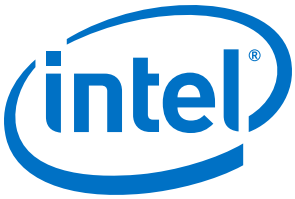


インテル® FPGA SDK for OpenCL

カスタム・プラットフォーム・ツールキット・ユーザーガイド

インテル® Quartus® Prime 開デザインスイートの更新情報: **16.1**

更新情報
フィードバック

UG-OCL007 | 2016.10.31
最新版をウェブからダウンロード: PDF | HTML

目次

1 インテル® FPGA SDK for OpenCL™ カスタム・プラットフォーム・ツールキット・ユーザーガイド

インテル® FPGA SDK for OpenCL™ カスタム・プラットフォーム・ツールキット・ユーザーガイド では、インテル FPGA SDK (ソフトウェア開発キット) for OpenCL カスタム・プラットフォームの作成手順について説明します。

インテル FPGA SDK for OpenCL [1] [2]カスタム・プラットフォーム・ツールキットは、完全に機能するカスタム・プラットフォームの実装に必要なツールを提供します。カスタム・プラットフォーム・ツールキットは環境変数 *ALTERAOCLSDKROOT* が SDK インストーレーションの位置を参照する *ALTERAOCLSDKROOT*/`board` ディレクトリー内で有効です。

SDK ユーザーは、次のタスクにより任意のカスタム・プラットフォームをシームレスにターゲットにすることができます。

1. アクセラレーター・ボードを入手し、システムに接続します。
2. カスタム・プラットフォームを入手し、ローカル・ディレクトリーに解凍します。
3. 環境変数 *AOCL_BOARD_PACKAGE_ROOT* がこのローカル・ディレクトリーを指すように設定します。
4. ターゲットデバイスに応じて、環境変数 *QUARTUS_ROOTDIR_OVERRIDE* が Quartus® Prime スタンダード・エディション・ソフトウェアまたは Quartus Prime プロ・エディション・ソフトウェアのインストール・ディレクトリーを指すように設定します。
5. `aocl install` ユーティリティー・コマンドを呼び出します。
6. OpenCL カーネルをコンパイルし、ホスト・アプリケーションをビルドします。
7. 環境変数を MMD (memory-mapped デバイス) ライブラリーの位置を指すように設定します。
   - Windows システムでは、*PATH* 環境変数を設定します。
   - Linux システムでは、*LD_LIBRARY_PATH* 環境変数を設定します。
8. ホスト・アプリケーションを実行します。

1.1 インテル FPGA SDK for OpenCL カスタム・プラットフォーム・ツールキットの前提条件

インテル FPGA SDK for OpenCL カスタム・プラットフォーム・ツールキット・ユーザーガイド は、インテル FPGA SDK for OpenCL カスタム・プラットフォームの作成に使用するカスタム・プラットフォーム・ツールキットに関するハードウェア・デザインの知識があることを前提としています。

---

[1] OpenCL および OpenCL のロゴは、Khronos Group™ の使用許諾による Apple Inc. の商標です。

[2] インテル FPGA SDK for OpenCL は、Khronos 社公開の仕様に基づいており、Khronos コンフォーマンス・テスト・プロセスに合格しています。現在の規格適合性の状態については www.khronos.org/conformance を参照してください。

ハードウェア・デザインの経験は、以下の場合に必要になります。

- Qsys、HDL および Tcl での Quartus Prime ソフトウェア・デザイン
- ボードの物理インターフェイスの通信に必要な インテル FPGA IP (Intellectual Property)
- 高速デザイン、タイミング解析および SDC (Synopsys Design Constraints) 制約
- クロックとグローバル配線、フロア・プランニング、および I/O を含む FPGA アーキテクチャー
- チームベースのデザイン (インクリメンタル・コンパイル)

コンピューターに、Quartus Prime ソフトウェア、関連のあるデバイスのサポートファイル、および SDK をインストールする必要があります。ターゲットデバイスにより、Quartus Prime スタンダード・エディション・ソフトウェア、Quartus Prime プロ・エディション・ソフトウェア、またはその両方のソフトウェアをインストールする必要があります。インストールの手順については、*インテル FPGA SDK for OpenCL スタートガイド*を参照してください。

カスタム・プラットフォーム・デザインには、以下のオプションがあります。

- この資料の情報を参照し、カスタム・プラットフォーム・ツールキットで使用可能なテンプレートからカスタム・プラットフォームを作成します。
- この資料と *インテル FPGA SDK for OpenCL Stratix V Network Reference Platform Porting Guide* を参照にし、Stratix V ネットワーク・リファレンス・プラットフォーム (s5_net) に関連するファイルを変更して、カスタム・プラットフォームを作成します。

  アルテラのウェブサイト (www.altera.co.jp) の インテル FPGA SDK for OpenCL FPGA プラットフォームのページから s5_net をダウンロードします。ダウンロードのリンク先は、**Custom** 内にあります。
- 次の資料を参照し、Cyclone® V SoC 開発キット・リファレンス・プラットフォーム (c5soc) に関連するファイルを変更して、SDK で使用可能なカスタム・プラットフォームを作成します。
  1. *インテル FPGA SDK for OpenCL カスタム・プラットフォーム・ツールキット・ユーザーガイド*
  2. *インテル FPGA SDK for OpenCL Cyclone V SoC Development Kit Reference Platform Porting Guide*
  3. *Cyclone V SoC Development Board Reference Manual*
- この資料と *インテル FPGA SDK for OpenCL Arria 10 GX FPGA Development Kit Reference Platform Porting Guide* を参照にし、Arria® 10 GX FPGA Development Kit Reference Platform で関連ファイルを変更して、SDK で使用可能なカスタム・プラットフォームを作成します。

関連情報

- アルテラ SDK for OpenCL スタートガイド
- Stratix V Network Reference Platform Porting Guide
- Cyclone V SoC Development Kit Reference Platform Porting Guide
- Cyclone V SoC Development Board Reference Manual
- アルテラのウェブサイトのインテル FPGA SDK for OpenCL FPGA プラットフォームのページ

1.2 インテル FPGA SDK for OpenCL カスタム・プラットフォームの概要

インテル FPGA SDK for OpenCL カスタム・プラットフォームは、インテル FPGA SDK for OpenCL オフライン・コンパイラーと FPGA ボード間の通信に必要なツールとライブラリーの集まりです。

現在、オフライン・コンパイラーは、一度にシングル・カスタム・プラットフォームをターゲットにします。

環境変数 *AOCL_BOARD_PACKAGE_ROOT* は、カスタム・プラットフォーム内の `board_env.xml` ボード環境 XML (eXtensible Markup Language) ファイルのパスを参照にします。特定のカスタム・プラットフォームのインストールには、同じボードのインターフェイスのいくつかのボード・バリエーションが含まれています。異なる FPGA パーツ、またはボード・インターフェイスの異なるサブセットをサポートする必要がある場合があります。ボード・バリエーションを同じ位置に設置することで、複数のデバイス環境で異なるボードとの同時通信が可能になります。

インテル FPGA SDK for OpenCL カスタム・プラットフォームには、以下のコンポーネントが含まれています。

- **Quartus Prime スケルトン・プロジェクト**—ボードの Quartus Prime プロジェクトです。SDK のオフライン・コンパイラーはコンパイルしたカーネルを含むように変更します。このプロジェクトは、カーネルクロックで制御されていないすべてのロジックでの配置配線のパーティションを含んでいる必要があります。
- **ボード・インストールのセットアップ**—ボードと、ボードの異なるコンポーネントの説明です。
- **汎用 I/O インターフェイス**—ホストとボードの間の基本の I/O を実装する MMD ソフトウェア・ライブラリーです。
- **ボード・ユーティリティー**—ボードのインストールやテストをするタスクを含んだアクセラレーター・ボードを管理するための SDK のユーティリティーの実装です。

