヒントが Invalid (無効) に設定されたメモリー読み出しリクエストです。このラインは FPGA にキャッシュされませんが、FPGA のキャッシュ・ポリューションを引き起こす可能性があります。

continued...

頭字語	展開	A/B	説明
			注意: キャッシュタグは、インテル・ウルトラ・パス・インターコネクト (インテル UPI) 上の未処理のリクエストすべてのリクエストステータスを追跡します。そのため、完了時に RdLine_I は無効とマークされますが、UPI でリクエストステータスを追跡するためにキャッシュタグを一時的に消費します。この動作はキャッシュラインのエビクションを引き起こし、キャッシュ・ポリューションが発生する場合があります。RdLine_I を使用する利点は、これが CPU ディレクトリーによって追跡されないことです。そのため、CPU からのスヌーピングを防ぎます。 *注意:* キャッシュ機能は、FPGA を統合したインテル Xeon®プロセッサーにのみ適用されます。
RdLine-S	共有される読み出しライン	A	FPGA のキャッシュヒントが Shared (共有) に設定されたメモリー読み出しリクエストです。これを共有状態で FPGA キャッシュに保持する試みが行われます。
Rx	受信	A、B	AFU の視点からの受信または入力
SMBUS	システム管理バス	A	システム管理バス (SMBUS) インターフェイスは、アウトオブバンドの温度の監視、ブートストラップ・プロセス中のコンフィグレーションおよびプラットフォームのデバッグを目的とする動作を行います。
Tx	送信	A、B	AFU の視点からの送信または出力
Upstream	CPU に向かう方向	A、B	CPU に向かう論理方向です。例えばアップストリーム・ポートは、CPU へ向かうポートです。
UMsg	CPU から AFU への順序付けされていないメッセージ	A	64 バイトのペイロードの順序付けされていない通知です。
UMsgH	CPU から AFU への順序付けされていないメッセージヒント	A	このメッセージは、後続する UMsg のヒントです。データペイロードはありません。
Intel UPI	インテル・ウルトラ・パス・インターコネクト	A	インテル コアやその他の IP 間の、インテル独自のコヒーレント・インターコネクト・プロトコルです。
WrLine_I	無効な書き込みライン	A、B	FPGA のキャッシュヒントが Invalid (無効) に設定されたメモリー書き込みリクエストです。FIU は、FPGA キャッシュにデータを保持することを意図せずにデータを書き込みます。
WrLine_M	変更された書き込みライン	A	FPGA のキャッシュヒントが Modified (変更済み) に設定されたメモリー書き込みリクエストです。FIU はデータを書き込み、それを変更済みの状態で FPGA キャッシュに残します。
WrPush_I	無効な書き込みプッシュ	A	FPGA のキャッシュヒントが Invalid (無効) に設定されたメモリー書き込みリクエストです。FIU は、データを FPGA キャッシュに保持することを意図せずに、プロセッサーのラスト・レベル・キャッシュ (LLC) にデータを書き込みます。書き込み先の LLC はかならず、DRAM アドレスが属するプロセッサーに関連付けられた LLC です。

関連情報

Basic Building Blocks (BBB) for OPAE-managed Intel FPGAs

1.1.5. アクセラレーションの用語集

表 4. アクセラレーション・スタック (インテル Xeon CPU & FPGA 対応) の用語集

用語	略語	説明
インテル® アクセラレーション・スタック (インテル® Xeon® CPU & FPGA 対応)	アクセラレーション・スタック	インテル FPGA と インテル Xeon プロセッサー間において最高のパフォーマンスをもたらす接続を可能にする一連のソフトウェア、ファームウェアおよびツールです。
インテル FPGA プログラマブル・アクセラレーション・カード (インテル FPGA PAC)	インテル FPGA PAC	インテル FPGA PAC 搭載の PCIe* アクセラレーター・カードです。 PCIe バス上で インテル Xeon プロセッサーと接続する FPGA インターフェイス・マネージャー (FIM) を含みます。
FPGA を統合したインテル® Xeon® スケーラブル・プラットフォーム	FPGA 統合プラットフォーム	インテル Xeon と FPGA が 1 つのパッケージになったプラットフォームで、インテル・ウルトラ・パス・インターコネクト (UPI) を使用し一貫したメモリービューを共有します。

1.2. 概要

CCI-P は、アクセラレーター・ファンクショナル・ユニット (AFU) のホスト・インターフェイス・バスであり、個別のヘッダーとデータワイヤーを備えます。これは、AFU を FPGA 内の FPGA インターフェイス・ユニット (FIU) に接続するためのものです。このドキュメントでは、CCI-P プロトコルと信号インターフェイスを定義します。これには、リクエストタイプ、ヘッダー・フォーマット、タイミング図、およびメモリーモデルの定義が含まれます。

CCI-P 信号とプロトコルのほかに、このドキュメントでは次の内容を説明します。

1. CCI-P に準拠する AFU をデザインするために必要な必須 AFU レジスター
2. デバイス・フィーチャー・リスト (DFL)—モジュラーデザインおよび、ソフトウェアからの AFU フィーチャーの容易な列挙を促進するレジスター構成の標準
3. インテル FPGA ベーシック・ビルディング・ブロック (BBB)—ハードウェア・モジュールとソフトウェア・モジュールで構成される再利用可能な FPGA ライブラリーを定義するアーキテクチャー

CCI-P は、PCIe や UPI などのさまざまなプラットフォーム・インターフェイスの上に実装可能な抽象化レイヤーを提供します。それにより、CCI-P に準拠する AFU の相互運用をプラットフォーム間で可能にします。

次の表は、AFU に向けた CCI-P インターフェイス固有の機能をまとめています。

表 5. CCI-P の機能

機能	説明
MMIO リクエスト—AFU I/O メモリーに対する CPU の読み出しおよび書き込み	• MMIO 読み出しペイロード—4B、8B • MMIO 書き込みペイロード—4B、8B、64B — MMIO 書き込みは、x86 書き込み結合バッファーによって結合することができます。 — 64B の MMIO 書き込みには、64B の書き込みを生成可能な CPU が必要です。 — FPGA 統合プラットフォームの CPU は、AVX512 を使用して 64B の MMIO 書き込みを生成できます。
メモリーリクエスト	メモリーに対する AFU の読み出しまたは書き込み

continued...

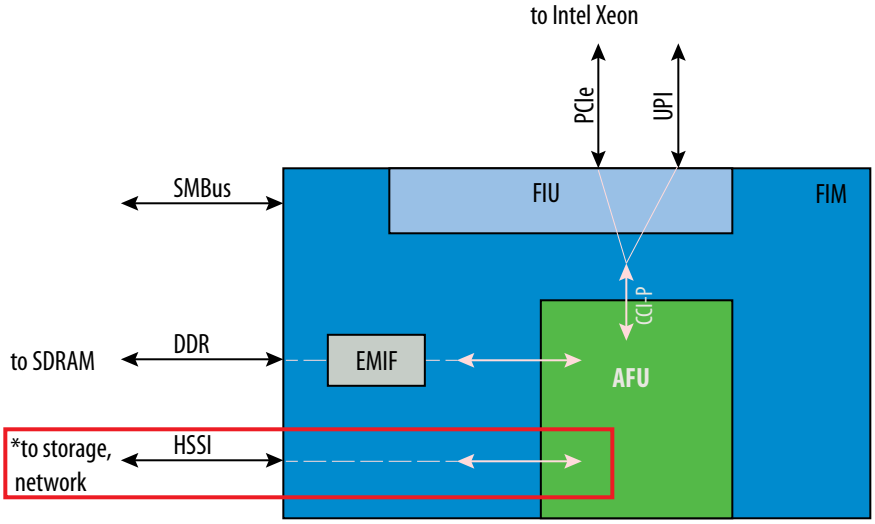
機能	説明
	• アドレス指定モード—物理アドレス指定モード • アドレス指定幅 (CL にアライメントされたアドレス)—42 ビット (CL アドレス) • データ長—64 バイト (1 CL)、128 バイト (2 CL)、256 バイト (4 CL) • バイトアドレス指定—サポートされていません
FPGA キャッシュヒント (FPGA 統合プラットフォームのみ)	AFU は、FIU に対して特定の状態で CL をキャッシュするようリクエストすることができます。VL0 に向けられるリクエストの場合、FIU はヒントとしてリクエストされた状態でデータのキャッシュを試みます。VH0 および VH1 でのキャッシュ・ヒント・リクエストは、WrPush_I を除き無視されます。 *注意:* キャッシュヒントは単なるヒントであり、最終的なキャッシュ状態を保証するものではありません。キャッシュヒントを無視することはパフォーマンスに影響しますが、機能的に影響はありません。 • <request>_I—キャッシュを行う意図はなし • <request>_S—共有状態 (S) でのキャッシュを要求する • <request>_M—変更済みの状態 (M) でのキャッシュを要求する
仮想チャネル (VC)	AFU には、仮想チャネルとして物理リンクが提供されます。AFU は、各メモリーリクエストに対して仮想チャネルを選択できます。 • VL0—低レイテンシー仮想チャネル (UPI にマッピングされます)。(FPGA 統合プラットフォームのみ) • VH0—高レイテンシー仮想チャネル (PCIe0 にマッピングされます)。この仮想チャネルは、大規模なデータ転送を処理するために調整されています。 • VH1—高レイテンシー仮想チャネル (PCIe1 にマッピングされます)。この仮想チャネルは、大規模なデータ転送を処理するために調整されています。(FPGA 統合プラットフォームのみ) • 仮想オート (VA)—FIU は、利用可能な物理リンクすべてにわたって最大の累積帯域幅を達成するために最適化されているポリシーを実装します。 — レイテンシー—高い分散が見込まれます — 帯域幅—高い定常状態の帯域幅が見込まれます
UMsg (FPGA 統合プラットフォームのみ)	CPU から AFU への順序付けされていない通知です。 • UMsg データペイロード—64B • サポートされる UMsg の数—AFU あたり 8
応答順序	順不同の応答
アップストリーム・リクエスト	利用可

関連情報

インテル FPGA ベーシック・ビルディング・ブロック (51 ページ)

1.2.1. FPGA インターフェイス・マネージャー (FIM)

図 -1: FPGA ブロックの概要



インテルの FPGA アクセラレーター・パッケージは、FIM および AFU で構成されます。FIM は、次の内容で構成されます。

- FIU
- 外部メモリーと接続するための EMIF
- 外部トランシーバー接続のための HSSI

FIM の実装に関する特定の内容については、お使いのプラットフォームを参照ください。

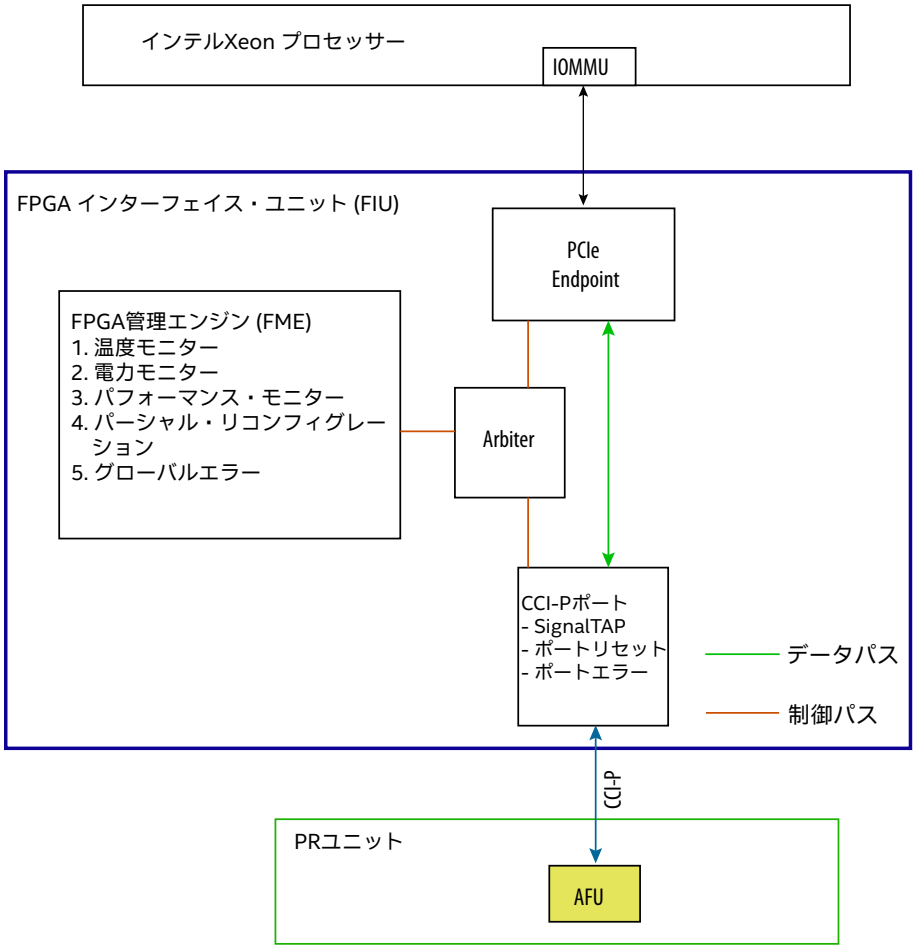
FIU は、AFU とプラットフォーム間のブリッジのような役割を担います。図 1 (9 ページ) は、PCIe、SMBus (管理性を向上するためのもの)、およびホストに対する UPI スタック全体と FIU の接続を表しています。FIM はまた、FPGA 上のすべてのハード IP (PLL など)、パーシャル・リコンフィグレーション (PR) エンジン、JTAG アトム、IO、温度センサーを所有します。FIM は起動時に最初にコンフィグレーションされ、プラットフォームの電源を再投入するまで持続しますが、AFU は動的にリコンフィグレーション可能です。インテルのパーシャル・リコンフィグレーション技術は動的なリコンフィグレーション機能を可能にし、AFU はパーシャル・リコンフィグレーション領域として定義され、FIM は静的領域として定義されます。AFU と FIU 間のインターフェイスはホットプラグ機能を提供し、トラフィックの一時停止およびパーシャル・リコンフィグレーション後の AFU の再列挙を行います。

FIM は、プラットフォームの機能に応じて 1 つもしくは複数のインターフェイスを AFU に提供します。このドキュメントでは、AFU が インテル Xeon プロセッサーと通信するためのインターフェイスである CCI-P に焦点を当てています。CCI-P はダイレクト I/O 向けインテル・バーチャライゼーション・テクノロジー (インテル VT-d) を使用し、アドレス空間の分離と保護を提供します。CCI-P は、PCIe の機能に定義されているものと同じ保護を備えます。AFU は、PCIe の列挙および VT-d の観点において単機能のデバイスです。
また FIU は場合によっては、エラー監視とレポート、電力と温度の監視、コンフィグレーション・ブートストラップ、ビットストリームのセキュリティー・フロー、リモートデバッグなどの管理性を向上させるための機能を実装し、データセンター環境での FPGA の展開および管理を容易にします。また、FIU は一部の実装において、ボード管理コントローラー (BMC) とのアウトオブバンド通信インターフェイスを備えている場合があります。

1.2.2. インテル FPGA インターフェイス・ユニット (FIU)

1.2.2.1. インテル FPGA PAC 向け FIU

図 -2: インテル FPGA PAC 向け FIU のブロック図



インテル FPGA PAC は、インテル Xeon プロセッサーに PCIe 物理リンクを介して接続します。図 2 (10 ページ) の インテル FPGA PAC FIU のブロック図は、CCI-P を PCIe リンクにマッピングするブロックのみを示しています。この図には、AFU に向かうボードローカルのメモリーの FIM ブロックは示されていません。インテル FPGA PAC 向け FIU は、1 つの物理リンクを CCI-P にマッピングするシンプルな機能を備えます。

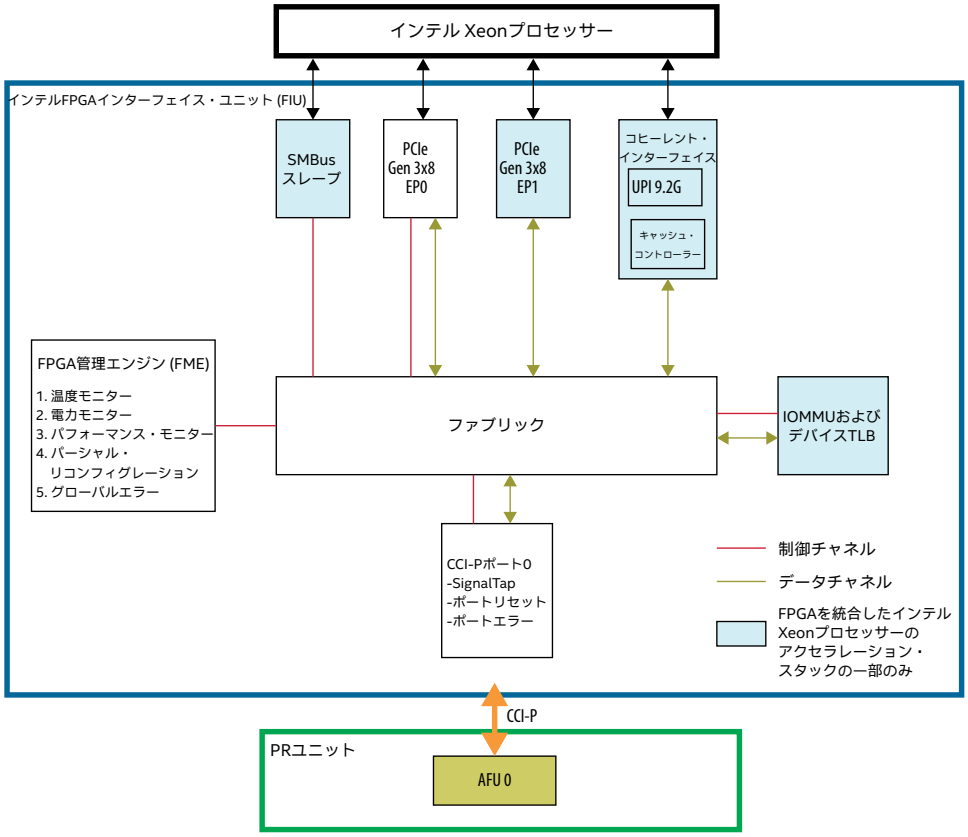
インテル FPGA PAC FIU は、CCI-P 仮想チャネルの VH0 および VA を PCIe リンクにマッピングします。仮想オートチャネル (VA) は、任意のプラットフォームのすべての利用可能なチャネルにわたってリクエストを最適にマッピングし、最大の帯域幅を実現します。