1.2.1 インテル FPGA SDK for OpenCL カスタム・プラットフォームのディレクトリーとファイル

ターゲット FPGA ボード上で OpenCL カーネルを実行可能にするために、ファイル、ライブラリー、ドライバーで インテル FPGA SDK for OpenCL カスタム・プラットフォームを実装します。

表 1. トップレベル・カスタム・プラットフォーム・ディレクトリー内の内容

内容	説明
`board_env.xml`	SDK にボード・インストーレーションを記述する XML ファイルです。
<hardware>	特定のカスタム・プラットフォーム内でサポートされるボードの QuartusPrime プロジェクトを含んだディレクトリーです。`board_env.xml` ファイルでこのディレクトリー名を指定します。このディレクトリー内で、SDK は `board_spec.xml` ファイルを含んだ任意のサブディレクトリーがボードであると見なします。
`include`	ボード固有のヘッダーファイルを含んだディレクトリーです。
`source`	ボード固有のファイル、ライブラリー、ドライバーを含んだディレクトリーです。
`platform`	プラットフォーム固有 (x86_64 Linux など) のドライバーとユーティリティーを含んだディレクトリーです。

1.2.2 カスタム・プラットフォーム・ディレクトリー構造の推奨事項

使い易くするために、インテル FPGA SDK for OpenCL カスタム・プラットフォームの作成する際に、インテル 推奨のディレクトリー構造と命名規則の採用を考慮します。

- *ALTERAOCLSDKROOT* が SDK インストーレーションの位置を参照するように、*ALTERAOCLSDKROOT*/board ディレクトリーをボード・インストーレーションの位置に作成します。

  注意: 既存のすべてのサブディレクトリーを *ALTERAOCLSDKROOT*/board ディレクトリーから削除しないでください。
- カスタム・プラットフォームを保存するために、*ALTERAOCLSDKROOT*/board ディレクトリー内に *<board_vendor_name>* サブディレクトリーを作成します。
- 特定のカスタム・プラットフォームの内容を *ALTERAOCLSDKROOT*/board/*<board_vendor_name>*/*<board_family_name>* サブディレクトリーに保存します。
- 同一名前を回避するために、ソフトウェア・ライブラリに一意の名前 (lib*<board_vendor_name>*_*<board_family_name>*.so など) を付けます。

例として ABC Incorporated が XYZ という名前のボードファミリーのカスタム・プラットフォームを作成する場合、SDK ユーザーが次のタスクを実行して XYZ にアクセスできるように、カスタム・プラットフォームを設定します。

1. XYZ カスタム・プラットフォームを、*ALTERAOCLSDKROOT* が SDK インストール・パッケージの絶対パスを参照する環境変数である *ALTERAOCLSDKROOT*/board/ABC/XYZ にインストールします。
2. *AOCL_BOARD_PACKAGE_ROOT* 環境変数を *ALTERAOCLSDKROOT*/board/ABC/XYZ を指すように設定します。

1.3 前方互換性でのカスタム・プラットフォームの自動移行

自動移行の機能は、現バージョンの Quartus Prime デザインスイートと インテル FPGA SDK for OpenCL で使用するために、既存の インテル 登録済みのカスタム・プラットフォームを更新します。

重要: カスタム・プラットフォームが インテル FPGA リファレンス・プラットフォームに可能な限り類似している場合、自動移行は正常に完了する可能性が高くなります。

バージョン 14.0 以降のカスタム・プラットフォームには、次の情報が適用されます。

1. SDK を含んだ現バージョンの Quartus Prime デザインスイートとともに使用するカスタム・プラットフォームを更新する場合は、カスタム・プラットフォームを変更しないでください。自動移行は、特定の特性に基づいてカスタム・プラットフォームのバージョンを検出し、自動的にその更新を実行します。
2. カスタム・プラットフォームを変更し、これを SDK を含む現バージョンの QPDS とともに使用するために更新する場合は、現バージョンのカスタム・プラットフォームにすべての必須機能を実装します。カスタム・プラットフォームを変更した後、自動移行は特性を正しく検出できません。そのため、手動でカスタム・プラットフォームを更新しなければなりません。

正常に移行されたカスタム・プラットフォームは、元の機能を保持します。ほとんどの場合、新しい QPDS または SDK バージョンの新機能は、カスタム・プラットフォームの機能を妨げません。

インテル FPGA SDK for OpenCL オフライン・コンパイラーはカーネルをコンパイルすると、次の情報のために board_spec.xml ファイルを検索します。

1. board XML エレメントの version 属性で指定されているカスタム・プラットフォームのバージョン
2. compile XML エレメントの auto_migrate 属性の platform_type パラメーターで指定されているプラットフォーム・タイプ

この情報に基づき、SDK はカスタム・プラットフォームの移行中に適用する必要がある一連の修正を指定します。修正は、オフライン・コンパイラーが OpenCL カーネルのコンパイルに使用する Quartus Prime プロジェクトに適用します。また、適用された修正を記述中の SDK ユーザーが現在作業中のディレクトリー内に `automigration.rpt` レポートファイルを生成します。

自動移行のプロセスは、インストールされたカスタム・プラットフォームの変更を行いません。

注意: 自動移行が失敗した場合は、お近くの インテル FPGA 製品の販売代理店にお問い合わせください。

1.3.1 自動移行のカスタマイズ

インストールされたカスタム・プラットフォームの自動移行は、ボード開発者も インテル FPGA SDK for OpenCL ユーザーも無効にすることができます。加えて、カスタム・プラットフォームに適用する SDK で識別された名前付き修正の選択も可能です。

1. 次のいずれかの方法で自動移行を無効にします。
   - ボード開発者の場合は、`board_spec.xml` ファイルにある `compile`XML エレメント内で、`auto_migrate` 属性の `platform_type` パラメーターを `none` に設定します。
   - SDK ユーザーの場合は、`aoc --no-auto-migrate` コマンドを呼び出します。
2. SDK が識別する修正を明示的に含めるか除外するには、`board_spec.xml` ファイルで、各修正をそれぞれ `include fixes` または `exclude fixes` パラメーターにリストしてサブスクライブまたはサブスクライブを解除します。`include fixes` と `exclude fixes` パラメーターは、`compile` エレメント内の `auto_migrate` 属性の一部です。複数の修正をリストする際は、各修正をコンマで区切ってください。

   `include fixes` と `exclude fixes` パラメーターで指定する修正のファイル名については、`automigration.rpt` ファイルを参照してください。

1.4 インテル FPGA SDK for OpenCL カスタム・プラットフォームの作成

インテル FPGA SDK for OpenCL を使用したカスタム・プラットフォームの作成の際に実行する必要があるタスクを、下の各項目で説明します。

1. 8 ページの ボード・ハードウェアのデザイン
2. 13 ページの ボード XML ファイルの作成
3. 18 ページの MMD ライブラリーの作成
4. 20 ページの Altera Client Driver のセットアップ
5. 22 ページの インテル FPGA SDK for OpenCL ユーティリティー・サポートの提供
6. 23 ページの ハードウェア・デザインのテスト

1.4.1 ボード・ハードウェアのデザイン

インテル FPGA SDK for OpenCL を使用してアクセラレーター・ボードをデザインするには、すべてのボードとシステム・コンポーネント、およびハードウェア・デザインを記述するファイルを インテル FPGA SDK for OpenCL オフライン・コンパイラーに作成する必要があります。

カスタム・プラットフォームの各ボードバリアントは、Quartus Prime プロジェクトと、オフライン・コンパイラーにシステムを記述する `board_spec.xml`XML ファイルで構成されています。`board_spec.xml` ファイルには、カーネルへの接続に必要なインターフェイスが記述されています。