仮想チャネルについての詳細は、「CCI-P の機能の概要」の章を参照ください。ダウンストリームの PCIe 制御パスは、FPGA 管理エンジン (FME)、CCI-P ポートおよび AFU にアドレスマッピングされます。

FME は、AFU のエラー、パフォーマンス、電力、温度の監視およびパーシャル・リコンフィグレーションの機能を提供します。CCI-P ポートモジュールは、ポートごとのリセット、休止、エラーの監視および、ネットワーク経由での Signal Tap を使用するリモートデバッグを実装します。

1.2.2.2. インテル FPGA 統合プラットフォーム向け FIU

図 -3: インテル FPGA 統合プラットフォーム向け FIU のブロック図



FPGA 統合プラットフォームは、FPGA をプロセッサーに接続するリンクを 3 つ備えています。1 つはインテル UPI コヒーレント・リンク、残りの 2 つは PCIe Gen3x8 リンクです。これらの 3 つのリンクを CCI-P インターフェイスにマッピングするのは FIU の機能であり、それによって AFU は、3 つのリンクの総帯域幅に等しい帯域幅を持つホスト・プロセッサーへの単一の論理通信インターフェイスを認識します。図 1 (9 ページ) は、UPI および PCIe リンクの CCI-P へのマッピングに関連する FIU ロジックのみを示しています。

FIU は、CCI-P へのマッピングを提供するための次の機能を実装しています。

- 単一の論理アップストリーム・リンク: CCI-P は、3 つの物理リンクを 4 つの仮想チャネルにマッピングします (PCIe0 を VH0、PCIe1 を VH1、UPI を VL0、すべての物理リンクを VA に)。VA を使用している AFU は物理リンクに依存せず、FPGA で利用可能なアップストリームの帯域幅全体を使用できる単一の論理リンクと接続します。VA は重み付きデマルチプレクサーを実装し、リクエストをすべての物理リンクにルーティングします。プラットフォームに依存しない AFU をデザインするには、VA 仮想チャネルの使用が推奨されます。
- 単一の制御ポイント: FIU は、単一の制御インターフェイスをシステムのソフトウェア・スタックに登録します。ドライバーと FIU の相互通信はすべて、PCIe-0 に向けられます。AFU は PCIe-0 で検出され、列挙されます。
- 統一されたアドレス空間を提供する VT-d の単一 ID: アップストリームのリクエストはすべて、単一の関数番号をアドレス変換に使用します。そのため、FPGA を統合したインテル® Xeon® スケーラブル・プラットフォームは、PCIe-0 および PCIe-1 ルートポートで IOMMU を無効にし、代わりに IOMMU を FIU でインスタンス化します。この IOMMU は、すべての 3 つの物理リンクを介してアップストリームに向かうリクエストの変換に使用されます。

インテル FPGA PAC と同様に、FPGA 統合プラットフォームもまた、FME および CCI-P ポートで提供されるサービス一式を実装し、FPGA の展開および管理を行います。

1.2.2.3. FIU 機能の比較

次の表において、インテル FPGA PAC と FPGA 統合プラットフォームでサポートされる機能を比較します。

表 6. FIU 機能の比較

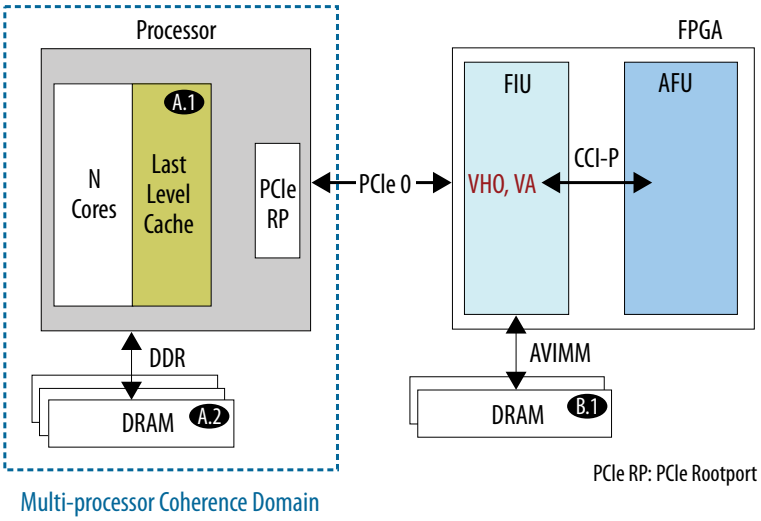
FIU の機能	インテル FPGA PAC でサポートされる	FPGA 統合プラットフォームでサポートされる
統一アドレス空間	はい	
AFU 向けインテル VT-d	はい	
パーシャル・リコンフィグレーション	はい	
リモートデバッグ	はい	
FPGA キャッシュサイズ	該当なし	128 KiB 直接マッピング
CCI-P		
メモリーマップド I/O (MMIO) 読み出しおよび書き込み	はい	
CPU への AFU 割り込み	はい	いいえ
CPU から AFU への UMsg	いいえ	はい
CCI-P メモリーリクエスト		
データ転送サイズ	64 バイト (1 CL)、128 バイト (2 CL)、256 バイト (4 CL)	
アドレス指定モード	物理アドレス指定モード	
アドレス指定幅 (CL にアライメントされたアドレス)	42 ビット	
キャッシュヒント	いいえ	はい
仮想チャネル	VA、VH0	VA、VH0、VH1、VL0

1.2.3. メモリーおよびキャッシュ階層

CCI-P プロトコルはキャッシュヒントのメカニズムを提供します。AFU に精通しているデベロッパーは、このメカニズムを使用しパフォーマンスを調整することができます。この章では、インテル FPGA PAC および FPGA 統合プラットフォームのメモリーとキャッシュ階層について説明します。CCI-P が提供する制御メカニズムについては、次の「インテル FPGA PAC」と「FPGA 統合プラットフォーム」の章で説明します。

インテル FPGA PAC

図 -4: **インテル FPGA PAC のメモリー階層**



上の図は、単一プロセッサーの インテル Xeon プラットフォームにおける インテル FPGA PAC のメモリーおよびキャッシュ階層を表しています。インテル FPGA PAC には、次の 2 つのメモリーノードがあります。

- ホストメモリーと呼ばれている、プロセッサーの **SDRAM (Synchronous Dynamic Random Access Memory)**
- ローカルメモリーと呼ばれている、FPGA に接続する **SDRAM**

AFU は、リクエストをローカルメモリーにルーティングする必要があるか、CPU メモリーにルーティングする必要があるかを決定します。

ローカルメモリー **(B.1)** は、ホストメモリー **(A.2)** とは異なるアドレス空間にあります。ローカルメモリーをターゲットにする AFU リクエストはかならず、図 4 (13 ページ) において **(B.1)** と示されている **SDRAM** によって直接処理されます。

注意: ローカル・メモリー・アクセス・パスにともなうキャッシュはありません。

PCIe を介して CPU メモリーに向かう AFU リクエストは、図 4 (13 ページ) で示されるように、プロセッサー側で処理することができます。

図中 **(A.1)** で示される**ラスト・レベル・キャッシュ**の場合

- 受信する読み出しリクエストのレイテンシーは、**SDRAM** (図中 **A.2**) から読み出すよりも小さくなります。
- 書き込みリクエストヒントを使用し、書き込まれたデータをどのように処理するかを**ラスト・レベル・キャッシュ**に指示できます (キャッシュ可または不可、および局所性など)。

リクエストが**ラスト・レベル・キャッシュ**でミスになった場合、**SDRAM** で処理することが可能です。

詳細は、CCI-P プロトコルの定義にある WrPush_I リクエストを参照ください。

FPGA 統合プラットフォーム

図 -5: FPGA 統合プラットフォームのメモリー階層

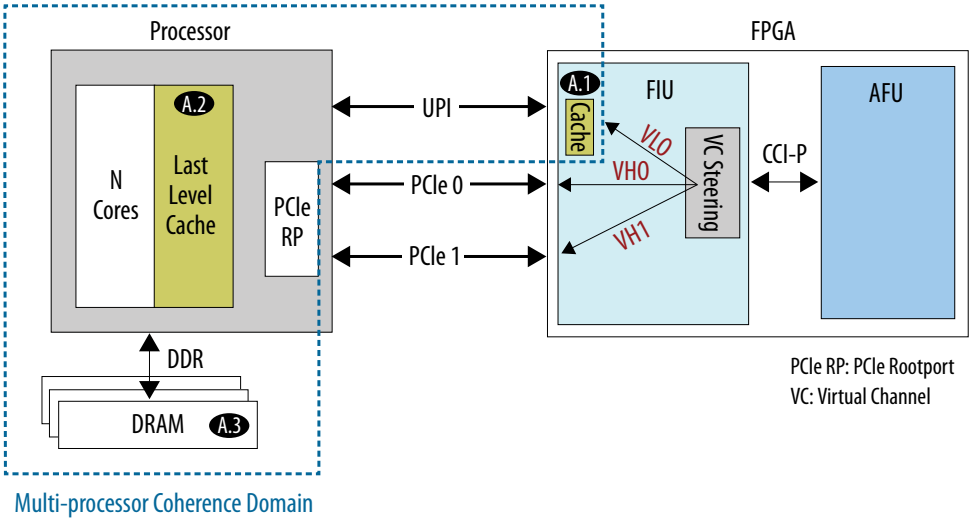


図 5 (14 ページ) は、1 つの インテル Xeon プロセッサーを備える FPGA 統合プラットフォームの AFU から見た 3 つのレベルのキャッシュとメモリー階層を表しています。単一プロセッサーの FPGA 統合プラットフォームは、1 つのメモリーノードのみを備えています (図中 **A.3** のプロセッサー側の **SDRAM**)。インテル UPI のコヒーレント・リンクは、図 5 (14 ページ) の緑の点線で示されるように、インテル Xeon プロセッサーのコヒーレンシー・ドメインを FPGA に拡張します。UPI のキャッシュ・エージェントは、CPU メモリーのそのほかの部分と一貫性のある FIU に FPGA キャッシュを保持します。CPU メモリーをターゲットにするアップストリームの AFU リクエストは、以下で処理することができます。

- FPGA キャッシュ **(A.1)** ―インテル UPI のコヒーレント・リンクは、インテル Xeon プロセッサーのコヒーレンシー・ドメインを FPGA キャッシュに拡張します。FPGA キャッシュでヒットするリクエストのレイテンシーは最も小さく、また、帯域幅は最も高くなります。VL0 仮想チャネルを使用する AFU リクエストおよび UPI パスの使用が選択された VA リクエストは、FPGA キャッシュをまず検索し、ミスになった場合にのみチップからプロセッサーに送信されます。
- プロセッサー側のキャッシュ **(A.2)** ―プロセッサー側のキャッシュでヒットする読み出しリクエストのレイテンシーは、FPGA キャッシュでヒットする場合よりも大きくなりますが、プロセッサーの **SDRAM** から読み出す場合よりも小さくなります。書き込みリクエストヒントを使用し、プロセッサー側のキャッシュへの書き込みを指示できます。詳細は、CCI-P プロトコルの定義にある WrPush_I リクエストを参照ください。
- プロセッサーの **SDRAM (A.3)** ―プロセッサー側のキャッシュでミスとなったリクエストは、**SDRAM** で処理されます。

データアクセスのレイテンシーは、**(A.1)** から **(A.3)** に増加します。

注意: ほとんどの AFU は、VL0、VH0、VH1 の使用を明示的に選択するのではなく、VA 仮想チャネルを選択することで最大の帯域幅を実現します。FIU に実装されている VC ステアリング・ロジックはプラットフォームに対して調整されており、物理リンクのレイテンシーと効率特性、物理リンクの使用率およびトラフィックの分散を考慮し最大限の帯域幅を提供します。

VC ステアリング・ロジックの制限の 1 つに、ステアリングの決定においてキャッシュの局所性を考慮しないことがあります。VC のステアリングの決定はキャッシュ検索の前に行われます。つまり、キャッシュラインが FPGA キャッシュにある場合でも、リクエストが VH0 または VH1 に向けられる場合があることを意味します。そのようなリクエストでは、リクエストを完了させるために場合によってはプロセッサーが FPGA キャッシュをスヌーピングする必要があるため、レイテンシーがさらに課される可能性があります。AFU がすでにアクセスの局所性を認識している場合は、VL0 仮想チャネルを使用しキャッシュの局所性を活用するほうが有効な場合があります。

関連情報

Avalon® インターフェイスの仕様書

1.3. CCI-P インターフェイス

CCI-P は次のメモリーアドレス空間に実装します。

- メインメモリー
- メモリーマップド I/O (MMIO)

表 7. CCI-P のメモリー・アクセス・タイプ

メモリータイプ	説明
メインメモリー	メインメモリーは、プロセッサーに接続され、オペレーティング・システムに公開されるメモリーです。AFU からメインメモリーへのリクエストはアップストリーム・リクエストと呼ばれます。後続の章では、メインメモリーは単純にメモリーと呼ばれます。
メモリーマップド I/O	I/O メモリーは、ホストから AFU への CCI-P リクエストとして実装されます。MMIO は一般的に、AFU 制御レジスターとして使用されます。このメモリーの実装方法および編成方法は、AFU デベロッパーによって異なります。AFU は、M20K または MLAB のロジックを選択することが可能です。 CCI-P インターフェイスは、メモリーマップド I/O (MMIO) リクエストを使用して、I/O メモリーにアクセスするリクエスト形式を定義します。プロセッサーから I/O メモリーへのリクエストは、ダウンストリーム・リクエストと呼ばれます。 AFU の MMIO アドレス空間サイズは 256 kB です。

図 -6: CCI-P 信号

次の図は、3 つの Tx チャネル、2 つの Rx チャネル、およびそのほかのいくつかの制御信号にグループ化されたすべての CCI-P 信号を表しています。

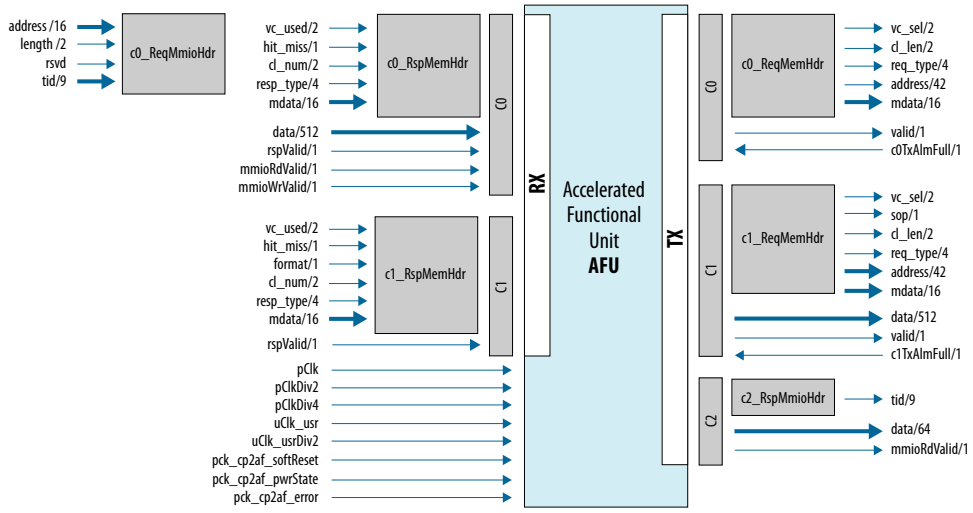


表 8. CCI-P 信号

信号の種類	説明
Tx または Rx	フローの方向は AFU の視点からのものです。Tx は AFU から FIU へのフローです。Rx は FIU から AFU へのフローです。
チャネル	信号をグループ化したもので、それらによってリクエストまたは応答を完全に定義します。

1.3.1. 信号情報

- CCI-P 信号はすべて、`pClk` に同期している必要があります。
- インテルでは、`ccip_if_pkg.sv` ファイル内で定義されている CCI-P 構造を使用することを推奨しています。このファイルは RTL パッケージに含まれています。
- AFU の入力および出力信号はすべて、レジスターする必要があります。
- RSVD でマークされている AFU 出力ビットは予約されており、0 に駆動する必要があります。
- RSVD-DNC でマークされている AFU 出力はドント・ケア・ビットです。AFU は、0 または 1 のいずれかを駆動できます。
- RSVD でマークされている AFU 入力ビットは、AFU でドントケア (X) として扱う必要があります。
- 明示的に示されない限り信号はすべてアクティブ High です。アクティブ Low の信号には、末尾に _n を付けます。

次の図は、`ccip_std_afu` モジュールのポートマップを表しています。ここで AFU をインスタンス化する必要があります。以降の章では、インターフェイス信号について説明します。

図 -7: ccip_std_afu ポートマップ

```
$ module ccip_std_afu(
  // CCI-P Clocks and Resets
  input           logic             pClk,          //          CCI-P clock domain. Primary
                                                   // interface clock
  input           logic             pClkDiv2,      //          CCI-P clock domain
  input           logic             pClkDiv4,      //          CCI-P clock domain
  input           logic             uClk_usr,      // User clock domain
  input           logic             uClk_usrDiv2,  // User clock domain. Half the programmed
                                                   // frequency
  input           logic             pck_cp2af_softReset,      // CCI-P ACTIVE HIGH Soft
                                                              // Reset
  input           logic [1:0]       pck_cp2af_pwrState,       // CCI-P AFU Power State
  input           logic             pck_cp2af_error,          // CCI-P Protocol Error
                                                              // Detected

  // Interface structures
  input            t_if_ccip_Rx      pck_cp2af_sRx,        // CCI-P Rx Port
  output           t_if_ccip_Tx      pck_af2cp_sTx         // CCI-P Tx Port
);
```

1.3.2. メインメモリーに対する読み出しおよび書き込み

CCI-P は物理アドレスを使用し、プロセッサーのメインメモリーにアクセスするアップストリームのメモリー読み出しおよび書き込みリクエストを定義します。非仮想化システムの場合、AFU はホストの物理アドレスを駆動することが求められます。仮想化システムの場合、AFU はゲストの物理アドレスを駆動することが求められます。アドレス指定モードは、AFU ハードウェア・デベロッパーに対してトランスペアレントです。ソフトウェア・アプリケーション・デベロッパーは、ソフトウェアが AFU に物理アドレスを提供することを確認する必要があります。

CCI-P の仕様は、アップストリームのメモリーリクエストに弱いメモリー整合性モデルを定義します。

詳細は、「順序付けの規則」の章を参照してください。

関連情報

順序付けの規則 (39 ページ)

1.3.2.1. メインメモリーからの読み出し

AFU は、`pck_af2cp_sTx.c0` を使用し、CCI-P `Channel 0 (C0)` を介してメモリー読み出しリクエストを送信します。また、`pck_cp2af_sRx.c0` を使用し、`C0` を介して応答を受信します。

`c0_ReqMemHdr` 構造は、フラット・ビットベクトルから読み出しリクエストフィールドまでの便利なマッピングを提供します。AFU は `pck_af2cp_sTx.c0.valid` 信号をアサートし、メモリー読み出しリクエストを `hdr` で駆動します。`req_type` はキャッシュヒントを指定します。`RDLINE_I` はキャッシュしないことを指定し、`RDLINE_S` は共有状態でキャッシュすることを指定します。`mdata` フィールドはユーザー定義のリクエスト ID であり、応答とともに変更されずに返されます。

`c0_RspMemHdr` 構造は、フラット・ビットベクトルから応答フィールドまでの便利なマッピングを提供します。FIU は `pck_cp2af_sRx.c0.resp_valid` 信号をアサートし、読み出し応答とデータを `hdr` および `data` でそれぞれ駆動します。`resp_type` はデコードされ、応答タイプを識別します (メモリー読み出しまたは Umsg)。読み出し応答順序は保証されていないため、`mdata` フィールドを定義し、リクエストで送信された値と同じ値が返されるようにする必要があります。

例えば、AFU は `mdata` を使用し、AFU 内部のリクエストおよび応答をルーティングできます。すなわち、情報は次にトリガーされる動作で運ばれます。

1.3.2.2. メインメモリーへの書き込み

AFU は、pck_af2cp_sTx.c1 を使用し、CCI-P Channel 1 (C1) を介してメモリー書き込みリクエストを送信します。また、pck_cp2af_sRx.c1 を使用し、C1 を介して書き込み完了確認応答を受信します。

c1_ReqMemHdr 構造は、フラット・ビットベクトルから書き込みリクエストフィールドまでの便利なマッピングを提供します。AFU は pck_af2cp_sTx.c1.valid 信号をアサートし、メモリー書き込みリクエストとデータをそれぞれ hdr および data で駆動します。req_type 信号は、リクエストタイプとキャッシュヒントを指定します。

- WrLine_I は、FPGA キャッシュの意図がないことを指定します。
- WrLine_M は、FPGA キャッシュを M 状態で保持する意図を指定します。
- WrPush_I は、プロセッサー側のキャッシュでキャッシュする意図を指定します。

c1_ReqMemHdr 構造はまた、モードフィールドの **pck_af2cp_sTx.c1.hdr.mode** を提供し、発行するメモリー書き込みリクエストの種類を指定します。次の 2 つのメモリー・リクエスト・モードがあります。

- eMOD_CL は、単一または複数のキャッシュにアライメントされた書き込みを指定します。
- eMOD_BYTE は、バイト・イネーブル書き込みを指定します。

  注意: このメモリー・リクエスト・モードは、インテル PAC (インテル Arria® 10 GX FPGA 搭載版) では利用できません。

c1_RspMemHdr 構造は、フラット・ビットベクトルから応答フィールドまでの便利なマッピングを提供します。FIU は pck_cp2af_sRx.c1.resp_valid 信号をアサートし、読み出し応答を hdr で駆動します。resp_type フィールドは、応答タイプをデコードするためにデコードされます (メモリー書き込み、書き込みフェンスまたは割り込み)。

WrFence は、メモリー書き込みリクエストをグローバルに可視化するために使用されます。WrFence リクエストは、データペイロードとアドレスを受け入れないことを除いて、メモリー書き込みリクエストと同じフローに従います。

詳細については、*Tx ヘッダーのフォーマット*内の書き込みリクエストヘッダーのフォーマットを参照してください。

関連情報

Tx ヘッダーのフォーマット (23 ページ)

1.3.3. 割り込み

割り込みは、FPGA 統合プラットフォームではサポートされていません。

AFU は割り込み ID を使用し、Tx チャネル C1 を介して割り込みを送信します。また、Rx チャネル C1 を介して応答を受信します。

AFU では、発行される割り込み ID は一度に 1 つのみ持つようにしてください。発行された割り込み ID の応答が返されるまで待機せず、AFU が同じ ID で別の割り込みを発行した場合、ホストは 2 番目の割り込みの到着を認識しない可能性があります。AFU が同じ割り込み ID を使用して割り込みを発行する前に、ソフトウェアによって割り込みリクエストを処理することが推奨されます。

1.3.4. UMsg

注意: UMsg は、FPGA 統合プラットフォームでのみサポートされています。

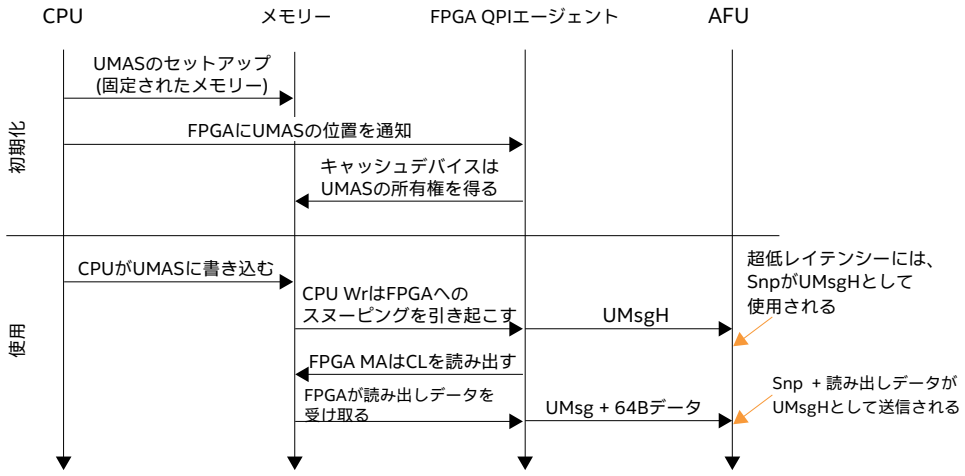
UMsg は、CCI-P の読み出し帯域幅を消費することなく、AFU のスピンループと同じ機能を提供します。これはスピンループの最適化と考えることができ、FPGA キャッシュ・コントローラー内の監視エージェントが、ドライバーによって割り当てられたキャッシュラインへのスヌーピングを監視しています。キャッシュラインへのスヌーピングを検出すると、データを読み出し、AFU に UMsg を送信します。

UMsg フローはキャッシュ・コヒーレンシー・プロトコルを使用し、CPU から AFU への順序付けされていない高速通信パスを実装します。このプロセスは図 8 (19 ページ) に示されるとおり、2 つの段階で構成されます。

最初の段階は初期化です。ここで SW は UMsg アドレス空間 (UMAS) を固定し、UMAS の開始アドレスを FPGA キャッシュ・コントローラーと共有します。これが完了すると、FPGA キャッシュ・コントローラーは UMAS の各キャッシュラインを読み出し、それを FPGA キャッシュに共有状態で配置します。

2 番目は実際に使用する段階であり、CPU は UMAS に書き込みを行います。CPU が UMAS に書き込むと、FPGA キャッシュへのスヌーピングが生成されます。FPGA はスヌーピングに応答し、ラインを無効としてマークします。CPU の書き込みリクエストが完了し、データがグローバルに可視化されます。UMAS アドレス範囲のスヌーピングは監視エージェント (MA) をトリガーします。MA はキャッシュライン (CL) の読み出しリクエストを CPU に送信し、また、オプションで AFU に UMsg をヒント (UMsgH) とともに送信します。読み出しリクエストが完了すると、64 B のデータをともなう UMsg が AFU に送信されます。

図 -8: UMsg の初期化および使用のフロー



機能的に UMsg は、スピンループもしくは、インテル Xeon プロセッサーの monitor および mwait 命令と同等です。

UMsg の主な特性

- マルチスレッド・アプリケーションの異なるアドレスへのスピンループに相対的な順序付けの保証がないように、異なるアドレスへの UMsg に順序付けの保証はありません。
- UMAS CL への CPU の書き込みがすべて、対応する UMsg になるわけではありません。AFU はそれまでに行われた CL の値の変更を見逃す可能性がありますが、CL の最新データを読むことが保証されています。前述のとおり、これをスピンループのように考えると理解しやすくなります。プロデューサー・スレッドがフラグ CL を複数回更新した場合、ポーリングスレッドはそれまでに行われた値の変更を見逃す可能性がありますが、最新の値を読むことが保証されています。

次に、UMsg にマッピングされる可能性のある記述子キューポインターのソフトウェア更新を使用例として示します。ポインターは常に増加することが想定されます。UMsg は、AFU がポインターの最終の値を読むことを保証しますが、それまでにポインターに行われた更新を見逃す可能性があります。ただし、これは許容されます。

1. UMsg は FPGA キャッシュを使用するため、キャッシュ・ポリューションを引き起こす可能性があります。キャッシュ・ポリューションとは、プログラムが不必要にデータをキャッシュにロードし、ほかの必要なデータを追い出すことでパフォーマンスが低下することです。
2. CPU は誤ったスヌーピングを示す場合があるため、UMsgH は手がかりとなる情報として扱う必要があります。つまり、投機的実行または UMsgH に基づくプリフェッチを開始することは可能ですが、結果を確定する前に UMsg を待つ必要があります。
3. UMsg のレイテンシーは、RdLine_S を使用する AFU の読み出しポーリングと同じですが、読み出しトラフィックに使用できる CCI-P チャネルの帯域幅を節約します。

1.3.5. I/O メモリーへの MMIO アクセス

CCI-P は、AFU レジスターファイルにアクセスするための MMIO 読み出しおよび書き込みリクエストを定義します。MMIO リクエストは、単一の PCIe チャネルを介して CPU から AFU へルーティングされます。

MMIO 読み出し

AFU は、`pck_cp2af_sRx.c0` を介して MMIO 読み出しリクエストを受信します。CCI-P は、`mmioRdValid` をアサートし、MMIO 読み出しリクエストを `hdr` で駆動します。`c0_ReqMmioHdr` 構造は、フラット・ビットベクトルから MMIO リクエストフィールド (`address`、`length`、`tid`) までの便利なマッピングを提供します。

AFU は、MMIO 読み出し応答を `pck_af2cp_sTx.c2` で駆動します。AFU は `mmioRdValid` をアサートし、応答ヘッダーとデータをそれぞれ `hdr` および `data` で駆動します。AFU は対応する応答とともに、応答をリクエストに関連付けるために使用したリクエスト `tid` を返すことが予期されます。

次に、CCI-P MMIO 読み出しリクエストの主要な属性を説明します。

- サポートされるデータ長は 4 バイトおよび 8 バイトです。
- 応答の長さはリクエストの長さと一致する必要があります。例えば、8 バイトの MMIO 読み出しリクエストに対して 4 バイトの応答を 2 つ返すことは不正です。
- 未処理の MMIO 読み出しリクエスト数は、最大 64 に制限されます。
- 未定義の AFU レジスターに対する MMIO 読み出しの場合でも応答を返します。

MMIO 書き込み

AFU は、MMIO 書き込みリクエストを pck_cp2af_sRx.c0 を介して受信します。CCI-P は mmioWrValid をアサートし、MMIO 書き込みリクエストヘッダーとデータを、それぞれ hdr および data で駆動します。c0_ReqMmioHdr 構造は、フラット・ビットベクトルから MMIO リクエストフィールド (address、length、tid) までの便利なマッピングを提供します。MMIO 書き込みリクエストはポストされ、AFU からの応答は予期されません。

サポートされるデータ長は、4 バイト、8 バイト、および 64 バイトです。

注意: インテル Xeon プラットフォームすべてでサポートされているわけではありません。

MMIO アクセスを実装する際の注意点

以下は、AFU MMIO レジスターファイルをデザインする際に考慮すべき重要な内容です。

- DFH を実装するには、AFU が 8 バイトのアクセスをサポートしていることが必須です。
- 4 バイトの MMIO アクセスに対するサポートはオプションです。AFU デベロッパーはソフトウェア・アプリケーション・デベロッパーと調整し、4 バイトのアクセスを回避してください。
- AFU は、到着した MMIO リクエストを連続して遅延なく受け入れることが可能です。
- アライメントされていない MMIO アクセスは、エラーになります。ソフトウェア・アプリケーション・デベロッパーは、MMIO アドレスがリクエストの長さにアライメントされていることを確認する必要があります。例えば、8 バイトの MMIO リクエストのバイトアドレスは、8 の倍数である必要があります。つまり、バイトアドレス [2:0] は 0 でなければなりません。

1.3.6. CCI-P Tx 信号

図 -9: ccip_if_pkg.sv 内の Tx インターフェイス構造

```
$ typedef struct packed {
    t_if_ccip_c0_Tx      c0;
    t_if_ccip_c1_Tx      c1;
    t_if_ccip_c2_Tx      c2;
} t_if_ccip_Tx;
```

Tx チャネルは 3 つあります。

C0 および C1 Tx チャネルは、メモリーリクエストに使用されます。C0 および C1 Tx チャネルはどちらも、独立したフロー制御を備えます。C0 Tx チャネルはメモリー読み出しリクエストに使用され、C1 Tx チャネルはメモリー書き込みリクエストに使用されます。

C2 Tx チャネルは、MMIO 読み出し応答を FIU に返すために使用されます。CCI-P ポートは C2 で応答を受け入れることを保証しているため、フロー制御はありません。

図 -10: ccip_if_pkg.sv 内の Tx チャネル構造

```
// Channel 0 : Memory Reads
typedef struct packed {
    t_ccip_c0_ReqMemHdr  hdr;            // Request Header
    logic                valid;          // Request Valid
} t_if_ccip_c0_Tx;
// corresponding AlmostFull inside t_if_ccip_Rx.c0TxAlmFull

// Channel 1 : Memory Writes
typedef struct packed {
    t_ccip_c1_ReqMemHdr  hdr;            // Request Header
    t_ccip_c1Data        data;           // Request Data
    logic                valid;          // Request Wr Valid
} t_if_ccip_c1_Tx;
// corresponding AlmostFull inside t_if_ccip_Rx.c1TxAlmFull

// Channel 2 : MMIO Read response
typedef struct packed {
    t_ccip_c2_RspMmioHdr    hdr;          // Response Header
    logic                   mmioRdValid;  // Response Read Valid
    t_ccip_mmioData         data;         // Response Data
} t_if_ccip_c2_Tx;
```

各 Tx チャネルには Valid 信号があり、構造内の対応するヘッダーおよびデータ信号を分類します。

次の表に、CCI-P Tx インターフェイスを構成する信号を示します。

表 9. Tx チャネルの説明 (チャネル 0)

信号	幅 (ビット)	方向	説明
pck_af2cp_sTx.c0.hdr	74	出力	チャネル 0 のリクエストヘッダーです。表 18 (26 ページ) を参照ください。
pck_af2cp_sTx.c0.valid	1	出力	1 に設定されている場合、チャネル 0 のリクエストヘッダーが有効であることを示します。
pck_cp2af_sRx.c0TxAlmFull	1	入力	1 に設定されている場合、Tx チャネル 0 はフルに近い状態です。この信号が設定されると、8 つまでのリクエストの送信が AFU に許可されます。 0 に設定されている場合、AFU はリクエストの送信をすぐに開始することができます。

表 10. Tx チャネルの説明 (チャネル 1)

信号	幅	方向	説明
pck_af2cp_sTx.c1.hdr	80	出力	チャネル 1 のリクエストヘッダーです。表 12 (23 ページ) を参照ください。
pck_af2cp_sTx.c1.data	512	出力	チャネル 1 のデータです。
pck_af2cp_sTx.c1.valid	1	出力	1 に設定されている場合、チャネル 1 のリクエストヘッダーおよびデータが有効であることを示します。
pck_cp2af_sRx.c1TxAlmFull	1	入力	1 に設定されている場合、Tx チャネル 1 はフルに近い状態です。この信号が設定されると、8 つまでのリクエストまたはデータの送信が AFU に許可されます。 0 に設定されている場合、AFU はリクエストの送信をすぐに開始することができます。

表 11. Tx チャネルの説明 (チャネル 2)