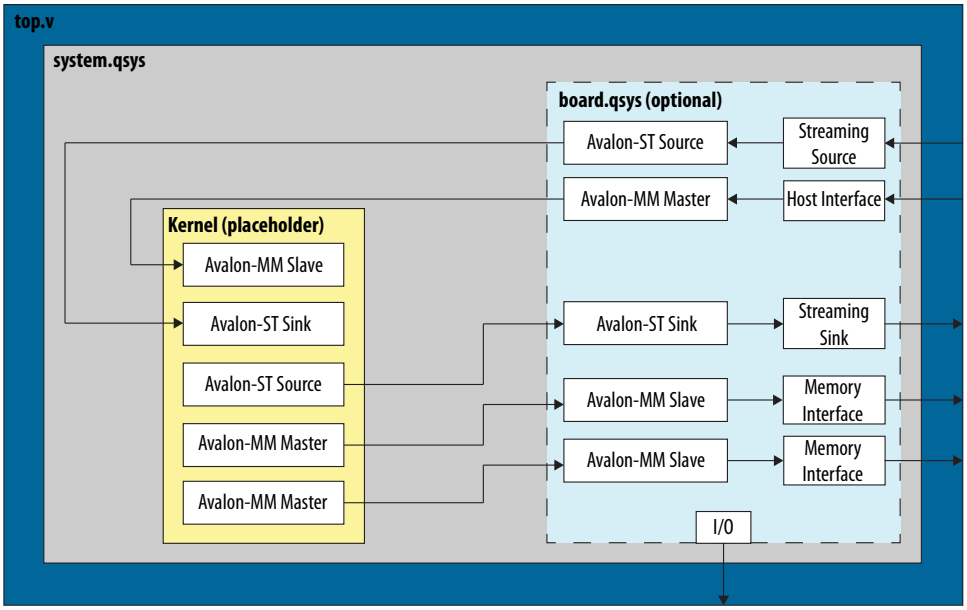
オフライン・コンパイラーは、board_spec.xml ファイルからのデータに基づいてカスタム回路を生成します。次に、すべての非カーネルロジック用に作成した Qsys システムに OpenCL カーネルを組み込みます。

すべての非カーネルロジックのデザインを保持する必要があります。次のいずれかの方法で、Quartus Prime ソフトウェアにデザインを保存することができます。

- 単一 HDL 階層の下にすべての非カーネルロジックを含んでいるデザイン・パーティションを作成し、パーティションをエクスポートします。例えば、board.qsys Qsys サブシステムを作成してエクスポートすることができます (下図を参照)。トップレベルの system.qsys Qsys システムは、このエクスポートされたボードの Qsys サブシステムをインスタンス化することができます。
- デザイン・パーティションの外のすべてのロジックを保持する CvP (Configuration via Protocol) コンフィグレーション・スキームを実装します。この場合、カーネルロジックの周りにパーティションを作成するだけです。すべての非カーネルロジックを単一のトップレベル Qsys システムファイル (例えば、system.qsys) に配置できる可能性があります。

すべてのコンポーネントとシステムを記述する board_spec.xml ファイルを SDK にデザインする必要があります。

図 -1: ボード Qsys サブシステムでのシステム階層例



1. 9 ページの ボード Qsys システムの作成
2. 13 ページの 保証されたタイミングフローの確立

1.4.1.1 ボード Qsys システムの作成

ボード・ハードウェアのデザインでは、オプションですべてのボードロジックを含む system.qsys 内に Qsys サブシステムを作成することができます。デザインコードの編成に加え、このサブシステムでは保存が可能な Quartus Prime パーティションを作成することができます。Qsys サブシステムでボードシステムを作成するには、カスタム・プラットフォーム・ツールキットで board.qsys テンプレートを変更する必要があります。

ボード Qsys サブシステムの実装は、次のコンポーネントを含んでいます。

- 適切なリセットシーケンス
- インテル FPGA SDK for OpenCL 固有のコンポーネント
- ホストから FPGA への通信 IP
- SDK のグローバルメモリーで使用されるメモリー IP
- ボード固有のインターフェイスへのストリーミング・チャネル

詳しい情報は、*ボード Qsys システム*の項を参照してください。

以下のハードウェア・デザイン・ファイルのテンプレートは、 *ALTERAOCLSDKROOT*/board/custom_platform_toolkit/board_package/hardware/template ディレクトリー内で使用可能です。

- board.qsys
- system.qsys
- top.v
- top.qpf
- board_spec.xml

post_flow.tcl ファイルのテンプレートは、カスタム・プラットフォーム・ツールキットの *ALTERAOCLSDKROOT*/board/custom_platform_toolkit/board_package/hardware/template/scripts ディレクトリー内で使用可能です。

非カーネルロジックを作成するには、system.qsys のトップレベルの Qsys システムまたはボード Qsys サブシステムで、次のタスクを実行します。

1. Qsys では、ホストとメモリー IP を Qsys システムに追加し、すべての必要な接続とエクスポートを確立します。

   注意: 個別の IP ライセンスの取得が必要な場合があります。有効なライセンスとライセンスがない IP ソリューションのリストは、アルテラのウェブサイト (www.altera.co.jp) の「All Intellectual Property」のページをご覧ください。各 IP の詳細は、「製品名」カラムのリンクをクリックして製品ページに移動し、参照してください。

   a. por_reset_controller/clk をドライブするように、ホスト・インターフェイス・クロックを接続します。デザインのグローバルリセットとクロック入力は、リセットカウンター (por_reset_counter) に供給されます。次に、このリセットカウンターは、Merlin Reset Controller (por_reset_controller) のホスト・インターフェイス・クロックに同期します。

      por_reset_counter ACL SW Reset コンポーネントは、パワーオンリセットを実装します。FPGA のコンフィグレーションが完了した後、サイクルの数のリセットの発行により、すべてのデバイス・ハードウェアをリセットします。

   b. ホスト IP からの要求を受信できるように、Avalon®Avalon-MM (メモリー・マップド) Pipeline Bridge コンポーネントの pipe_stage_host_ctrl パラメーターを変更します。ホスト・インターフェイスの Avalon-MM マスターポートを pipe_stage_host_ctrl の s0 ポートに接続します。pipe_stage_host_ctrl の m0 ポートを、OpenCL Kernel Clock Generator と OpenCL Kernel Interface コンポーネントを含んだホスト・インターフェイスと通信する必要のあるすべてのペリフェラルに接続します。

c. Avalon-MM Clock Crossing Bridge コンポーネントの `clock_cross_kernel_mem_<N>` の数を、ボード上のメモリー・インターフェイスの数と一致するように調整します。このコンポーネントは、カーネルとメモリー・インターフェイスの間でクロック・クロッシングを実行します。各コンポーネントのパラメーターを、OpenCL Memory Bank Divider コンポーネントのパラメーターと `board_spec.xml` に記述されている `interface` 属性と一致するように変更します。`m0` マスター、クロック、および `clock_cross_kernel_mem_<N>` リセットポート (つまり、それぞれ `m0`、`m0_clk`、および `m0_reset`) をメモリー IP に接続します。

重要: OpenCL Memory Bank Divider コンポーネントからの `kernel_reset` のアサーションがこのリセットをトリガーする方法で、`m0_reset` を接続します。

2. SDK 固有の Qsys システム・コンポーネントをカスタマイズします。

注意: `board.qsys` システム・テンプレートを使用して Qsys サブシステムを作成する場合、SDK 固有のシステム・コンポーネントと `board_spec.xml` ファイルと一致するようにエクスポートされた適切なインターフェイスとの間での必要な接続があらかじめ設定されていることに注意してください。インテル は、あらかじめ設定された接続をできるだけ保持することを推奨しています。

a. Qsys では、**Tools** > **Options** をクリックします。**Options** のダイアログボックスで、*ALTERAOCLSDKROOT*`/ip/board` を **Qsys IP Search Path** に追加し、**Finish** をクリックします。

b. OpenCL Kernel Clock Generator コンポーネントをインスタンス化します。コンポーネントのパラメーターを指定し、*OpenCL Kernel Clock Generator* の説明のとおりに信号とポートを接続します。

c. OpenCL Kernel Interface コンポーネントをインスタンス化します。コンポーネントのパラメーターを指定し、*OpenCL Kernel Interface* の説明のとおりに信号とポートを接続します。

d. 各グローバルメモリーのタイプの場合、OpenCL Memory Bank Divider コンポーネントをインスタンス化します。コンポーネントのパラメーターを指定し、*OpenCL Memory Bank Divider* の説明のとおりに信号とポートを接続します。