信号	幅 (ビット)	方向	説明
pck_af2cp_sTx.c2.hdr	9	出力	チャネル 2 の応答ヘッダーです。表 12 (23 ページ) を参照ください。
pck_af2cp_sTx.c2.mmioRdValid	1	出力	1 に設定されている場合、チャネル 2 の応答ヘッダーおよびデータが有効であることを示します。
pck_af2cp_sTx.c2.data	64	出力	チャネル 2 のデータで、AFU が FIU に返す MMIO 読み出しデータです。 4 バイトの読み出しの場合、データはビット [31:0] で駆動する必要があります。 8 バイトの読み出しの場合、AFU は 8 バイトのデータ応答を 1 つ駆動しなければなりません。応答を 4 バイトの応答 2 つに分割することはできません。

1.3.7. Tx ヘッダーのフォーマット

表 12. Tx ヘッダーフィールドの定義

フィールド	説明
mode	メモリー・アクセス・モード • eMOD_CL (1'b0)—キャッシュにアライメントされた書き込みです。cl_len で指定されるホストメモリーへの 1、2、4 のキャッシュライン書き込みを有効にします。 • eMOD_BYTE(1'b1)—バイトにアライメントされた書き込みです。byte_start および byte_len フィールドで指定されるようにラインの連続するサブセットをホストメモリーに書き込みます。**注意:**このメモリー・リクエスト・モードは、インテル PAC (インテル Arria 10 GX FPGA 搭載版) では利用できません。 *注意:* eMOD_BYTE に設定されている場合、キャッシュ長 (cl_len) は 0 に設定する必要があります。これは、単一のキャッシュライン書き込みを示します。 eMOD_CL に設定されている場合、byte_len および byte_start は 0 に設定する必要があります。 *注意:* マルチ・キャッシュ・ライン書き込みの途中でモードを変更することはできません。 *注意:* このフィールドは、インテル FPGA PAC N3000 および インテル PAC (インテル Arria 10 GX FPGA 搭載版) に RSVD0 です。
byte_start	バイト・アクセス・モードのバイト開始インデックスです。 • ホストメモリーに書き込む 512 ビット TX データバスの最初のバイトのインデックスを示します。 • mode = eMOD_CL の場合、byte_start は 0 に設定されている必要があります。 • mode = eMOD_BYTE の場合、byte_start は byte_enable モードに設定され、正当な範囲は 0 から 63 です。 *注意:* このフィールドは、インテル FPGA PAC N3000 および インテル PAC (インテル Arria 10 GX FPGA 搭載版) に RSVD0 です。
byte_len	バイト・アクセス・モード (mode = eMOD_BYTE) のバイト長です。 • ホストメモリーに書き込むバイト数を示します。 • byte_len—byte_start インデックスの左 (最上位) のバイト数を指定し、メモリーリクエストをバイト・アクセス・モードで含めます。 • mode = eMOD_CL の場合、byte_len は 0 に設定されている必要があります。 • mode = eMOD_BYTE の場合、byte_len はバイト・イネーブル・モードに設定され、正当な範囲は 1 から 63 です。 *注意:* このフィールドは、インテル FPGA PAC N3000 および インテル PAC (インテル Arria 10 GX FPGA 搭載版) に RSVD0 です。
mdata	メタデータです。ユーザー定義のリクエスト ID で、リクエストから応答ヘッダーに変更されずに返されます。 C1 Tx でのマルチ CL 書き込みの場合、mdata は sop=1 の場合にのみヘッダーに有効です。

continued...

フィールド	説明
tid	トランザクション ID です。AFU は、tid MMIO 読み出しリクエストを応答ヘッダーに返す必要があります。これは、応答をリクエストに対して一致させるために使用されます。
vc_sel	選択される仮想チャネルです。 • 2'h0 – VA • 2'h1 – VL0 • 2'h2 – VH0 • 2'h3 – VH1 マルチ CL 書き込みリクエストを形成する CL はすべて、同じ仮想チャネルでルーティングされます。
req_type	表 13 (24 ページ) で一覧になっているリクエストタイプです。
sop	マルチ CL メモリー書き込みのパケットの開始です。 • 1'h1 – 最初のヘッダーを示します。アドレスの昇順で書き込む必要があります。 • 1'h0 – 後続のヘッダーです。
cl_len	メモリーリクエストの長さです。 • 2'h0 – 64 バイト (1 CL) • 2'h1 – 128 バイト (2 CL) • 2'h3 – 256 バイト (4 CL) *注意:* mode = eMOD_BYTE の場合、cl_len は 2'h0 である必要があります。
address	64 バイトにアライメントされた物理アドレスです。つまり、byte_address>> 6 です。 アドレスは、cl_len フィールドに自然にアライメントする必要があります。例えば cl_len=2'b01 の場合、address[0] は 1'b0、同様に cl_len=2'b11 の場合、address[1:0] は 2'b00 でなければなりません。

表 13. AFU Tx リクエストのエンコーディングとチャネルマッピング

リクエストタイプ	エンコーディング	データペイロード	説明	ヘッダー・フォーマット
t_if_ccip_c0_tx: enum t_ccip_c0_req				
eREQ_RDLINE_I	4'h0	いいえ	キャッシュする意図のないメモリー読み出しリクエスト。	C0 メモリー・リクエスト・ヘッダー。表 14 (25 ページ) を参照ください。
eREQ_RDLINE_S	4'h1	いいえ	キャッシュヒントが Shared (共有) に設定されたメモリー読み出しリクエスト。	
t_if_ccip_c1_tx: enum t_ccip_c1_req				
eREQ_WRLINE_I	4'h0	はい	FPGA キャッシュにデータを保持する意図のないメモリー書き込みリクエスト。 キャッシュラインを FPGA キャッシュに保持しません。また、CPU 側のキャッシュについてのガイダンスも提供しません。 *注意:* CPU は CPU 側のキャッシュを担います。	C1 メモリー・リクエスト・ヘッダー。表 15 (25 ページ) を参照ください。
eREQ_WRLINE_M	4'h1	はい	キャッシュヒントが Modified (変更済み) に設定されたメモリー書き込みリクエスト。	
eREQ_WRPUSH_I	4'h2	はい	キャッシュヒントが Invalid (無効) に設定されたメモリー書き込みリクエスト。FIU は、FPGA キャッシュにデータを保持する意図なしに、プロセッサーのラスト・レベル・キャッシュ (LLC) にデータを書き込みます。書	

continued...

リクエストタイプ	エンコーディング	データペイロード	説明	ヘッダー・フォーマット
			き込み先の LLC は常に、**SDRAM** アドレスが属するプロセッサーに関連付けられた LLC です。 FPGA キャッシュにキャッシュラインを保持しませんが、そのラインを CPU LLC にプッシュします。	
eREQ_WRFENCE	4'h4	いいえ	メモリー書き込みフェンスリクエスト。	フェンスヘッダー。表 16 (26 ページ) を参照ください。
eREQ_INTR	4'h6	いいえ	割り込み	割り込みヘッダー。表 17 (26 ページ) を参照ください。
t_if_ccip_c2_tx – リクエスト・タイプ・フィールドはありません。				
MMIO Rd	該当なし	はい	MMIO 読み出し応答	MMIO 読み出し応答ヘッダー。 表 18 (26 ページ) を参照ください。

未使用のエンコーディングはすべて RSVD0 とみなされます。

表 14. C0 読み出しメモリー・リクエスト・ヘッダー・フォーマットの構造 (t_ccip_c0_ReqMemHdr)

ビット	ビット数	フィールド
[73:72]	2	vc_sel
[71:70]	2	RSVD
[69:68]	2	cl_len
[67:64]	4	req_type
[63:58]	6	RSVD
[57:16]	42	address
[15:0]	16	mdata

お使いのプラットフォームでバイト・イネーブルが利用可能かを特定するには、Verilog を使用して CCIP_ENCODING_HAS_BYTE_WR が定義されているか、およびパラメーター ccip_cfg_pkg::BYTE_EN_SUPPORTED が 0 以外であるかを確認する必要があります。バイト・イネーブルを利用するには、これらの 2 つの条件が満たされている必要があります。

注意: CCIP_ENCODING_HAS_BYTE_WR が定義されている場合、byte_start および byte_len が利用可能です。これは、バイト・イネーブルがお使いのプラットフォームで利用可能であることを意味するものではありません。

表 15. C1 書き込みメモリー・リクエスト・ヘッダー・フォーマットの構造 (t_ccip_c1_ReqMemHdr)

ビット	ビット数	Field SOP=1	Field SOP=0
[79:74]	6	byte_len (mode=eMOD_CL の場合は 0 でなければなりません) *注意:* このフィールドは、インテル FPGA PAC N3000 および インテル PAC (インテル Arria 10 GX FPGA 搭載版) に RSVD0 です。	byte_len (sop=0 の場合は 0 でなければなりません) *注意:* このフィールドは、インテル FPGA PAC N3000 および インテル PAC (インテル Arria 10 GX FPGA 搭載版) に RSVD0 です。
[73:72]	2	vc_sel	RSVD-DNC

continued...

ビット	ビット数	Field SOP=1	Field SOP=0
[71]	1	sop=1	sop=0
[70]	1	mode 注意: このフィールドは、インテル FPGA PAC N3000 および インテル PAC (インテル Arria 10 GX FPGA 搭載版) に RSVD0 です。	mode (sop=0 の場合、eMOD_CL である必要があります) 注意: このフィールドは、インテル FPGA PAC N3000 および インテル PAC (インテル Arria 10 GX FPGA 搭載版) に RSVD0 です。
[69:68]	2	cl_len	RSVD-DNC
[67:64]	4	req_type	req_type
[63:58]	6	byte_start (mode=eMOD_CL の場合は 0 でなければなりません) 注意: このフィールドは、インテル FPGA PAC N3000 および インテル PAC (インテル Arria 10 GX FPGA 搭載版) に RSVD0 です。	byte_start (sop=0 の場合は 0 でなければなりません) 注意: このフィールドは、インテル FPGA PAC N3000 および インテル PAC (インテル Arria 10 GX FPGA 搭載版) に RSVD0 です。
[57:18]	40	address[41:0]	RSVD-DNC
[17:16]	2		address[1:0]
[15:0]	16	mdata	RSVD-DNC

表 16. C1 フェンス・ヘッダー・フォーマットの構造 (t_ccip_c1_ReqFenceHdr)

ビット	ビット数	フィールド
[79:74]	6	RSVD
[73:72]	2	vc_sel
[71:68]	4	RSVD
[67:64]	4	req_type
[63:16]	48	RSVD
[15:0]	16	mdata

表 17. C1 割り込みヘッダー・フォーマットの構造 (t_ccip_c1_ReqIntrHdr) (インテル FPGA PAC のみ)

ビット	ビット数	フィールド
[79:74]	6	RSVD
[73:72]	2	vc_sel
[71:68]	4	RSVD
[67:64]	4	req_type
[63:12]	62	RSVD
[1:0]	2	id

表 18. C2 MMIO 応答ヘッダーのフォーマット

ビット	ビット数	フィールド
[8:0]	9	tid

1.3.8. CCI-P Rx 信号

図 -11: ccip_if_pkg.sv 内の Rx インターフェイス構造

```
typedef struct packed {
    logic               c0TxAlmFull;    //  C0 Request Channel Almost Full
    logic               c1TxAlmFull;    //  C1 Request Channel Almost Full

    t_if_ccip_c0_Rx     c0;
    t_if_ccip_c1_Rx     c1;
} t_if_ccip_Rx;
```

Rx チャネルは 2 つあります。

- チャネル 0 はメモリー応答、MMIO リクエストおよび UMsg をインターリーブします。
- チャネル 1 は、Tx チャネル 1 で開始された AFU リクエストに対する応答を返します。

`c0TxAlmFull` 信号および `c1TxAlmFull` 信号は、AFU への入力です。これらは Rx 信号の構造で宣言されますが、論理的には Tx インターフェイスに属します。そのため、前章において説明されています。

Rx チャネルにはフロー制御がありません。AFU は、生成したメモリーリクエストに対する応答を受け入れる必要があります。AFU は、メモリーリクエストを生成する前にバッファーを事前に割り当てる必要があります。AFU はまた、MMIO リクエストを受け入れる必要があります。

Rx チャネル 0 には、メモリー応答および MMIO リクエストに対する個別の Valid 信号があります。それらの Valid 信号の 1 つのみをサイクルに設定できます。MMIO リクエストには、MMIO 読み出しと MMIO 書き込みに個別の Valid 信号があります。`mmioRdValid` もしくは `mmioWrValid` のどちらかが設定されている場合、そのメッセージは MMIO リクエストであり、`t_if_ccip_c0_Rx.hdr` を `t_ccip_c0_ReqMmioHdr` にキャストし処理する必要があります。

表 19. Rx チャネルの信号の説明 (チャネル 0)

信号	幅 (ビット)	方向	説明
`pck_cp2af_sRx.c0.hdr`	28	入力	チャネル 0 の応答ヘッダーもしくは MMIO リクエストヘッダー。 表 21 (28 ページ) を参照してください。
`pck_cp2af_sRx.c0.data`	512	入力	チャネル 0 データバスのメモリー読み出し応答と UMsg。 • 64 バイトのデータを返します MMIO 書き込みリクエスト • 4 バイトの書き込みの場合、データはビット [31:0] で駆動されます。 • 8 バイトの書き込みの場合、データはビット [63:0] で駆動されます。
`pck_cp2af_sRx.c0.rspValid`	1	入力	1 に設定されている場合、チャネル 0 のヘッダーとデータが有効であることを示します。ヘッダーは、メモリー応答として解釈され、resp_type フィールドをデコードする必要があります。
`pck_cp2af_sRx.c0.mmioRdValid`	1	入力	1 に設定されている場合、チャネル 0 の MMIO 読み出しリクエストを示します。
`pck_cp2af_sRx.c0.mmioWrValid`	1	入力	1 に設定されている場合、チャネル 0 の MMIO 書き込みリクエストを示します。

表 20. Rx チャネルの信号の説明 (チャネル 1)

信号	幅 (ビット)	方向	説明
pck_cp2af_sRx.c1.hdr	28	入力	チャネル 1 の応答ヘッダー。表 21 (28 ページ) を参照ください。
pck_cp2af_sRx.c1.rspValid	1	入力	1 に設定されている場合、チャネル 1 のヘッダーは有効な応答であることを示します。

1.3.8.1. Rx ヘッダーおよび Rx データの形式

表 21. Rx ヘッダーのフィールドの定義

フィールド	説明
mdata	メタデータです。ユーザー定義のリクエスト ID で、メモリーリクエストから応答ヘッダーに変更されずに返されます。 マルチ CL のメモリー応答の場合、各 CL に同じ mdata が返されます。
vc_used	使用される仮想チャネルです。VA を使用する場合、このフィールドはリクエストに対して FIU が選択した仮想チャネルを識別します。そのほかの VC の場合はリクエスト VC を返します。
format	マルチ CL のメモリー書き込みリクエストを使用している場合、FIU はペイロード全体に対する単一の応答、もしくはペイロードの各 CL に対して応答を返す場合があります。 • 1'b0: アンパッキングされた書き込み応答で、各 CL に対して応答を返します。cl_num フィールドを検索し、キャッシュラインを識別します。 • 1'b1: パッキングされた書き込み応答で、単一の応答をペイロード全体に返します。cl_num フィールドは、1 CL、2 CL、または 4 CL のペイロードサイズを提供します。 *注意:* メモリー・プロパティー・ファクトリー (MPF) インテル FPGA ベーシック・ビルディング・ブロックからの書き込み応答は、AFU に送信される際にかならずパッキングされます。
cl_num	**Format=0:** 1 CL のデータペイロードを超える応答の場合、このフィールドは cl_num を識別します。 2'h0 – 最初の CL。最下位アドレス。 2'h1 – 2 番目の CL。 2'h2 – 3 番目の CL。 2'h3 – 4 番目の CL。最上位アドレス。 *注意:* 応答は順不同で返される場合があります。
	Format=1: このフィールドは、データのペイロードサイズを識別します。 2'h0 – 1 CL、または 64 バイト 2'h1 – 2 CL、または 128 バイト 2'h3 – 4 CL、または 256 バイト
hit_miss	キャッシュのヒットもしくはミスのステータスです。AFU はこれを使用し、さまざまなモジュールの詳細なヒットおよびミスの統計を生成できます。 1'b0 – キャッシュミス 1'b1 – キャッシュヒット
MMIO の長さ	MMIO リクエストの長さ 2'h0 – 4 バイト 2'h1 – 8 バイト 2'h2 - 64 バイト (MMIO 書き込みのみ)
MMIO アドレス	ダブルワード (DWORD) にアライメントされた MMIO アドレスオフセットです。つまり、byte address>>2 です。
UMsg ID	UMsg に対応する CL を識別します。
UMsg タイプ	2 つのタイプの UMsg がサポートされています。

continued...

フィールド	説明
	1'b1 – データなしの UMsgH (ヒント) 1'b0 – データ付きの UMsg

表 22. AFU Rx 応答のエンコーディングとチャネルマッピング

リクエストタイプ	エンコーディング	データペイロード	ヘッダー・フォーマット
t_if_ccip_c0_Rx: enum t_ccip_c0_rsp			
eRSP_RDLINE	4'h0	はい	メモリー応答ヘッダー。表 23 (29 ページ) を参照ください。 c0.rspValid で識別されます。
MMIO 読み出し	該当なし	いいえ	MMIO リクエストヘッダー。表 24 (29 ページ) を参照ください。
MMIO 書き込み	該当なし	はい	該当なし
eRSP_UMSG	4'h4	はい/いいえ	Umsg 応答ヘッダー。表 26 (30 ページ) を参照ください。c0.rspValid で識別されます。
t_if_ccip_c1_Rx: enum t_ccip_c1_rsp			
eRSP_WRLINE	4'h0	いいえ	メモリー応答ヘッダー。表 25 (30 ページ) を参照ください。 c1.rspValid で識別されます。
eRSP_WRFENCE	4'h4	いいえ	Wr フェンス応答ヘッダー。表 27 (30 ページ) を参照ください。
eRSP_INTR	4'h6	いいえ	割り込み応答ヘッダー。表 28 (30 ページ) を参照ください。

表 23. C0 メモリー読み出し応答ヘッダー・フォーマットの構造 (t_ccip_c0_RspMemHdr)

ビット	ビット数	フィールド
[27:26]	2	vc_used
[25]	1	RSVD
[24]	1	hit_miss
[23:22]	2	RSVD
[21:20]	2	cl_num
[19:16]	4	resp_type
[15:0]	16	mdata

表 24. MMIO リクエストヘッダーのフォーマット

ビット	ビット数	フィールド
[27:12]	16	address
[11:10]	2	length
[9]	1	RSVD
[8:0]	9	TID

表 25. C1 メモリー書き込み応答ヘッダー・フォーマットの構造 (t_ccip_c1_RspMemHdr)

ビット	ビット数	フィールド
[27:26]	2	vc_used
[25]	1	RSVD
[24]	1	hit_miss
[23]	1	format
[22]	1	RSVD
[21:20]	2	cl_num
[19:16]	4	resp_type
[15:0]	16	mdata

表 26. UMsg ヘッダーのフォーマット (FPGA 統合プラットフォームのみ)

ビット	ビット数	フィールド
[27:20]	8	RSVD
[19:16]	4	resp_type
[15]	1	UMsg タイプ
[14:3]	12	RSVD
[2:0]	3	UMsg ID

表 27. WrFence ヘッダー・フォーマットの構造 (t_ccip_c1_RspFenceHdr)

ビット	ビット数	フィールド
[27:20]	8	RSVD
[19:16]	4	resp_type
[15:0]	16	mdata

表 28. 割り込みヘッダー・フォーマットの構造 (t_ccip_c1_RspIntrHdr) (インテル FPGA PAC のみ)

ビット	ビット数	フィールド
[27:26]	2	vc_used
[25:20]	6	RSVD
[19:16]	4	resp_type
[15:2]	14	RSVD
[1:0]	2	id

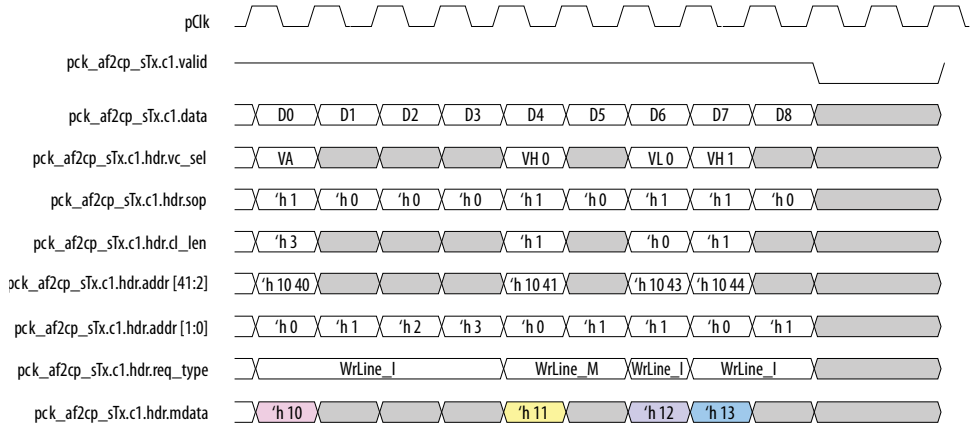
1.3.9. マルチキャッシュ・ライン・メモリー・リクエスト

最高のリンク効率を実現するには、メモリーリクエストを大きな転送サイズにパッキングします。これにはマルチ CL リクエストを使用します。以下に、マルチ CL メモリーリクエストの特性を示します。

- 4 CL のデータペイロードを使用する場合に、最高のメモリー帯域幅が実現されます。
- メモリー書き込みリクエストは、かならず最下位アドレスから開始する必要があります。c1_ReqMemHdr の SOP=1 は、最初の CL を示します。マルチ CL リクエストの後続のヘッダーはすべて、Address[1:0] の増分値を駆動する必要があり、Address[41:2] はドントケアとして扱われます。
- N CL メモリー書き込みリクエストは、チャネル 1 で N サイクルかかります。マルチ CL リクエストの途中でアイドルサイクルを持つことは正当ですが、1 つのリクエストを別のリクエストとインターリーブすることはできません。マルチ CL 書き込みリクエストのデータペイロード全体を完了せずに新しいリクエストを開始することは不正です。
- FIU は、マルチ CL の VA リクエストを単一の VC で完了することを保証します。
- メモリー・リクエスト・アドレスは自然にアライメントされている必要があります。2CL リクエストは 2-CL 境界で開始し、その CL アドレスは 2 で割り切れなければなりません。すなわち、address[0] = 1'b0 です。4CL リクエストは 4-CL 境界でアライメントし、その CL アドレスは 4 で割り切れなければなりません。すなわち、address[1:0] = 2'b00 です。
- マルチ CL バーストは、他のリクエストを発行する前にすべてのワードを送信し、完了している必要があります。これは、単一のマルチ CL バースト内では次の特別なメモリー書き込みリクエストをインターリーブできないことを意味します。
  - 書き込みフェンス
  - 割り込み
  - バイト・イネーブル書き込み

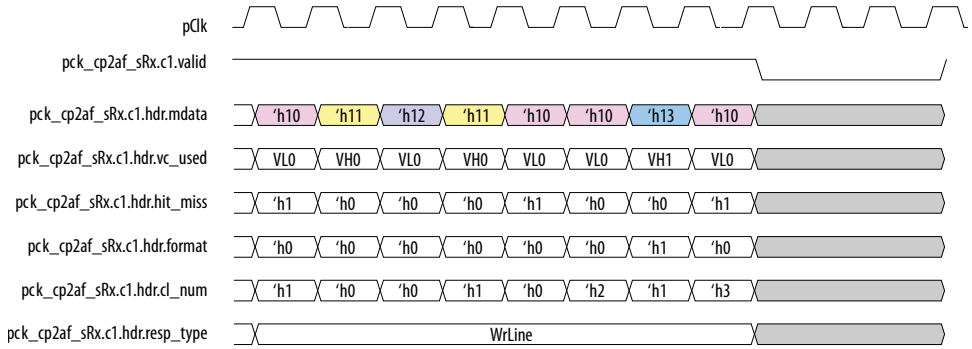
次の図は、マルチ CL メモリー書き込みリクエストの例です。

図 -12: マルチ CL メモリーリクエスト



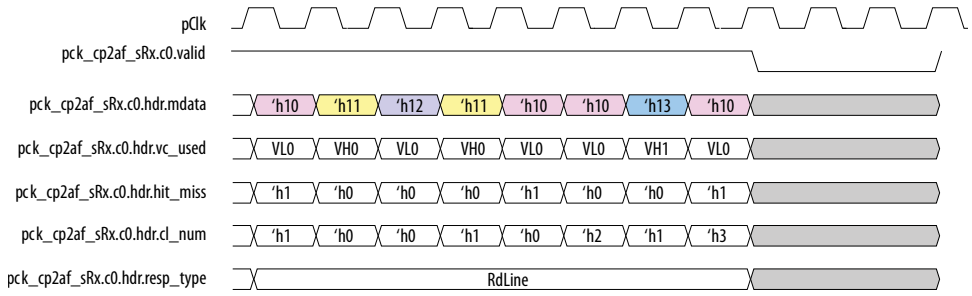
次の図は、メモリー書き込み応答サイクルの例です。アンパッキングされた応答の場合、各 CL は順不同で返される場合があります。

図 -13: マルチ CL メモリー書き込み応答



以下は、メモリー読み出し応答サイクルの例です。読み出し応答は、それ自体の中で並べ替えることができます。すなわち、マルチ CL 読み出しの各 CL の順序は保証されていません。マルチ CL 応答内の CL はすべて、同じ mdata と同じ vc_used を持ちます。マルチ CL 読み出しのそれぞれの CL は、cl_num フィールドを使用して識別されます。

図 -14: マルチ CL メモリー読み出し応答



1.3.10. バイト・イネーブル・メモリー・リクエスト (インテル FPGA PAC D5005)

書き込みデータを詳細に制御し、データの特定のバイトのみをホストメモリーに書き込むには、バイト・イネーブル・モードを使用します。以下に、バイト・イネーブル・メモリー・リクエストの特性を示します。

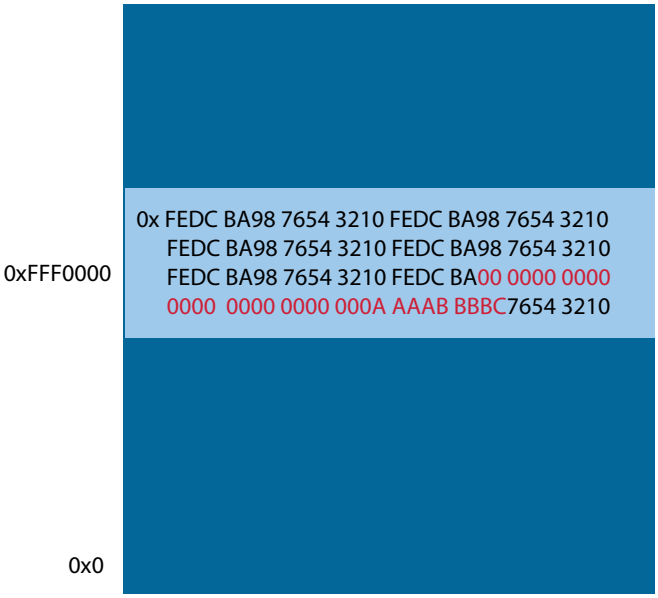
- バイト・イネーブル・メモリー・リクエストは、バイト不変のリトル・エンディアン・スキームを使用します。すなわち、これは次のことを意味します。
  - データ構造内の単一または複数バイトの要素の場合、その要素はメモリーの同じ連続バイトを使用します。これは、byte_start および byte_len ヘッダーフィールドで指定されます。
  - 書き込みデータは、キャッシュライン内の格納されるオフセットのデータフィールドに配置されます。最初に書き込まれるデータビットは、ビット byte_start*8 です。
- byte_start はバイト・インデックスを指定します。最下位バイトは 0 (pck_af2cp_sTx.c1.data[7:0]) であり、最上位バイトは 63 (pck_af2cp_sTx.c1.data[511:504]) です。
- byte_len は、バイト・イネーブルのメモリー書き込みトランザクションに含まれるバイト数を指定します。バイト長は、書き込みデータを最上位バイトに向かって拡張します。
- バイト・イネーブル・メモリー・リクエストは、キャッシュ長 (cl_len) を 0 (1 CL メモリー書き込みリクエスト) に設定して行う必要があります。
- 長さは、pck_af2cp_sTx.c1.data のバイト 63 を超えることができません。許容される最大のバイト長は、次の計算式で表すことができます。
  - byte_start が 0 の場合
    - MAX_BYTE_LEN = 63
  - byte_start が 0 ではない場合
    - MAX_BYTE_LEN = 64 – byte_start

次の表は、チャネル 1 のリクエストヘッダー (pck_af2cp_sTx.c1.hdr) が、バイト・イネーブル・モードにおいてバイトをインデックス付けする方法の例を示しています。この例において AFU デザイナーは、データワード 0xAAAABBBBCCCCDDDDE のバイト [20:4] (pck_af2cp_sTx.c1.data[167:32]) を、アドレス 0xFFF00 のホストメモリーのバイト [20:4] に書き込みます。

	ビット・インデックス	幅 (ビット)	フィールド	値
ヘッダー	79:74	6	byte_len	0x11
	73:72	2	vc_sel	eVC_VA (0x0)
	71	1	sop=1	1
	70	1	(CL/バイト) 0:CL、1:バイト	1
	69:68	2	cl_len	0
	67:64	4	req_type	eREQ_WRLINE_I (0x0)
	63:58	6	byte_start	0x4
	57:18	40	address	0xFFF00
	17:16	2	—	0x0
	15:0	16	mdata	0x0
データ	—	512	データ	0xAAAABBBBCCCCDDDDE

図 -15: アドレス 0xFFF00 のホストメモリー

この図は、書き込み前後のホストメモリーの値を示しています。

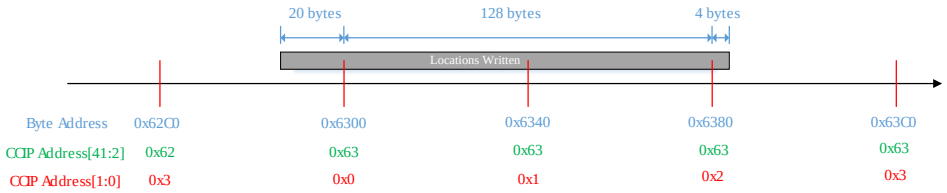


1.3.10.1. バイト・イネーブルとフル・キャッシュ・ライン・アクセスの併用

一部のアプリケーションでは、AFU は、ホストメモリーの 64 バイト境界に始まりがアライメントされていないバッファー、または次の 64 バイト境界よりも前に終わるバッファーにアクセスすることが必要になります。AFU は、バイト・イネーブルのトランザクションおよびフル・キャッシュライン・アクセスを併用し、任意の境界で開始または終了するバッファー書き込みを実行することができます。そのような転送の場合に AFU は、バイト・イネーブルのバースト (mode=eMOD_BYTE) とフル・キャッシュ・ラインのバースト (mode=eMOD_CL) を混在させることはできません。

次の例において AFU は、バイトアドレス 0x62EC で始まるインクリメント・パターンの 152 バイトを書き込みます。アクセスされる最初と最後のバイトアドレスは 64 バイト境界に揃っていないため、転送は 3 つのセクションに分割されます。開始セクションと終了セクションはバイト・イネーブルを使用して、メモリーの 64 バイトにアライメントされた領域内のバイトのサブセットを更新します。最初のセクションは 20 バイトをメモリーに書き込み、その後 128 バイトがフル・キャッシュ・ライン・バーストを使用して書き込まれます。そして最後の 4 バイトの書き込みが続きます。

図 -16: アクセスされるメモリーと対応する CCIP アドレス

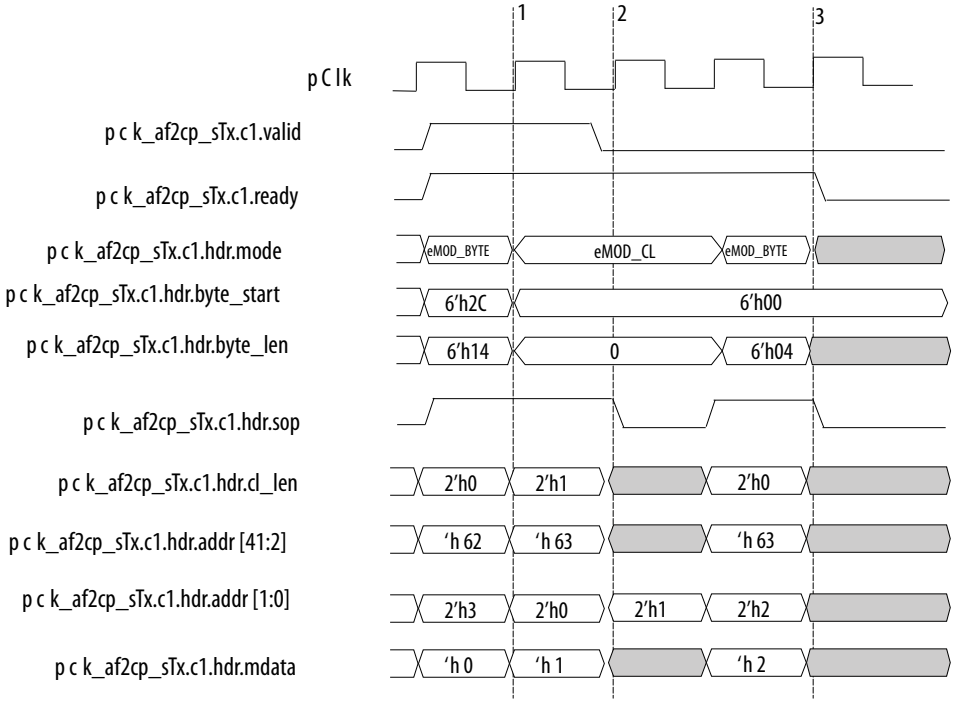


最初のアクセスが 64 バイト境界で開始しないため、モードは eMOD_BYTE に設定されます。byte_start フィールドは 0x2C、byte_len フィールドは 0x14、そして CCIP アドレスビット 41:2 は 0x62 に設定され、CCIP アドレスビット 1:0 は 0x3 に設定されます。

2 番目のアクセスは 2CL の境界にアライメントされているため、次の 128 バイトは、モードが eMOD_CL に設定された 2 ビートのバーストとしてポストすることができます。2 つのモードを同じバーストでインターリーブすることはできないため、このアクセスは、モードを eMOD_BYTE に設定するビートと組み合わせることはできません。