注意: 結果になるバンクマスターが `board_spec.xml` ファイルにある `global_mem` エレメントのインターフェイス属性で定義されるように、カーネルからのものと同等のアドレスビットとバースト幅を有するように、パラメーターを設定します。各メモリーバンクでは、Qsys は仕様として同じ特性を継承するマスターを生成します。

3. 非カーネルロジックの Qsys サブシステムの作成を選択した場合、必要なすべての I/O をトップレベルの `system.qsys` Qsys システムにエクスポートします。
4. トップレベルの `top.v` ファイルを編集して `system.qsys` をインスタンス化し、ボード固有の I/O を接続します。
5. ボードデザインのために必要なすべての設定を使用して `top.qpf` Quartus Prime プロジェクトを設定します。
6. Quartus Prime のコンパイル時に `fpga.bin` ファイルを生成する Tcl コードを含めるために、`Post_flow.tcl` ファイルを変更します。

   `fpga.bin` ファイルは、ボードのプログラミングで必要です。
7. ボード固有の記述を含めるために、`board_spec.xml` ファイルを編集します。

関連情報

- アルテラのウェブサイトのすべての IP のページ
- 32 ページの OpenCL Kernel Clock Generator

- 33 ページの OpenCL Kernel Interface
- 34 ページの OpenCL Memory Bank Divider
- 31 ページの ボード Qsys サブシステム

1.4.1.1.1 エクスポートされたボード・パーティションの一般的な品質に関する考察

配置配線のパーティションを生成する際、インテル FPGA SDK for OpenCL コンパイルで予期しない結果が起こり得るエクスポートされたボード・パーティションで、いくつかのデザイン考慮事項を考察しなければなりません。ボード・パーティションのベストな最適化の方法は、さまざまな異なる OpenCL カーネルで試すことです。

下のキャプチャーは、SDK のコンパイル結果の品質に影響を与え得るいくつかのパラメーターです。

- Resources Used

  パーティションが OpenCL カーネルで使用できるリソースを最大化するために使用するリソースの数を最小化します。

- Kernel Clock Frequency

  インテル は、カーネルクロックが高いクロック制約 (例えば、Stratix® V デバイスでは 350 MHz 以上) を有することを推奨しています。カーネルクロックでクロックされるパーティションでのロジックの量は、比較的少量でなければなりません。このロジックは、最もシンプルな OpenCL カーネルであっても、カーネルクロックのスピードを制限すべきではありません。したがって、少なくともパーティション内ではカーネルクロックは高いクロック制約でなければなりません。

- Host-to-Memory Bandwidth

  ホストからメモリーへの帯域幅は、ホスト・プロセッサーからアクセラレーター・カード上の物理メモリーまでの転送速度です。このメモリー帯域幅を測定するには、カスタム・プラットフォーム・ツールキットに含まれているホスト・アプリケーションをコンパイルして実行します。

- Kernel-to-Memory Bandwidth

  カーネルからメモリーへの帯域幅は、OpenCL カーネルとグローバルメモリーの間で可能な最大転送速度です。

  このメモリー帯域幅を測定するには、カスタム・プラットフォーム・ツールキットの`/tests/boardtest/host` ディレクトリーに含まれているホストプログラムをコンパイルして実行します。

- Fitter Quality of Results (QoR)

  デバイスのリソースを多く消費する OpenCL デザインが依然として高いクロック周波数が達成させるために、パーティションを FPGA のエッジに領域制約します。制約は、OpenCL カーネルロジックがデバイスの中心を占有でき、すべてのノードとの接続性も最も高くなります。

  大規模なデザインをテストコンパイルし、パーティション内の他のフィッターが引き起こすアーティファクトがカーネルコンパイルの QoR を妨げないようにします。

- Routability

  パーティションが消費する配線リソースは、コンパイルした OpenCL デザインの配線に影響を与えます。カーネルは、FPGA 上のすべての DSP (デジタル信号処理) ブロックまたはメモリーブロックを使用する場合がありますが、しかしながら、パーティションが使用する配線リソースでこれらのブロックの 1 つが配線不能になる可能性があります。この配線問題は、Quartus Prime プロジェクトのコンパイルがフィッティング・ステップで失敗する原因となります。したがって、すべての DSP およびメモリーブロックを使用するデザインでのパーティションのテストが不可欠です。

1.4.1.2 保証されたタイミングフローの確立

カスタム・プラットフォームの一部として、クリーン・タイミング・クロージャー・フローを有する非カーネルロジックにおけるデザイン・パーティションを提供します。

1. Quartus Prime ソフトウェアのインクリメンタル・コンパイル機能を使用し、配置および配線されたデザイン・パーティションを作成します。これは、非カーネルロジックのデザイン・パーティションです。

   タイミング・クロージャーのデザイン・パーテンションの生成のためのインクリメンタル・コンパイル機能の使用法について詳しくは、*Quartus Prime スタンダード・エディション・ハンドブック Vol.1* の *Quartus PrimeIncremental Compilation for Hierarchical and Team-Based Design* (英語版) を参照してください。
2. コンパイルフローの一部として、ステップ 1 の Post-Fit パーティションをトップレベルのデザインにインポートします。
3. *ALTERAOCLSDKROOT* が インテル FPGA SDK for OpenCL インストーレーションのパスを指す *ALTERAOCLSDKROOT*`/ip/board/bsp/adjust_plls.tcl` スクリプトを Post-Flow プロセスとして実行します。

   `adjust_plls.tcl` スクリプトは、最大カーネルクロック周波数を決定し、OpenCL Kernel Clock Generator コンポーネントの `pll_rom` オンチップメモリーを保存します。

関連情報

Quartus Prime Incremental Compilation for Hierarchical and Team-Based Design

1.4.2 ボード XML ファイルの作成

カスタム・プラットフォームは、カスタム・プラットフォームとそれぞれのハードウェア・システムを インテル FPGA SDK for OpenCL に記述する XML ファイルを含んでいる必要があります。これらの XML ファイルは、シンプル・テキスト・エディター (Windows ではワードパッド、Linux では vi など) で作成することができます。

13 ページの board_env.xml ファイルの作成

15 ページの board_spec.xml ファイルの作成

1.4.2.1 board_env.xml ファイルの作成

インテル FPGA SDK for OpenCL オフライン・コンパイラーがカスタム・プラットフォームをターゲットにする場合、インテル FPGA SDK for OpenCL ユーザーは環境変数 *AOCL_BOARD_PACKAGE_ROOT* を `board_env.xml` ファイルが存在する Custom Platform ディレクトリーを参照するように設定しなければなりません。`board_env.xml` ファイルは、カスタム・プラットフォームを SDK に記述します。カスタム・プラットフォームの他の内容とともに、`board_env.xml` ファイルはオフライン・コンパイラーが特定のアクセラレーター・ボードをターゲットにできるボード・インストーレーションを設定します。

`board_env.xml` テンプレートは、カスタム・プラットフォーム・ツールキットの`/board_package` ディレクトリーにあります。

1. トップレベルの `board_env` XML エレメントを作成します。`board_env` 内に、次の XML エレメントを含めます。
   - `hardware`
   - `platform`