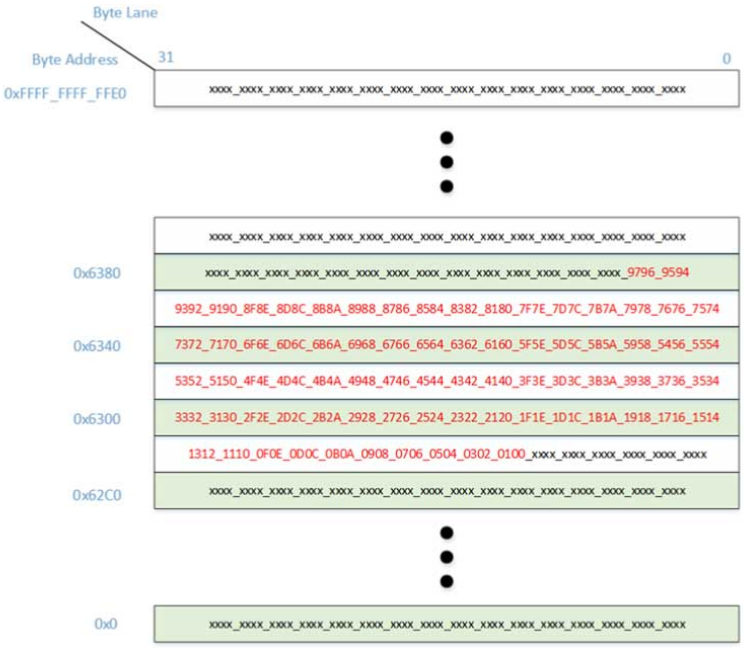
3 番目のアクセスは 64 バイト境界で開始しますが、メモリーの 4 バイトにしかアクセスしません。そのため、モードは eMOD_BYTE に設定されます。byte_start フィールドは 0x0、byte_len フィールドは 0x4、そして CCIP アドレスビット 41:2 は 0x63 に設定され、ビット 1:0 は 0x2 に設定されます。

図 -17: バイト・イネーブルとフル・キャッシュ・ラインを併用するアクセスのタイミング図



注記:
1. バイト・イネーブル (mode = eMOD_BYTE) を使用して開始バイトアドレス0x62ECにポストされる20バイトアクセス
2. 2CLバースト (mode = eMOD_CL) として開始バイトアドレス0x6300にポストされる128バイトアクセス
3. バイト・イネーブル (mode = eMOD_BYTE) を使用して開始バイトアドレス0x6380にポストされる4バイトアクセス

図 -18: ホストメモリー



1.3.11. そのほかの制御信号

特に明記されない限り、信号はすべてアクティブ High です。

表 29. クロックとリセット

信号	幅 (ビット)	方向	説明
pck_cp2af_softReset	1	入力	アクティブ HIGH の同期ソフトリセット。 1 に設定された場合、AFU はすべてのロジックをリセットする必要があります。最小リセットパルス幅は 256 pClk サイクルです。未処理の CCI-P リクエストはすべて、ソフトリセットをディアサートする前にフラッシュされます。 ソフトリセットは FIU をリセットしません。
pClk	1	入力	一次インターフェイス・クロック。CCI-P インターフェイス信号はすべて、このクロックに同期しています。
pClkDiv2	1	入力	pClk に同期し位相しています。0.5x の pClk クロック周波数です。
pClkDiv4	1	入力	pClk に同期し位相しています。0.25x の pClk クロック周波数です。
uClk_usr	1	入力	ユーザー定義のクロックは、pClk と同期していません。 AFU は CCI-P インターフェイスを駆動する前に、この信号を pClk ドメインに同期する必要があります。 AFU のロード・ユーティリティーは、pck_cp2af_softReset をディアサートする前にユーザー定義のクロック周波数をプログラムします。
uClk_usrDiv2	1	入力	uClk_usr と同期し、0.5x の周波数です。

continued...

信号	幅 (ビット)	方向	説明
			注意: 周波数は、uClk_usr と同期しない値に設定することが可能です。
pck_cp2af_pwrState	2	入力	現在の AFU の消費電力状態に対するリクエストを示します。これに対応し、AFU は消費電力の削減を試みる必要があります。十分な消費電力の削減が達成されない場合、AFU がリセットになる場合があります。 2'h0 – AP0 - 通常の動作モード 2'h1 – AP1 - 50%の消費電力削減リクエスト 2'h2 – 予約済み 2'h3 – AP2 - 90%の消費電力削減リクエスト pck_cp2af_pwrState が AP1 に設定されると、FIU は 50%のスループットの削減を実現するためにメモリー・リクエスト・パスの抑制を開始します。AFU もまた、FPGA 内部メモリーリソースおよび計算エンジンへのアクセスを抑制することで、電力使用率を 50%まで低減することが期待されます。同様に、AP2 に移行すると、FIU はメモリー・リクエスト・パスを抑制し、通常状態に対して 90%のスループットを削減します。AFU もまた、電力使用率を 90%まで低減することが期待されます。
pck_cp2af_error	1	入力	CCI-P プロトコルエラーが検出され、PORT Error レジスターに記録されています。このレジスターは、AFU に対して可視化されています。 これは、信号タップのトリガー条件として使用できます。 このようなエラーが検出されると、CCI-P インターフェイスは新しいリクエストの受け入れを停止し、AlmFull を 1 に設定します。 CCI-P プロトコルエラーが発生した際は、AFU がまだアクティブな (リセット状態になっていない) 場合でも、未処理のトランザクションが完了すると考えないでください。

関連情報

Accelerator Functional Unit (AFU) Developer's Guide for Intel FPGA Programmable Acceleration Card

1.3.12. プロトコルフロー

1.3.12.1. アップストリーム・リクエスト

表 30. AFU から FIU へのアップストリーム・リクエストのプロトコルフロー

Tx データの欄は、リクエストが Tx データペイロードを予期しているかを示します。Rx データの欄は、応答が Rx データペイロードを返すかどうかを示します。

タイプ	Tx リクエスト	Tx データ	Rx 応答	Rx データ
メモリー書き込み	WrLine_I	はい	WrLine	いいえ
	WrLine_M			
	WrPush_I			
メモリー読み出し	RdLine_I	いいえ	RdLine	はい
	RdLine_S			
特別なメッセージ	WrFence	いいえ	WrFence	いいえ
	Interrupt	いいえ	Interrupt	いいえ

表 31. AFU から FIU へのアップストリーム・リクエストのプロトコルフロー

- WrLine_I: CL にまず書き込み、それをその後キャッシュからエビクションする必要があるため、特別な処理を必要とします。エビクションはリクエストのフェーズ 2 を形成します。
- RdLine_I: デフォルトの読み出しタイプとして推奨されます。
- RdLine_S: 多くリファレンスされている CL を特定した場合にのみ、控えめに使用してください。
- RdCode: CPU ディレクトリーを更新し、FPGA に共有状態でラインをキャッシュさせます。RdCur は CPU ディレクトリーを更新せず、FPGA はこのラインをキャッシュしません。CPU からこのラインへのその後のアクセスでは、FPGA をスヌーピングしません。

CCI-P リクエスト	FPGA キャッシュ		UPI サイクル	次の状態	CCI-P 応答	UPI サイクル	次の状態	CCI-P 応答	UPI サイクル	次の状態
	ヒット/ミス	状態	フェーズ 1			フェーズ 2			フェーズ 3	
WrLine_I	ヒット	M	なし	M	WrLine	WbMtoI	I			
	ヒット	S	InvItoE							
	ミス	I								
WrLine_M	ヒット	M	なし	M	WrLine	該当なし				
	ヒット	S	InvtoE							
	ミス	I								
WrLine_I	ミス	M	WbMotI	I		InvItoE	M	WrLine	WbMotI	I
WrLine_M										
WrPush_I									WbPushMotI	I
WrLine_I	ミス	S	EvctCln	I		InvItoE	M	WrLine	WbMotI	I
WrLine_M										
WrPush_I									WbPushMotI	I
WrPush_I	ヒット	M	なし	M	WrLine	WbPushMotI	I			
		S、I	InvItoE							
RdLine_S	ヒット	S、M	なし	変更なし	RdLine	該当なし				
	ミス	I	RdCode	S	RdLine					
RdLine_I	ヒット	S、M	なし	変更なし	RdLine	該当なし				
	ミス	I	RdCur	I	RdLine					
RdLine_I	ミス	M	WbMtoI	I		RdCur	I	RdLine		
RdLine_S						RdCode	S			
RdLine_I		S	EvctCln			RdCur	I			
RdLine_S						RdCode	S			

1.3.12.2. ダウンストリーム・リクエスト

表 32. CPU から AFU へのダウンストリーム・リクエストのプロトコルフロー

Rx リクエスト	Rx データ	Tx 応答	Tx データ
MMIO 読み出し	いいえ	MMIO 読み出しデータ	はい
MMIO 書き込み	はい	なし	該当なし
UMsg	はい	なし	該当なし
UMsgH	いいえ	なし	該当なし

1.3.13. 順序付けの規則

1.3.13.1. メモリーリクエスト

CCI-P メモリーの一貫性モデルは、PCIe の一貫性モデルとは異なります。CCI-P は、「緩和された」メモリーの一貫性モデルを実装します。

以下に対するリクエストの順序付けの要件が緩和されます。

- 同じアドレス
- 異なるアドレス

表 33 (39 ページ) は、CCI-P 上の 2 つのメモリーリクエスト間の順序付けの関係を定義しています。同じ規則が、同じアドレスもしくは異なるアドレスへのリクエストに適用されます。下記表の内容は次のように定義されています。

- はい: 最初の列のリクエストは、最初の行のリクエストを渡すことができます。
- いいえ: 最初の列のリクエストは、最初の行のリクエストを渡すことができません。

表 33. AFU からのアップストリーム・リクエストにおける順序付けの規則

行は列をバイパスするか	読み出し	書き込み	WrFence	割り込み
読み出し	はい	はい	はい	はい
書き込み	はい	はい	いいえ	はい
WrFence	はい	いいえ	いいえ	いいえ
割り込み	はい	はい	いいえ	はい

上記表は、次のように解釈できます。

- WrFences に関しては、読み出しを除いてすべての動作が順序付けられます。
- そのほかの動作はすべて、順序付けられません。

VC 内における書き込みの可観測性

メモリー書き込み応答を受信すると、書き込みはローカルの可観測点に達します。

注意: VA は物理チャネルではないため、VA に対するリクエストにそのような保証はありません。

- AFU から同じ物理チャネルへのその後の読み出しはすべて、新しいデータを受信します。
- 同じ物理チャネルでのその後の書き込みはすべて、データを置き換えます。

VC 間における書き込みの可観測性

メモリー書き込み応答は、そのデータがすべてのチャネルにわたりグローバルに可観測であることを意味するものではありません。別のチャネルでの後続の読み出しが古いデータを返し、別のチャネルでの後続の書き込みが元の書き込みよりも先に退く場合があります。VA への WrFence は、VC 間で同期することが保証されているプロトコルを呼び出します。WrFence VA は、すべてのチャネルでブロードキャスト動作を行います。

- 書き込みフェンスに先行する書き込みはすべて、グローバルな可観測点にプッシュされます。
- WrFence 応答を受信すると、AFU からのその後の読み出しはすべて、書き込みフェンスが発行されるまでに書き込まれたデータの最新のコピーを受信します。

1.3.13.1.1. メモリー書き込みフェンス

CCI-P WrFence リクエスト

CCI-P は `WrFence` のリクエストタイプを定義します。これは、VA を含むすべての VC に使用可能です。FIU の `WrFence` の実装は C1 チャネルをストールするため、CCI-P 書き込みパスを共有している書き込みストリームをすべてブロックします。さらに、`WrFence` リクエストはグローバルな可観測性を保証するため、FIU は PCIe パスに対して長さゼロの読み出し (ZLR) を生成し、書き込みをプッシュします。そのため、`WrFence` リクエストは C1 チャネルで長いストールを引き起こす可能性があります。これを回避するには、AFU のデータフローの同期点にその使用を制限します。

- WrFence は、書き込みフェンスに続く書き込みが処理される前に、そのフェンスに先行するすべての割り込みまたは書き込みがメモリーに確定されることを保証します。
- WrFence の順序は、ほかのメモリー書き込み、割り込み、または WrFence リクエストと入れ替わりません。
- WrFence は、読み出しリクエストに関して順序付けの保証はしません。
- WrFence はそれに続く読み出しをブロックしません。すなわち、メモリー読み出しは WrFence をバイパスできます。この規則は、「メモリーリクエスト」の章で説明されています。
- WrFence リクエストには vc_sel フィールドがあります。これにより、WrFence が適用される仮想チャネルを決定できます。例えば、VL0 を使用してデータブロックを移動する場合は、VL0 上の書き込みリクエストのみをシリアル化します。つまり、WrFence を VL0 で使用する必要があります。同様に、VA でのメモリー書き込みを使用する場合は、WrFence を VA で使用します。
- WrFence リクエストは応答を返します。応答は RX C1 を介して AFU に伝達され、resp_type フィールドで識別されます。読み出しは WrFence をバイパスできるため、書き込みに続いて読み出しを行い、最新のデータが読み出されるようにするには (RaW ハザード)、WrFence を発行し、同じ位置に読み出しを発行する前に WrFence の応答を待ちます。

書き込み応答のカウント

AFU はメモリー書き込みバリアーを実装します。これは、未処理の書き込みがすべて完了するまで待機し、その後、バリアーの後の次の書き込みを送信することで可能になります。未処理の書き込みを追跡するロジックは、リクエスト時に増加し応答時に減少する単純なカウンターにすることができます。よって、「書き込み応答のカウント」というタイトルが付けられています。書き込み応答は、ローカルの可観測性のみを保証します。この手法は、単一の VC (VL0、VH0、VH1) を対象とする書き込みストリームにメモリーバリアーを実装する場合にのみ機能します。書き込みストリームが VA、もしくはいくつかの VC の組み合わせを使用している場合、この手法は使用しないでください。

注意: そのような場合は、代わりに書き込みフェンスを実装します。

この手法の主な利点の 1 つは、AFU が詳細なバリアーを実装できることです。例えば、AFU に 2 つの独立した書き込みストリームがある場合、各ストリームに書き込み応答トラッカーを実装することができます。書き込みストリーム 1 にメモリーバリアーが必要な場合、ストリーム 1 からの書き込みのみがストールし、ストリーム 2 からの書き込みの送信は継続します。Mdata フィールドは、ストリーム ID のエンコーディングに使用できます。このような詳細なメモリーバリアーにより、以下が可能になります。

- すべての書き込みではなく、特定の未処理の書き込みの完了のみを待機するため、バリアーのレイテンシーの影響を最小限に抑えます。
- 無関係な書き込みストリームを継続して進行させることができるため、リンクの使用率が向上します。

関連情報

メモリーリクエスト (39 ページ)

1.3.13.1.2. メモリーの一貫性の説明

CCI-P は、同じアドレスおよび異なるアドレスへのリクエストの順序を変更することができます。同じアドレスに対するリクエストのデータハザードを特定するロジックは実装していません。

同じ VC への 2 つの書き込み

メモリーでは、最初の書き込み応答が受信された後に 2 つ目の書き込みリクエストが生成されない限り、同じ VC に対する 2 つの書き込みは、それらの実行順序とは異なる場合があります。これは一般的に、ライトアフターライト (WaW) ハザードとして知られています。

次の表は、最初の書き込みが受信された後に 2 番目の書き込みが実行される場合の、同じ VC への 2 つの書き込みを示しています。

表 34. 同じ VC への 2 つの書き込み (1 つのみ未処理)

AFU	プロセッサー
VH1: Write1 Addr=X, Data=A Resp 1 VH1: Write2 Addr=X, Data=B Resp 2	—
—	Read1 Addr=X, Data = A Read2 Addr=X, Data = B

AFU は同じ VC でアドレス X へ 2 回書き込みますが、最初の書き込みが受信されてからのみ 2 番目の書き込みを送信します。これにより、最初の書き込みがリンクで送信された後に次の書き込みが送信されるようになります。CCI-P は、プロセッサーがこれらの書き込みを発行された順序で認識するよう保証します。プロセッサーはアドレス X を複数回読み出す際に、データ A を認識し、その後データ B を認識します。

同じ VC への書き込み間に順序付けを強制するには、WrFence を代わりに使用します。WrFence のセマンティクスはより強力で、フェンス前の書き込みがすべて完了するまでフェンス後の書き込み処理すべてをストールすることに注意してください。

異なる VC への 2 つの書き込み

次の表は、異なる VC への 2 つの書き込みが、発行された順序とは異なる順序でメモリーに確定される場合があることを示しています。

表 35. 異なる順序での書き込みの確定

AFU	プロセッサー
VH1: Write1 X, Data=A VL0: Write2 X, Data=B	—
—	Read1 X, Data = B Read2 X, Data = A

AFU は X に対して 2 回の書き込みを行います (VH1 を介して Data=A、VL0 を介して Data=B)。プロセッサーはアドレス X でポーリングし、X への更新を逆の順序で認識する場合があります (CPU は Data=B の後に Data=A を認識する)。すなわち、プロセッサーが認識する書き込み順序は、AFU が書き込みを完了した順序とは異なる場合があります。異なるチャネルへの書き込みには順序付けの規則がないため、書き込みフェンスを VA にブロードキャストし、それらを同期させる必要があります。

次の表は、WrFence を使用し書き込み順序を強制する方法を示しています。

表 36. 書き込み順序強制のための WrFence の使用

AFU	プロセッサー
VH1: Write1 Addr=X, Data=A VA: WrFence VL0: Write2 Addr=X, Data=B	—
—	Read1 Addr=X, Data = A Read2 Addr=X, Data = B

この場合、AFU は VA WrFence を 2 つの書き込み間に追加しています。WrFence は、プロセッサーに WrFence 前の書き込みを可視化した後に、WrFence 後の書き込みを可視化します。よって、プロセッサーは Data=A を認識し、その後 Data=B を認識します。シリアル化される書き込みが異なる VC で送信されているため、WrFence は VA に発行されます。

異なる VC からの 2 つの読み出し

異なる VC へ読み出しを発行すると、順不同で完了する場合があり、最後の読み出し応答が古いデータを返す場合があります。

次の表に、異なる VC を介した同じアドレスからの読み出しが順不同になる場合があることを示します。

表 37. 同じアドレスに対する読み出し順序の変動 (異なる VC を介して)