カスタム・プラットフォームがサポートする各動作システムで、platform エレメントを含めます。

2. 各 platform エレメント内に、次の XML エレメントを含めます。
   - mmdlib
   - linkflags
   - linklibs
   - utilbindir
3. 次の表の概要を参照に、各エレメントおよび対応する属性をカスタム・プラットフォームに固有の情報でパラメーター化します。

表 2. board_env.xml ファイルの XML エレメントと属性の仕様

エレメント	属性の説明
board_env	Version: カスタム・プラットフォームの作成に使用する SDK のカスタム・プラットフォーム・ツールキットのリリースです。 ***注意:*** カスタム・プラットフォームのバージョンは、カスタム・プラットフォームの開発に使用する SDK のバージョンと一致している必要があります。 name: カスタム・プラットフォームを含んでいるボード・インストレーション・ディレクトリー名です。
hardware	dir: ボードバリアントを含むボード・インストレーション・ディレクトリー内のサブディレクトリー名です。 default: SDK ユーザーが --board <*board_name*> オフライン・コンパイラー・オプションで明示的な引数を指定しない際にオフライン・コンパイラーがターゲットにするデフォルトのボードバリアントです。
platform	Name: 動作システムの名前です。 詳しくは、*インテル FPGA SDK for OpenCL スタートガイド*および *インテル FPGA RTE for OpenCL Getting Started Guide* を参照してください。
mmdlib	カスタム・プラットフォームの MMD ライブラリーへのパスを指定する文字列です。 複数のライブラリーをロードするには、それらを順番にコンマ区切りのリストで指定します。ホスト・アプリケーションは、リストに表示されている順序でライブラリーをロードします。 *ヒント:* %b を使用し、ボード・インストレーション・ディレクトリーを参照することができます。
linkflags	ボードで使用可能な MMD レイヤーへのリンクに必要なリンカーフラグを指定する文字列です。 *ヒント:* %a を使用して SDK インストレーション・ディレクトリーを、%b を使用してボード・インストレーション・ディレクトリーを参照にすることができます。
linklibs	ボードで使用可能な MMD レイヤーの使用に対して SDK がリンクする必要があるライブラリーを指定する文字列です。 *注意:* ライブラリーは MMD レイヤーですべてのデバイスで必要なため、このフィールドには SDK で使用可能な alterahalmmd ライブラリーを含みます。
utilbindir	SDK がユーティリティー実行可能ファイル (install、uninstall、program、diagnose および flash) を検出するディレクトリーです。 *ヒント:* %a を使用して SDK インストール・ディレクトリーを、%b を使用してボード・インストレーション・ディレクトリーを参照することができます。

board_env.xml ファイルは、次の例に類似します。

```
<?xml version="1.0"?>
<board_env version="16.1" name="a10_ref">
  <hardware dir="hardware" default="a10gx"></hardware>
  <platform name="linux64">
    <mmdlib>%b/linux64/lib/libaltera_a10_ref_mmd.so</mmdlib>
    <linkflags>-L%b/linux64/lib</linkflags>
    <linklibs>-laltera_a10_ref_mmd</linklibs>
    <utilbindir>%b/linux64/libexec</utilbindir>
  </platform>
```

```
  <platform name="windows64">
    <mmdlib>%b/windows64/bin/altera_a10_ref_mmd.dll</mmdlib>
    <linkflags>/libpath:%b/windows64/lib</linkflags>
    <linklibs>altera_a10_ref_mmd.lib</linklibs>
    <utilbindir>%b/windows64/libexec</utilbindir>
  </platform>
</board_env>
```

関連情報

- アルテラ SDK for OpenCL の前提条件
- インテル FPGA RTE for OpenCL の前提条件

1.4.2.1.1 board_env.xml ファイルのテスト

`board_env.xml` ファイルを生成した後、ボード・インストレーション・ディレクトリー内のファイルをテストし、インテル FPGA SDK for OpenCL オフライン・コンパイラーがボード・インストレーションを認識するようにします。

1. `board_env.xml` ファイルが存在するカスタム・プラットフォーム・サブディレクトリーを指すように、環境変数 *AOCL_BOARD_TOOLKIT_ROOT* を設定します。
2. コマンドプロンプトで、`aocl board-xml-test` コマンドを呼び出し、インテル FPGA SDK for OpenCL が正しいフィールド値を検索できるかを確認します。

   SDK は、下に類似した出力を生成します。

```
board-path       = <path_to_a10_ref>
board-version    = 16.1
board-name       = a10_ref
board-default    = a10gx_es3
board-hw-path    = <path_to_a10_ref>/hardware/a10_ref
board-link-flags = /libpath:<path_to_a10_ref>/windows64/lib
board-libs       = alterahalmmd.lib altera_a10_ref_mmd.lib
board-util-bin   = <path_to_a10_ref>/windows64/libexec
board-mmdlib     = <path_to_a10_ref>/windows64/bin/altera_a10_ref_mmd.dll
```

3. `aoc --list-boards` コマンドを呼び出し、オフライン・コンパイラーがカスタム・プラットフォームでボード・バリエーションを識別して通知できるかを確認します。
   例えば、カスタム・プラットフォームに 2 つの FPGA ボードが含まれている場合、SDK は次のいずれかに類似した出力を生成します。

```
Board list:
   <board_name_1><board_name_2>
```

最後のボード・インストレーション・テストは、オフライン・コンパイラーを使用してボードのデザインを生成する際に行われます。

関連情報

- 13 ページの board_env.xml ファイルの作成
- 23 ページの ハードウェア・デザインのテスト

1.4.2.2 board_spec.xml ファイルの作成

`board_spec.xml`XML ファイルには、ハードウェア・システムを インテル FPGA SDK for OpenCL に記述するために必要なメタデータが含まれています。

board_spec.xml ファイルに含む必要のある型について詳しくは、*board_spec.xml ファイルの XML エレメント、属性、およびパラメーター*の項を参照してください。board_spec.xml テンプレートは、カスタム・プラットフォーム・ツールキットの *ALTERAOCLSDKROOT*/board/custom_platform_toolkit/board_package/hardware/template ディレクトリーで使用可能です。

1. 次の XML エレメントと属性を含めるために board_spec.xml ファイルを構造化します。

表 3. board_spec.xml ファイルで指定した XML エレメントと属性

エレメント	属性
board	version、name
device	device_model、used_resources
global_mem	name、max_bandwidth、interleaved_bytes、config_addr、[default]、interface
host	kernel_config
[channels]	interface
interfaces	interface、kernel_clk_reset
compile	project、revision、qsys_file、generic_kernel、generate_cmd、synthesize_cmd、auto_migrate

2. board エレメントでは、ボードのバージョンとアクセラレーター・ボードの名前を指定します。ボード名は、board_spec.xml ファイルが存在するディレクトリー名と一致している必要があります。

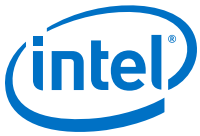
   重要: ボードバージョンは、カスタム・プラットフォームの開発に使用する SDK のバージョンと一致している必要があります。

   注意: ボード名は、文字、数字、アンダースコア (_)、ハイフン (-)、もしくはピリオド (.) のみでの組み合わせ (例 : a10_ref) でなければなりません。

3. device エレメントでは、次の手順を実行してデバイス・モデル・ファイル名を指定します。
   a. *ALTERAOCLSDKROOT* が SDK インストーレーションへのパスを参照する *ALTERAOCLSDKROOT*/share/models/dm ディレクトリーに移動します。ディレクトリーは、アクセラレーター・ボード上で使用可能な FPGA リソースを記述するデバイス・モデル・ファイルのリストを含んでいます。
   b. デバイスが dm ディレクトリーにリストされている場合、device_model 属性にデバイス・モデル・ファイルの名前を指定します。手順 4 に進みます。

      例 : device_model="10ax115s2f45i2sges_dm.xml"

   c. デバイスが dm ディレクトリーにリストされていない場合、またはボードにデバイスモデルがない FPGA を使用している場合は、手順 d~g を実行して新しいデバイスモデルを作成します。
   d. *ALTERAOCLSDKROOT*/share/models/dm ディレクトリー (例 : 10ax115s2f45e2lg_dm.xml) からデバイスモデルをコピーします。
   e. board_spec.xml ファイルが存在するファイルが存在するカスタム・プラットフォーム・サブディレクトリーにデバイスモデルのコピーを保存します。
   f. ファイル名を変更し、ボードが使用するパーツを記述するために値を変更します。



g. board_spec.xml ファイルで、ファイル名とともに device エレメントの device_model 属性を更新します。

4. device エレメントでは、used_resources 属性でパラメーターを指定して、OpenCL カーネルが存在しないボードデザインが消費する FPGA リソースを記述します。

   デザインにすべてのボードロジック周辺に定義されたパーティションが含まれている場合、Fitter レポートの Partition Statistics セクションからデータを抽出することができます。

5. 各グローバルメモリーのタイプで、次の情報を指定します。
   a. メモリータイプの名前。
   b. 結合された最大グローバルメモリーの帯域幅。

      メモリーのデータシートから帯域幅の値が算出できます。
   c. インテル FPGA SDK for OpenCL オフライン・コンパイラーがメモリーバンクをまたいでインターリーブするデータのサイズ。

      注意: interleaved_bytes = burst_size x width_bytes
   d. ホモジニアス・メモリー・システムを使用している場合は手順 e に進みます。ヘテロジニアス・メモリー・システムを使用している場合は、グローバルメモリー・タイプごとに ACL Mem Organization Control Qsys コンポーネント (mem_org_mode) のベースアドレスで config_addr 属性を指定します。
   e. グローバル・メモリー・タイプをデフォルト設定に選択する場合は、オプションの default 属性に値 1 を割り当てます。

      この属性を含まない場合は、board_spec.xml ファイルで定義されている最初のメモリーがデフォルトメモリーになります。
   f. interface 属性にパラメーターを指定し、各メモリー・インターフェイスの特性を記述します。

6. host エレメントでは、kernel_config 属性にパラメーターを指定し、カーネルが存在する場所でオフセットを記述します。OpenCL Kernel Interface Qsys コンポーネントで、kernel_cra マスターの観点からオフセットの開始を決定します。

7. ボードが直接 OpenCL カーネルから I/O へのアクセスのためのチャネルを提供している場合は、すべてのチャネル・インターフェイスで channels エレメントを含めます。interface 属性にパラメーターを指定して各チャネル・インターフェイスの特性を記述します。

8. OpenCL カーネルに接続し OpenCL カーネルを制御するカーネル・インターフェイスを記述する interfaces エレメントを含めます。各インターフェイス型 (つまり、master、irq、および streamsource) の 1 つを含めます。
   a. interface 属性にパラメーターを指定して各カーネル・インターフェイスの特性を記述します。

      streamsource インターフェイス型では、スヌーブストリームが使用するクロック名で clock 属性も指定します。通常、このクロックはカーネルクロックです。

      重要: streamsource カーネル・インターフェイスで指定されたスヌープ・インターフェイス (acl_internal_snoop) の幅を更新します。幅の更新は、board_spec.xml の global_mem インターフェイス・エントリーが、デフォルトメモリーで対応する OpenCL Memory Bank Divider コンポーネントからの bank<*N*> Avalon メモリーマップド (Avalon-MM) マスターの特性と一致していることを確認します。

b. kernel_clk_reset 属性でパラメーターを指定し、抽出したカーネルクロックを含めてインターフェイスをカーネル・インターフェイスとしてリセットします。

9. compile エレメントを含んで Quartus Prime コンパイル、レジストレーション、および自動移行を制御するためにその属性を指定します。

下は、board_spec.xml ファイルの XML コード例です。

```
<?xml version="1.0"?>
<board version="16.1" name="a10gx">

  <compile project="top" revision="top" qsys_file="none" generic_kernel="1">
    <generate cmd="quartus_sh -t scripts/pre_flow_pr.tcl"/>
    <synthesize cmd="quartus_cdb -t import_compile.tcl"/>
    <auto_migrate platform_type="a10_ref" >
      <include fixes=""/>
      <exclude fixes=""/>
    </auto_migrate>
  </compile>

  <device device_model="10ax115s2f45i2sges_dm.xml">
    <used_resources>
      <alms num="33120"/> <!-- Total ALMs - ALMs available to
kernel_system_inst -->
      <ffs num="132480"/>
      <dsps num="0"/>
      <rams num="333"/>
    </used_resources>
  </device>

  <!-- DDR4-2400 -->
  <global_mem name="DDR" max_bandwidth="19200" interleaved_bytes="1024"
config_addr="0x018">
    <interface name="board" port="kernel_mem0" type="slave" width="512"
maxburst="16" address="0x00000000" size="0x80000000" latency="240"
addpipe="1"/>
  </global_mem>

  <host>
    <kernel_config start="0x00000000" size="0x0100000"/>
  </host>

  <interfaces>
    <interface name="board" port="kernel_cra" type="master" width="64"
misc="0"/>
    <interface name="board" port="kernel_irq" type="irq" width="1"/>
    <interface name="board" port="acl_internal_snoop" type="streamsource"
enable="SNOOPENABLE" width="31" clock="board.kernel_clk"/>
    <kernel_clk_reset clk="board.kernel_clk" clk2x="board.kernel_clk2x"
reset="board.kernel_reset"/>
  </interfaces>

</board>
```

関連情報

35 ページの Board_spec.xml ファイル内の XML エレメント、属性、およびパラメーター

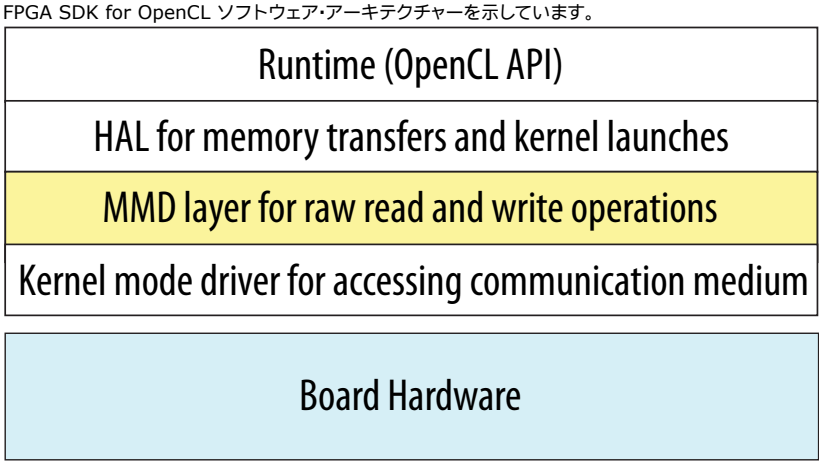
1.4.3 MMD ライブラリーの作成

カスタム・プラットフォームは、アクセラレーター・ボードとの通信に必要な MMD レイヤーが必要です。

任意の媒体上のアクセラレーター・ボードと通信するには、オープン、リード、ライト、およびクローズなどといった I/O ファイルに似たソフトウェア・インターフェイスを実装する必要があります。実装結果は、OpenCL ホスト・アプリケーションがターゲットボードの MMD レイヤーに対してリンクできるようにするリンカー引数のセットです。通信には、MMD レイヤーを完全に実装する DLL (ダイナミック・リンク・ライブラリー) も必要です。

図 -2: インテル FPGA SDK for OpenCL ソフトウェア・アーキテクチャー

この図は、ランタイム、HAL (ハードウェア・アブストラクション・レイヤー)、MMD レイヤー、およびカーネル・モード・ドライバーの 4 つの インテル FPGA SDK for OpenCL ソフトウェア・アーキテクチャーを示しています。