プロセッサー	AFU	
Store addr=X, Data=A Store addr=X, Data=B	Request	Response
	VH1: Read1 Addr=X	—
	VL0: Read2 Addr=X	—
	—	VL0: Resp2 Addr=X, Data=B
	—	VH1: Resp1 Addr=X, Data=A

プロセッサーは X=1 を書き込み後に X=2 を書き込みます。AFU は、異なる VC を介して X を 2 回読み出します。Read1 は VH1 で送信され、Read2 は VL0 で送信されます。FIU は応答の順序を変更し、データを異なる順序で返す場合があります。AFU は X=2 を認識し、その後 X=1 を認識する場合があり、これはプロセッサーの書き込み順序とは異なります。

同じ VC からの 2 つの読み出し

同じ VC への読み出しは、順序が変わって完了する場合がありますが、最後の読み出し応答はかならず最新のデータを返します。次の表で示すように、最後の読み出し応答は古い読み出しリクエストに対応する場合があります。

注意: VA 読み出しは、異なる VC からの 2 つの読み出しのように動作します。

次の表に、同じ VC を介した同じアドレスからの読み出しの順序が入れ替わる場合があることを示しています。ただし、AFU は書き込まれた順序と同じ順序で更新を認識します。

表 38. 同じアドレスに対する読み出し順序の変動 (同じ VC を介して)

プロセッサー	AFU	
`Store Addr=X, Data=A` `Store Addr=X, Data=B`	`Request`	`Response`
	`VL0: Read1 Addr=X`	—
	`VL0: Read2 Addr=X`	—
	—	`VL0: Resp2 Addr=X, Data=A`
	—	`VL0: Resp1 Addr=X, Data=B`

プロセッサーは X=1 の後に X=2 を書き込みます。AFU は同じ VC を介してアドレス X を 2 回読み出します。Read1 および Read2 はどちらも VL0 に送信されます。FIU が読み出し応答順序を変更する場合がありますが、CCI-P 標準は最新のデータを最後に返すことを保証しています。つまり、AFU はアドレス X への更新を、プロセッサーが書き込んだ順序で認識します。

VA を使用する場合、FIU は順不同でデータを返す場合があります。これは、VA リクエストは VL0、VH0 または VH1 に向けられる場合があるためです。

同じ VC からのリードアフターライト

CCI-P 標準は、同じアドレスへの読み出しおよび書き込みリクエストの場合でも、それらのリクエストの順序付けを行いません。AFU は明示的にこの依存関係を解決する必要があります。

異なる VC からのリードアフターライト

AFU は異なる VC が使用される場合において、リードアフターライトの依存関係を解決することはできません。

同じ VC または異なる VC に対するライトアフターリード

CCI-P は同じアドレスに対するリクエストの場合でも、読み出し後の書き込みの順序付けを行いません。AFU は明示的にこの依存関係を解決する必要があります。AFU は、読み出し応答が受信された後にのみ書き込みリクエストを送信する必要があります。

トランザクションの順序付けのシナリオ例

同じアドレスへのトランザクション—同じアドレスへの複数の未処理の読み出しまたは書き込みリクエストは、非決定的な動作になります。

- **例 1:** 同じアドレス X への 2 つの書き込みは、順不同で完了する場合があります。アドレス X の最終的な値は非決定的です。順序付けを強制するには、書き込みリクエストの間に WrFence を追加します。または、同じ仮想チャネルにアクセスする場合は、最初の書き込みからの応答が返されるまで待機し、その後 2 番目の書き込みを発行します。
- **例 2:** 同じアドレス X からの 2 つの読み出しは、順不同で完了する場合があります。これはデータハザードではありませんが、AFU デベロッパーは順序付けに関する前提を立てないでください。読み出しがどちらも同じ仮想チャネルに発行されている場合、受信する 2 番目の読み出し応答はアドレス X に保存されている最新のデータを含みます。
- **例 3:** アドレス X に対する書き込み後のアドレス X からの読み出しは非決定的です。すなわち、読み出しはアドレス X の新しいデータ (書き込み後のデータ) または古いデータ (書き込み前のデータ) を返します。確実に最新データの読み出しを行うには、同じ仮想チャネルを使用し、アドレス X への読み出しを発行する前に書き込み応答が返されるのを待機します。
- **例 4:** アドレス X に対する読み出し後の書き込みは非決定的です。すなわち、読み出しはアドレス X の新しいデータ (書き込み後のデータ) または古いデータ (書き込み前のデータ) を返します。

  読み出し応答を使用し、読み出しの依存関係を解決します。

異なるアドレスへのトランザクション—異なるアドレスへの読み出しまたは書き込みリクエストは、順不同で完了する場合があります。

- **例 1:** AFU がアドレス Z へデータを書き込み、その後アドレス X のフラグの値を更新することで SW スレッドに通知を行うとします。

  これを実装するには、AFU は Z への書き込みと X への書き込みの間に書き込みフェンスを使用する必要があります。書き込みフェンスは、X への書き込みが処理される前に Z をグローバルに可視化します。
- **例 2:** AFU がアドレス Z から始まるデータを読み出し、その後アドレス X のフラグの値を更新することでソフトウェア・スレッドに通知を行うとします。

  これを実装するには、AFU は Z からの読み出しを実行し、読み出し応答をすべて待機した後に X への書き込みを実行する必要があります。

1.3.13.2. MMIO リクエスト

FIU は、AFU の MMIO アドレス空間を 64 ビットのプリフェッチ可能な PCIe BAR にマッピングします。AFU の MMIO がマッピングされたレジスターに、読み出しの副作用はありません。これらのレジスターへの書き込みは、書き込みマージを許容します。

プリフェッチ可能な BAR に関する詳細は、*PCIe の仕様*を参照ください。

AFU をターゲットとする MMIO リクエストは、PCIe リンクから受信した順序のとおりに AFU に送信されます。同様に、MMIO 読み出し応答は、AFU が CCI-P インターフェイスに送信した順序と同じ順序で PCIe リンクに返されます。すなわち FIU は、AFU をターゲットとする MMIO リクエストまたは応答順序の変更を行いません。

IA プロセッサーは、PCIe BAR を UC もしくは WC のメモリータイプとしてマッピングすることができます。表 39 (45 ページ) に、UC および WC タイプの BAR に対する IA の順序付けの規則が説明されています。

UC (キャッシュ不可) および WC (ライトコンバイン) の順序付けの規則に関する詳細は、*Intel Software Developers Manual* を参照ください。

表 39. MMIO の順序付けの規則

リクエスト	メモリー属性	ペイロードサイズ	メモリーの順序	備考
MMIO 書き込み	UC	4 バイト、8 バイト、または 64 バイト	強く順序付けられます	一般的なケース (ソフトウェアの動作)
	WC	4 バイト、8 バイト、または 64 バイト (インテル® Advanced Vector Extensions 512 (インテル® AVX-512) が必要です)	弱い順序付けです	特別なケース
MMIO 読み出し	UC	4 バイトまたは 8 バイト	強く順序付けられます	一般的なケース (ソフトウェアの動作)
	WC	4 バイトまたは 8 バイト	弱い順序付けです	特別なケース。ストリーミング読み出し (MOVNTDQA) は、より広い読み出しを引き起こす可能性があります。サポートされていません。

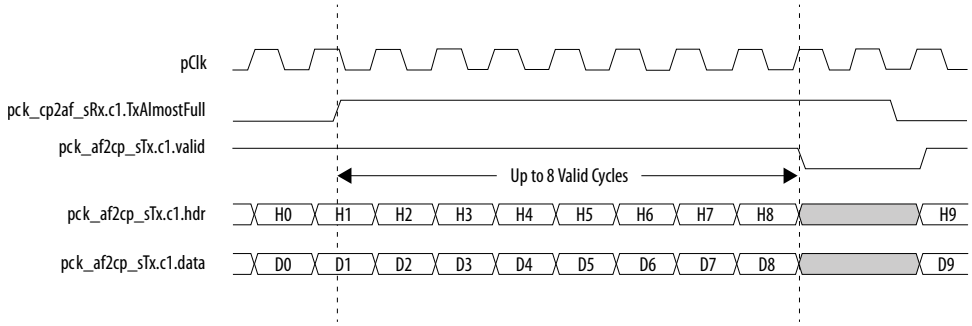
関連情報

Intel Software Developers Manual

1.3.14. タイミング図

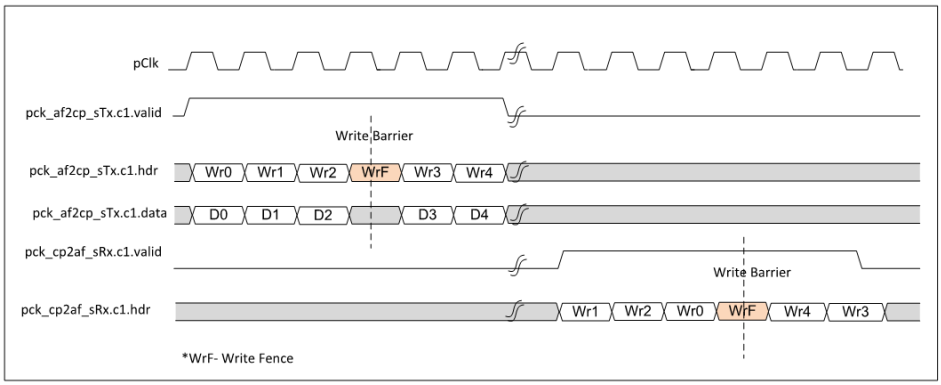
この章では、CCI-P インターフェイス信号のタイミング図を提供します。

図 -19: Tx チャネル 0 および 1 がフルのしきい値に近い状態



注意: Tx チャネル 0 およびチャネル 1 がフルのしきい値に近いことを示す信号は、許容できるトランザクションがあと 8 つしかない場合にアサートされます。TX チャネル 0 およびチャネル 1 は、フルに近いことを示す信号がアサートされた後に有効な信号を 8 サイクルまでディアサートする必要があります。

図 -20: 書き込みフェンスの動作



WrFence は、WrLine リクエストの間に挿入されます。WrFence 応答は、Rx チャネル 1 から返されます。

注意: 図 20 (46 ページ) では、WrFence 前に生成された書き込みはすべて、WrFence 後に到着した書き込みが完了する前に応答 (完了) します。

WrFence は、選択された VC に対してそれまでに発行された書き込みのみをフェンスします。すべての VC にわたって書き込みをフェンスするには、VA を選択します。

図 -21: MMIO リクエストとメモリー応答間でインターリーブされる C0 Rx チャネル

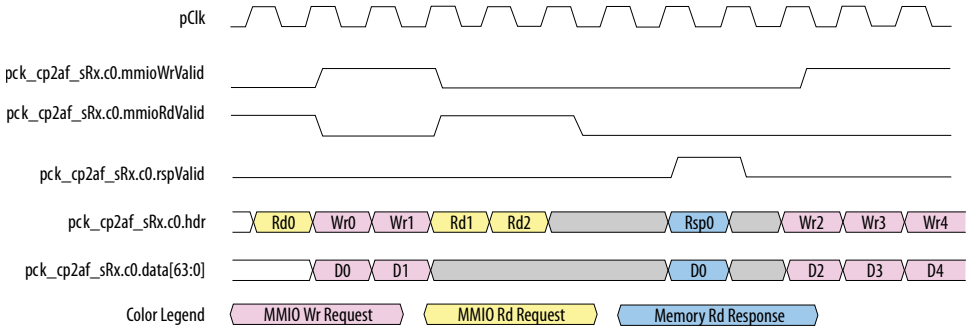
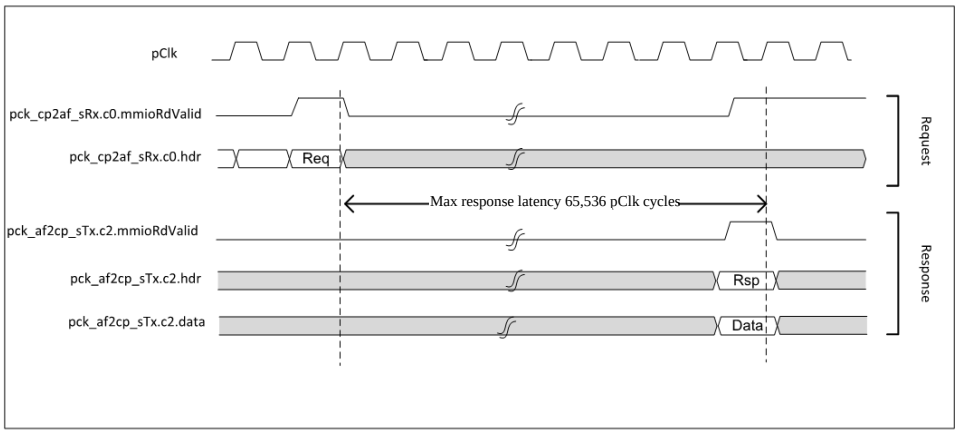


図 -22: MMIO 読み出し応答のタイムアウト



注意: AFU は、最大応答時間 65,536 pClk で MMIO 読み出しトランザクションに応答します。

1.3.15. CCI-P のガイダンス

この章では、インテル FPGA IP システムで FPGA 統合プラットフォームまたは インテル FPGA PAC の使用を開始する際に推奨される有効な手法および設定を提供します。

CCI-P インターフェイスは、FPGA キャッシュ状態および仮想チャネルを詳細に制御するための複数の高度な機能を提供します。それらを正しく使用することで、インターフェイスにわたる最適なパフォーマンスを実現することができます。それらが正しく使用されないと、パフォーマンスが大幅に低下する場合があります。

次の表に、リクエストフィールドに推奨されるパラメーターをいくつか示します。

表 40. メモリーリクエストに推奨される選択

フィールド	推奨されるオプション	
vc_sel	Producer と Consumer タイプのフローの場合	VA
	レイテンシーに影響されやすいフローの場合	VL0
	データに依存するフローの場合	VA を除く VC のいずれかを使用する、もしくは MPF の VC マップを使用する
cl_len	最大の帯域幅に向けて	4 CL (256 バイト)
req_type	メモリー読み出し	RdLine_I
	メモリー書き込み	WrLine_M

AFU でリクエスト・バッファー・サイズを設定する際は、次のガイダンスを使用します。

- インテル FPGA PAC
  - VH0 で 64 の未処理のリクエスト
  - VA および VH0 は、同じ 64 の未処理のリクエスト・バッファーを共有できます。
- FPGA 統合プラットフォーム
  - VH0 および VH1 はそれぞれ、64 の未処理のリクエストを持つことができます。
  - VL0 がフルの帯域幅に達するには最低 128 のインフライトのトランザクションを必要とします。また、長いレイテンシー・テイルに対処するのに 256 を超える未処理のリクエストは必要ありません。
  - VA の場合、最大限のパフォーマンスは最低 256、最大 384 のトランザクションで達成することができます。デザイン領域の節減に向け、VA バッファーをほかの VC と共有することを検討します。

1.4. AFU の要件

この章では、電源投入時における AFU の初期化フローと、必須の AFU のコントロールおよびステータスレジスター (CSR) について定義します。

AFU の CSR に関する詳細は、「デバイス・フィーチャー・リスト」の章を参照ください。

関連情報

デバイス・フィーチャー・リスト (51 ページ)

1.4.1. 必須 AFU CSR の定義

AFU の CSR に対するソフトウェア・アクセスには、次の要件が定義されています。

1. ソフトウェアは、アライメントされたクワッドワード (8 バイト) として 64 ビットの CSR にアクセスすることが想定されています。64 ビット・レジスターのフィールド (ビットまたはバイト) の変更には、クワッドワード全体が読み出され、適切なフィールドが変更され、クワッドワード全体が書き戻されます (リードモディファイライト動作)。
2. 32 ビットの CSR をサポートする AFU の場合も同様に、ソフトウェアは、アライメントされたダブルワード (4 バイト) でそれらにアクセスすることが想定されています。

CCI-P に準拠する AFU にはそれぞれ、次の表で定義されている 4 つの必須レジスター (0x0000 の DEV_FEATURE_HDR （DFH） を除く) の実装が必要です。これらのレジスターを実装していない場合、もしくは正しく実装していない場合、AFU の検出に失敗する、もしくは他の予期せぬ動作が起こる場合があります。

表 41. レジスター属性の定義

属性	展開	説明
RO	Read Only	ビットはハードウェアでのみ設定されます。ソフトウェアはこのビットの読み出しのみ可能です。書き込みには効果がありません。
Rsvd	Reserved	今後定義される内容に予約されています。AFU はこれらを 0 に設定する必要があります。ソフトウェアは、これらのフィールドを無視する必要があります。

表 42 (48 ページ)は、必須の AFU CSR のバイトオフセットおよび DWORD オフセットを示しています。ベースアドレスはプラットフォームで設定され、AFU で指定する必要はありません。

表 42. 必須 AFU CSR

名称	DWORD アドレス オフセット (CCI-P)	バイト・アドレス・オフセット (ソフトウェア)
DEV_FEATURE_HDR (DFH) ビットの説明に関しては、表 43 (49 ページ) を参照ください。 *注意:* AFU の CSR は 64 ビットです。	0x0000	0x0000
AFU_ID_L AFU_ID GUID の下位 64 ビット	0x0002	0x0008
AFU_ID_H AFU_ID GUID の上位 64 ビット	0x0004	0x0010
DFH_RSVD0	0x0006	0x0018
DFH_RSVD1	0x0008	0x0020

図 23 (49 ページ) は、AFU が必須 AFU CSR を設定する際の例を示しています。ご自身の独自の AFU ID を定義する必要があります。AFU は DWORD アドレスを使用していることに注意してください。図 24 (49 ページ) では、ソフトウェア・プログラムが AFU ID を読み出す際の例を示しています。