次のタスクは、PCI Express® (PCIe®)で使用する MMD ライブラリーの作成手順についての概要です。

1. MMD レイヤーを実装する新しいライブラリー・ファイルを次のように命名します。

   *<board_vendor_name>*_*<board_family_name>*[_*<unique_string>*]_mmd.*<a|so|lib|dll>*

   位置 :

   - *<board_vendor_name>*は、アクセラレーターボードを担うエンティティーです。
   - *<board_family_name>*は、ライブラリーがサポートするボードファミリー名です。
   - *<unique_string>*は、作成の指示です。インテル は、リビジョンやインターフェイス型などの情報を含めることを推奨しています。
   - *<a|so|lib|dll>*は、ファイル拡張子です。アーカイブファイル (.a)、共有オブジェクト・ファイル (.so)、ライブラリー・ファイル (.lib)、ダイナミック・リンク・ライブラリー・ファイル (.dll) のいずれかになります。

   ライブラリー・ファイル名の例 : altera_svdevkit_pcierev1_mmd.so
2. MMD レイヤーのオペレーティング・システム固有の実装に、 *ALTERAOCLSDKROOT*/board/custom_platform_toolkit/mmd/aocl_mmd.h ヘッダーファイルを含めます。

   *MMD API の概要*の項と aocl_mmd.h ファイルでは、MMD アプリケーション・プログラミング・インターフェイス (API) の概要、それらの引数および戻り値の詳細を説明しています。
3. カスタム・プラットフォームで MMD レイヤーを実装し、C/C++ ライブラリーにコンパイルします。

ファンクショナル MMD ライブラリーのソースコード例は、Arria10 GX FPGA Development Kit Reference Platform の *<path_to_a10_ref>*/source/host/mmd ディレクトリーで使用可能です。特に、acl_pcie.cpp ファイルには、aocl_mmd.h ファイルで定義されている API 関数が実装されています。

SDK ユーザーがランタイムで特定のライブラリーをロードする必要がある場合、オペレーティング・システムが検出できるディレクトリーにライブラリーを設置します。SDK ユーザーに、ランタイムに *LD_LIBRARY_PATH* (Linux の場合) または *PATH* (Windows の場合) 環境変数にライブラリー・パスを追加するよう指示します。

4. MMD レイヤーとのリンクに必要なライブラリー・フラグの指定により、board_env.xml ファイルの mmdlib と linkflags エレメントを変更します。

関連情報

42 ページの MMD API の概要

1.4.3.1 カーネル・パワーアップの状態

OpenCL カーネルは、システムに電源投入後、または FPGA の再プログラム後は不明の状態です。その結果、MMD レイヤーはこの間にカーネルからのどの割り込みも無効、または無応答になります。カーネルは aocl_mmd_set_interrupt_handler が呼び出された後でのみ既知の状態になります。したがって、ハンドラーが MMD レイヤーで使用可能になった後にのみ、カーネルからの割り込みを有効にします。

シングル・ホスト・アプリケーションの一般的なコールは次の通りです。

1. get_offline_info
2. open
3. get_info
4. set_status_handler
5. set_interrupt_handler
6. get_info /read/write/copy/yield
7. Close

1.4.4 Altera Client Driver のセットアップ

インテル FPGA SDK for OpenCL は、Altera Client Driver (ACD) カスタム拡張をサポートしています。ACD は、SDK がホストのランタイム時にカスタム・プラットフォーム・ライブラリーを自動的に検索してロードすることができます。

注意: SDK ユーザーが ACD を使用するには、board_env.xml ファイルの linklibs エレメントから MMD ライブラリーを削除しなければなりません。

Windows での Custom Platform ACD の列挙

レジストリー・キー **HKEY_LOCAL_MACHINE\SOFTWARE\Altera\OpenCL\Boards** にカスタム・プラットフォーム・ライブラリーを指定します。値の名前をライブラリーのパスになるように指定し、データを 0 に設定された **DWORD** に指定します。

例：

[HKEY_LOCAL_MACHINE\SOFTWARE\Altera\OpenCL\Boards] "c:\board_vendor a\my_board_mmd.dll"=dword:00000000

Windows 上でカスタム・プラットフォーム ACD を列挙するため、ACD Loader は レジストリー・キー **HKEY_LOCAL_MACHINE\SOFTWARE\Altera\OpenCL\Boards** で値をスキャンします。このキーでの各値では、名前は DLL へのパスを指定し、データは **dword** です。**dword** データが 0 の場合は、Installable Client Driver (ICD) Loader は対応する DLL を開こうとします。DLL が MMD ライブラリーにある場合は、SDK はそのライブラリーに関連付けられているボードを開こうとします。

この場合、ACD はライブラリー `c:\board_vendor a\my_board_mmd.dll` を開きます。

レジストリー・キーで複数のライブラリーを指定する場合、Loader はキーに表示される順にライブラリーをロードします。カスタム・プラットフォームで使用可能なライブラリー間で順序の依存関係がある場合は、レジストリー・キーに適切にライブラリーをリストするようにしてください。

Linux での Custom Platform ACD の列挙

`.acd` ファイルに Custom Platform ライブラリーの絶対パスを入力します。`.acd` ファイルを`/opt/Altera/OpenCL/Boards/`ディレクトリーに保存します。

Linux 上で Custom Platform ACD を列挙するため、ACD Loader は `/opt/Altera/OpenCL/Boards/` パスの拡張子 `.acd` でファイルをスキャンします。ACD Loader は、このパスの各 `.acd` ファイルをテキストファイルとして開きます。各 `.acd` ファイルは、カスタム・プラットフォームの各行につき 1 つのライブラリーですべてのライブラリーへの絶対パスを含んでいる必要があります。ICD Loader は各ライブラリーを開こうとします。ライブラリーが MMD ライブラリーの場合は、SDK はそのライブラリーに関連付けられているボードを開こうとします。

例として、`/opt/Altera/OpenCL/Boards/PlatformA.acd` ファイルを考慮します。ライン`/opt/PlatformA/libPlatformA_mmd.so` が含まれている場合、ACD Loader はライブラリー`/opt/PlatformA/libPlatformA_mmd.so` をロードします。

`.acd` ファイルで複数のライブラリーを指定する場合、Loader はファイルに表示される順序でライブラリーをロードします。カスタム・プラットフォームで使用可能なライブラリー間で順序の依存関係がある場合は、`.acd` ファイルに適切にライブラリーをリストするようにしてください。

インテル FPGA SDK for OpenCL のバージョン 16.1 以降、ACD は SDK ユーザーが異なるカスタム・プラットフォームから複数のヘテロジニアス・ボードをロードし、シングル・ホスト・アプリケーションでそれらを一緒に使用できます。ただし、この機能はバージョン 16.1 より前にリリースされた従来のカスタム・プラットフォームのサポートに限られます。

SDK ユーザーがこれらのホスト・アプリケーションを ICD と ACD にリンクする方法について詳しくは、*インテル FPGA SDK for OpenCL Programming Guide* の *Linking Your Host Application to the Khronos ICD Loader Library* の項を参照してください。

関連情報

Linking Your Host Application to the Khronos ICD Loader Library

1.4.5 インテル FPGA SDK for OpenCL ユーティリティー・サポートの提供

インテル FPGA SDK for OpenCL を使用して開発する各カスタム・プラットフォームは、SDK ユーティリティーのセットをサポートしています。ユーザーはこれらのユーティリティーで SDK を介してアクセラレーター・ボードを管理することができます。

新しいカスタム・プラットフォームを作成する場合は、SDK ユーティリティーの実行を作成し、カスタム・プラットフォームの `utilbindir` ディレクトリーに保存するために、次のタスクを実行します。