図 -23: AFU での必須 AFU レジスターの設定

```
t_ccip_c0_ReqMmioHdr mmioHdr;
:
:
case (mmioHdr.address)
// AFU header
       16'h0000 : af2cp_sTxPort.c2.data <= {                // DFH
          4'b0001,  // Feature Type        = AFU
          8'b0,     // Reserved
          4'b0,     // AFU Minor Revision  = 0
          7'b0,     // Reserved
          1'b1,     // End of DFH list     = 1
          24'b0,    // Next DFH offset     = 0
          4'b0,     // AFU Major version   = 0
          12'b0     // Feature ID          = 0
        };
     16'h0002 : af2cp_sTxPort.c2.data <= 64'ha12e_bb32_8f7d_d35c;   // AFU_ID_L
                                                                    // (arbitrary example)
     16'h0004 : af2cp_sTxPort.c2.data <= 64'ha455_783a_3e90_43b9;   // AFU_ID_H
                                                                    // (arbitrary example)
     16'h0006 : af2cp_sTxPort.c2.data <= 64'h0;                     // Reserved
     16'h0008 : af2cp_sTxPort.c2.data <= 64'h0;                     // Reserved
```

ソフトウェアと AFU RTL は同じ AFU ID を参照する必要があります。

図 -24: ソフトウェアによる AFU ID の読み出し

```
   btUnsigned32bitInt AFUID_H, AFUID_L;
:
:
   IALIMMIO      *m_pALIMMIOService;        //< Pointer to MMIO Service:
:
:
// the AFUID to be passed to the Resource Manager. It will be used to locate the appropriate device.
      ConfigRecord.Add(keyRegAFU_ID,"A455783A-3E90-43B9-A12E-BB328F7DD35C");
:
      m_pALIMMIOService->mmioRead32(0x0008, &AFUID_L);
      printf("Read AFUID_L= 0x%08x\n", AFUID_L);

      m_pALIMMIOService->mmioRead32(0x0010, &AFUID_H);
      printf("Read AFUID_H= 0x%08x\n", AFUID_H);
```

表 43. フィーチャー・ヘッダーの CSR 定義

アドレスオフセット = 0x0

ビット	属性	デフォルト	説明
63:60	RO	0x1	タイプ: AFU
59:52	Rsvd	0x0	予約済み
51:48	RO	0x0	AFU マイナーバージョン番号 ユーザー定義の値
47:41	Rsvd	0x0	予約済み
40	RO	該当なし	リストの末尾 1'b0: この後に別のフィーチャー・ヘッダーがあります (「次の DFH のバイトオフセット」を確認ください。) 1'b1: これはこの AFU の最後のフィーチャー・ヘッダーです。
39:16	RO	0x0	次のデバイス・フィーチャー・ヘッダーへのバイトオフセットです。つまり、現在のアドレスからのオフセットです。 DFH のバイトオフセット例に関しては、表 45 (54 ページ) を参照ください。
15:12	RO	0x0	AFU メジャーバージョン番号

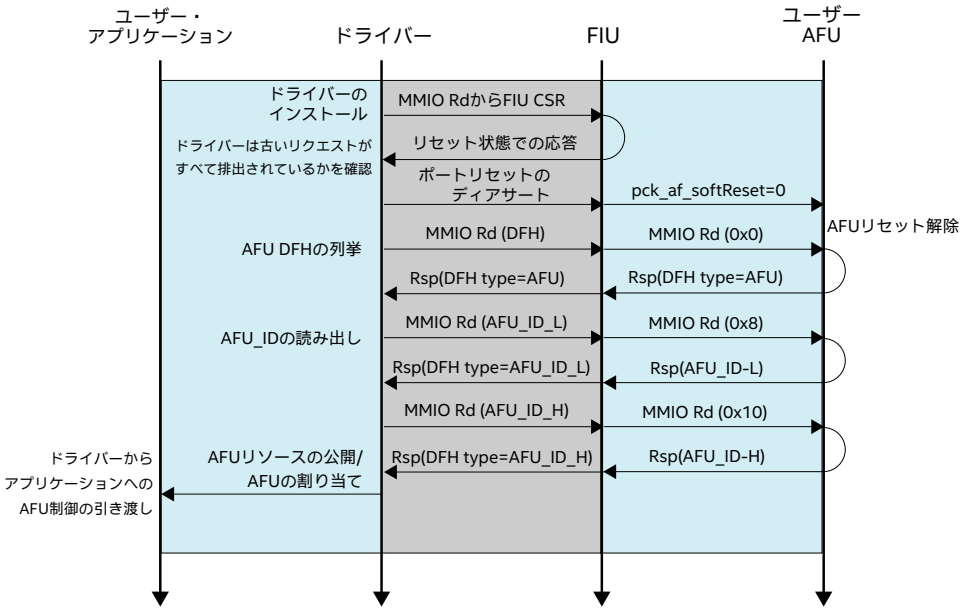
continued...

ビット	属性	デフォルト	説明
			ユーザー定義の値
11:0	RO	該当なし	CCI-P バージョン番号 ccip_if_pkg.sv の CCIP_VERSION_NUMBER パラメーターを使用します。

1.4.2. AFU の検出フロー

CCI-P に準拠する AFU には、必須の AFU CSR を実装する必要があります。次の図は、pck_cp2af_softReset がディアサートされた直後の最初のトランザクションを表しています。AFU は、ソフトリセットがディアサートされた直後に MMIO 読み出しサイクルを受け入れる必要があります。

図 -25: AFU の検出フロー



1.4.3. AFU_ID

AFU_ID の目的は、AFU のアーキテクチャーのインターフェイスを正確に識別することです。このインターフェイスは、AFU がソフトウェアと行うコントラクトです。

AFU の複数のインスタンスは同じ AFU_ID の値を持つことができますが、AFU のアーキテクチャーのインターフェイスが変更になった場合は新しい AFU_ID が必要です。

AFU のアーキテクチャーのインターフェイスは、AFU の機能、CSR の定義、CSR を操作する際に AFU が想定するプロトコル、およびバッファーに関するすべての暗黙的または明示的な仮定もしくは保証から成る AFU デザインの構文とセマンティクスで構成されています。

ソフトウェア・フレームワークとアプリケーション・ソフトウェアは、AFU_ID を使用しそれらが正しい AFU と一致することを確認します。つまり、それらは同じアーキテクチャーのインターフェイスに従っています。

AFU_ID は 128 ビットの値です。また、UUID/GUID ジェネレーターを使用し、値が一意になるように生成することができます。

UUID/GUID の詳細については「Online GUID Generator」の Web ページを確認ください。

関連情報

Online GUID Generator

1.5. インテル FPGA ベーシック・ビルディング・ブロック

インテル FPGA ベーシック・ビルディング・ブロック (BBB) は、インテルが提供する IP で、ユーザーが各自の AFU でインスタンス化できるものです。BBB には、ソフトウェアに可視化されるタイプ (レジスター・インターフェイスを公開し、ソフトウェアとの通信が必要) と、ソフトウェアに可視化されないタイプ (ソフトウェアの通信を必要としない) の 2 つがあります。どちらのタイプの場合においても、AFU デベロッパーはハードウェアとソフトウェアを AFU に統合する責任を負います。

BBB の詳細については、「Intel FPGA Basic Building Blocks (BBB)」の Web ページを参照ください。

関連情報

Intel FPGA Basic Building Blocks (BBB)

1.6. デバイス・フィーチャー・リスト

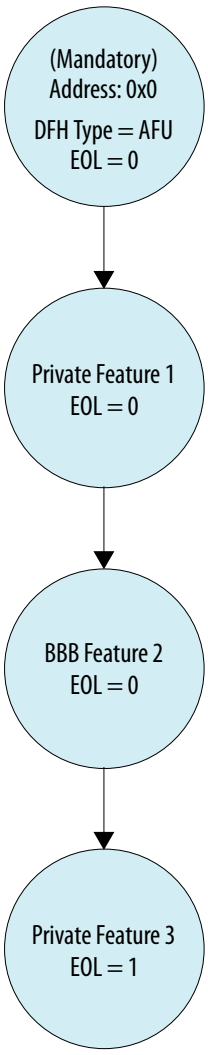
この章では、MMIO 空間内のフィーチャーのリンクリストを作成するデバイス・フィーチャー・リスト (DFL) 構造を定義し、それによりフィーチャーの追加および列挙を行う拡張可能な方法を提供します。フィーチャー領域 (「フィーチャー」と呼ばれる場合もあります) は、関連する CSR のまとまりです。例えば、DMA エンジンの 2 つの異なるフィーチャーに、キュー管理と QoS 機能があります。キュー管理と QoS 機能は、2 つの異なるフィーチャー領域にグループ化することが可能です。デバイス・フィーチャー・ヘッダー (DFH) レジスターは、フィーチャー領域の開始を示します。

ソフトウェアは DFL 全体を見渡し、以下を列挙することができます。

- AFU
  - AFU は CCI-P インターフェイスに準拠しており、CCI-P ポートに直接接続されます。AFU ID などの必須 AFU レジスターを実装する必要があります。
  - AFU DFH は、MMIO アドレス 0x0 に位置している必要があります。
- プライベート・フィーチャー
  - これは AFU 内のフィーチャーのリンクリストであり、AFU 内の機能を編成する方法を提供します。それらの列挙および管理は、AFU デベロッパーの責任です。
  - これは ID の実装を必要としません。
- インテル FPGA ベーシック・ビルディング・ブロック
  - AFU 内の特別なフィーチャーであり、再利用可能なビルディング・ブロックを意味します (一度デザインし、何度も再利用する)。ソフトウェアに可視化される インテル FPGA ベーシック・ビルディング・ブロックは一般的に、インテル FPGA ベーシック・ビルディング・ブロックの列挙とコンフィグレーションを行うための、対応するソフトウェア・サービスとともに提供されます。また、場合によっては インテル FPGA ベーシック・ビルディング・ブロックに向けたより高いレベルのソフトウェア・インターフェイスが提供されます。
  - AFU のような厳しいハードウェア・インターフェイス要件はありませんが、ソフトウェアの観点から明確に定義されたアーキテクチャーのセマンティクスを必要とします。
  - 可視化される場合、必須 DFH レジスターを実装する必要があります。
  - ソフトウェアに可視化される インテル FPGA ベーシック・ビルディング・ブロックに対してのみ、GUID を実装する必要があります。

次の図は、BBB とプライベート・フィーチャーで構成される AFU のフィーチャー階層の例を表しています。

図 -26: フィーチャー階層の例



デバイス・フィーチャー・ヘッダー (DFH) レジスター (下に示されています) は、フィーチャー領域の開始を示します。

表 44. デバイス・フィーチャー・ヘッダーの CSR

デバイス・フィーチャー・ヘッダー			
ビット	説明		
63:60	フィーチャー・タイプ		
	4'h1 – AFU	4'h2 – BBB	4'h3 – プライベート・フィーチャー
59:52	予約済み		
51:48	AFU マイナーバージョン番号 ユーザー定義の値	予約済み	
47:41	予約済み		

continued...

デバイス・フィーチャー・ヘッダー		
ビット	説明	
40	リストの末尾 1'b0: この後に別のフィーチャー・ヘッダーがあります (「次の DFH のバイトオフセット」を参照ください)。 1'b1: これは、この AFU の最後のフィーチャー・ヘッダーです。	
39:16	次の DFH のバイトオフセット 次の DFH アドレス = 現在の DFH アドレス + 次の DFH のバイトオフセット このフィーチャーが占有する MMIO 領域の最大サイズの指標としても使用されます。最後のフィーチャーの場合、このオフセットは割り当てられていない MMIO 領域の開始を指します (ある場合)。もしくは、MMIO 空間の範囲を超えます。 表 45 (54 ページ) にある例を参照ください。	
15:12	AFU メジャーバージョン番号 ユーザー定義	フィーチャーのリビジョン番号 ユーザー定義
11:0	CCI-P バージョン番号 ccip_if_pkg.sv の CCIP_VERSION_NUMBER パラメーターを使用します。	フィーチャー ID AFU 内のフィーチャーを識別するユーザー定義の ID を含みます。

表 45. 次の DFH のバイトオフセット例

フィーチャー	DFH アドレス	EOL	次の DFH のバイトオフセット
0	0x0	0x0	0x100
1	0x100	0x0	0x180
2 (最後のフィーチャー)	0x280	0x1	0x80
割り当てられていない MMIO 空間、DFH なし	0x300	該当なし	該当なし

BBB にタイプが設定されている DFH は、次の表にある必須 BBB レジスターを後に続ける必要があります。

表 46. 必須 BBB DFH レジスターマップ

DFH 内のバイト・アドレス・オフセット	レジスター名
0x0000	DFH Type=BBB
0x0008	BBB_ID_L
0x0010	BBB_ID_H

表 47. BBB_ID_L CSR の定義

レジスター名		BB_ID_L
ビット	属性	説明
63:0	RO	BBB_ID ID の下位 64 ビット

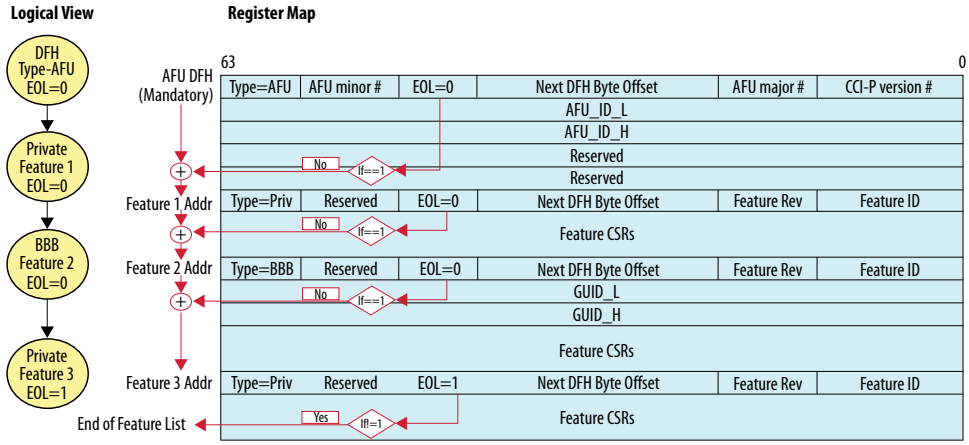
表 48. BBB_ID_H CSR の定義

レジスター名		BB_ID_H
ビット	属性	説明
63:0	RO	BBB_ID ID の上位 64 ビット

BBB_ID は GUID であり、概念的には AFU_ID に類似しています。それぞれの BBB が一意の識別子を持つように定義されます。これにより、ソフトウェアが BBB ハードウェアに関連付けられたソフトウェア・サービスを識別できるようになります。

次の図は、この章で定義されている DFH レジスターを使用し、論理フィーチャー階層 (左側に示されています) をどのように表現することができるかを示しています。

図 -27: デバイス・フィーチャーの概念図



1.7. インテル® アクセラレーション・スタック (インテル® Xeon® CPU & FPGA 対応) コア・キャッシュ・インターフェイス (CCI-P) リファレンス・マニュアルの改訂履歴

ドキュメント・バージョン	インテル・アクセラレーション・スタックのバージョン	変更内容
2019.11.04	2.0.1 (インテル Quartus® Prime プロ・エディション 19.2 でサポートされています)、2.0 (インテル Quartus Prime プロ・エディション 18.1.2 でサポートされています)、1.2 (インテル Quartus Prime プロ・エディション 17.1.1 でサポートされています)	CCI-P バイト・イネーブルのフィーチャーを追加しました。
2019.08.05	2.0 (インテル Quartus Prime プロ・エディション 18.1.2 でサポートされています) および 1.2 (インテル Quartus Prime プロ・エディション 17.1.1 でサポートされています)	• *アクセラレーション・スタック (インテル Xeon CPU & FPGA 対応) コア・キャッシュ・インターフェイス (CCI-P) リファレンス・マニュアルの頭字語一覧*において、RdLine_I の頭字語に、インテル FPGA プログラマブル・アクセラレーション・カード (インテル FPGA PAC) を追加しました。 • *メモリーおよびキャッシュ階層*において、インテル FPGA PAC のメモリー階層の図を更新しました。 • *CCI-P インターフェイス*において、CCI-P 信号の図を更新しました。 • *MMIO リクエスト*の次の文において、64 バイトを 64 ビットに変更しました。「FIU は、AFU の MMIO アドレス空間を **64 ビット**のプリフェッチ可能な PCIe BAR にマッピングします。」
2018.12.04	1.2 (インテル Quartus Prime プロ・エディション 17.1.1 でサポートされています)	このドキュメントのアーカイブバージョンを含む「インテル・アクセラレーション・スタック (インテル Xeon CPU & FPGA 対応) コア・キャッシュ・インターフェイス (CCI-P) リファレンス・マニュアルのアーカイブ」の章を追加しました。
2018.08.06	1.1 (インテル Quartus Prime プロ・エディション 17.1.1 でサポートされています) および 1.0 (インテル Quartus Prime プロ・エディション 17.0.0 でサポートされています)	「FIU 機能の比較」の章にある表 6 からクロック周波数の詳細を削除しました。また、このドキュメントから「クロック周波数」の章を削除しました。 *注意:* このドキュメントでは複数のプラットフォームについて説明しているため、クロック周波数はこのドキュメントから削除されました。
2018.04.11	1.0 (インテル Quartus Prime プロ・エディション 17.0 でサポートされています)	• インテル FPGA PAC と FPGA 統合プラットフォームの違いを明確に定義するため、ドキュメントの構成を変更しました。 • 「メモリーリクエスト」の章に追加された書き込み、読み出し、書き込みフェンスに関して、IRQ の順序付けを追加しました。