ヒント: Arria 10 GX FPGA Development Kit Reference Platform の *<path_to_a10_ref>*`/source/util` ディレクトリー内で、`reprogram` と `flash` サブディレクトリーにある `program` と `flash` ユーティリティーのそれぞれのソースコードを検索することができます。`install` と `uninstall` ユーティリティーのスクリプトは、 *<path_to_a10_ref>*`/`*<OS_platform>*`/libexec` ディレクトリーにあります。

Arria 10 GX FPGA Development Kit Reference Platform 内の *<path_to_a10_ref>*`/source/util/diagnostic` ディレクトリーにある `diagnose` ユーティリティーのソースコードを検索することができます。

1. MMD レイヤーを介してボードと通信するために、必要なドライバーを使用して現在のホスト・コンピューターを設定する `install` ユーティリティー実行可能を作成します。`install` ユーティリティーには引数がありません。

   例えば、PCIe ベースの MMD では、ホスト・オペレーション・システム内に PCIe ドライバーのインストールが必要な場合があります。

   実行可能コール：`aocl install`
2. ボードとの通信で使用する現在のホスト・コンピューターのドライバー (PCIe ドライバー など) を削除する `uninstall` ユーティリティー実行可能を作成します。`uninstall` ユーティリティーには引数はありません。

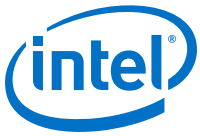
   実行可能コール：`aocl uninstall`
3. ボードの整合性と MMD レイヤーの機能を確認する `diagnose` ユーティリティー実行可能を作成します。

   `diagnose` ユーティリティーは、次の内部コールモードをサポートしています。

コールモード	説明
`-probe`	カスタム・プラットフォームで使用可能なデバイスをプリントします。特定のハードウェア・コンフィグレーションでは、このユーティリティーはデバイスを同じ順序でリストし、各デバイスは毎回同じ識別文字列に関連付けられます。
`probe[ ]` *<device_name>*	指定したデバイスを照会し、デバイスに関する統計の情報を出力します。
<device_name> *<device name>*は FPGA デバイスに対応する文字列です。	指定したデバイスの完全な診断テストを実行します。 ユーティリティーは、出力としてメッセージ `DIAGNOSTIC_PASSED` を生成します。それ以外の場合は、ユーティリティーはメッセージ `DIAGNOSTIC_FALIED` を生成します。

   ユーザーが引数なしで `diagnose` ユーティリティー・コマンドを呼び出すと、カスタム・プラットフォームにデバイスを照会し、デバイスのリストに割り当てられた有効な*<device_name>*文字列のリストを提供します。

   引数なしの実行可能コール：`aocl diagnose`



<*device_name*> 引数で diagnose ユーティリティー・コマンドを呼び出すと、ユーティリティーはボードに対して診断テストを実行します。ユーザーは、ボードにカスタム・プラットフォームに関連付けられた物理デバイス名とは異なる論理デバイス名を付けることができます。aocl ユーティリティーは単純にユーザー側の物理デバイス名をカスタム・プラットフォーム側のデバイス名に変更します。診断テストが正常に実行されると、ユーティリティーは出力としてメッセージ DIAGNOSTIC_PASSED を生成します。その他の場合は、ユーティリティーはメッセージ DIAGNOSTIC_FALIED を生成します。

引数ありの実行可能コール：aocl diagnose <*device_name*> .

4. fpga.bin ファイルと インテル FPGA SDK for OpenCL オフライン・コンパイラー実行可能ファイル (.aocx) を受信し、そのデザインを FPGA に構成する program ユーティリティー実行可能を作成します。FPGA プログラミングの主な手法はホストと MMD を介する方法ですが、ホストシステムがないユーザーやオフラインの再プログラミングを実行するユーザーもこのユーティリティーが使用できるようになります。

   Program ユーティリティー・コマンドは、引数として<*device_name*>、fpga.bin、および <*kernel_filename*>.aocx を取ります。ユーザーがコマンドを呼び出すと、SDK は fpga.bin ファイルを抽出し、program ユーティリティーに渡します。

   重要: インテル は、program ユーティリティーが MMD レイヤーに実装された aocl_mmd_reprogram 関数にリンクし、コールすることを強く推奨しています。詳しい情報はリファレンス・マニュアルの *aocl_mmd_reprogram と再プログラミングのサポート*の項を参照してください。

   実行可能コール：aocl program <*device_name*> <*kernel_filename*>.aocx です。<*device name*>は対応する FPGA デバイスの acl ナンバーです。

5. fpga.bin ファイルを受信し、ボード上でフラッシュメモリー内にデザインをプログラミングする flash ユーティリティー実行可能を作成します。flash ユーティリティー・コマンドは、引数として <*device_name*>と.aocx ファイル名を取ります。ユーザーがコマンドを呼び出すと、SDK は fpga.bin ファイルを抽出し、そのファイルを flash ユーティリティーに渡します。

   実行可能コール：aocl flash <*device_name*> <*kernel_filename*>.aocx で、<*device name*>は対応する FPGA デバイスの acl ナンバーです。

ユーザーがユーティリティー・コマンドを呼び出すと、ユーティリティーは現在のカスタム・プラットフォームの board_env.xml ファイルをプローブし、utilbindir ディレクトリー内の <*utility_executable*> ファイルを実行します。

関連情報

- 13 ページの board_env.xml ファイルの作成
- 52 ページの 再プログラミングのサポート
- 51 ページの aocl_mmd_reprogram

1.4.6 ハードウェア・デザインのテスト

ソフトウェア・ユーティリティーおよび MMD レイヤーの作成後、ハードウェア・デザインはタイミング・クロージャーを達成し、デザインをテストします。

ハードウェア・デザインをテストするには、次のタスクを実行します。

1. *ALTERAOCLSDKROOT*/board/custom_platform_toolkit/tests/boardtest ディレクトリー内の boardtest.cl OpenCL カーネルに移動します。

ALTERAOCLSDKROOT は インテル FPGA SDK for OpenCL インストレーションの位置を参照します。

2. `aoc --no-interleaving default boardtest.cl` コマンドを呼び出して インテル FPGA SDK for OpenCL オフライン・コンパイラー実行可能ファイル (`.aocx`) を生成するために、カーネルをコンパイルします。
3. `aocl program acl0 boardtest.aocx` コマンドを呼び出して、アクセラレーター・ボードをプログラムします。
4. `aocl compile-config` と `aocl link-config` のコマンドを呼び出します。正常にコンパイルとリンクのために必要なフラグが MMD レイヤーに含まれていることを確認します。
5. `boardtest` ホスト・アプリケーションをビルドします。
   - Windows システムでは、`make -f Makefile.windows` コマンドを呼び出すか、Microsoft Visual Studio を使用する場合があります。

     `make` コマンドを呼び出す場合は、*ALTERAOCLSDKROOT*`/board/custom_platform_toolkit/tests/boardtest` ディレクトリーにある `Makefile.windows` ファイルを使用します。

     Microsoft Visual Studio でホスト・アプリケーションをビルドする場合は、`boardtest.sln` と `main.cpp` ファイルは *ALTERAOCLSDKROOT*`/board/custom_platform_toolkit/tests/boardtest/host` ディレクトリー内にあります。
   - Lunux システムでは、`make -f Makefile.linux` コマンドを呼び出します。

     `Makefile.linux` ファイルは *ALTERAOCLSDKROOT*`/board/custom_platform_toolkit/tests/boardtest` ディレクトリー内にあります。
6. `boardtest` 実行可能を実行します。

注意: ハードウェア・デザインで一貫したパフォーマンスが得られるように、`boardtest.cl` に加えて複数の OpenCL カーネルを使用してテストする必要があります。

インテル FPGA Preferred Board として証明を得るには、複数のボードにわたる厳密なテストが必要です。具体的には、すべてのカスタム・プラットフォームのテストを夜間通して行い、複数のボードで SDK のデザイン例を実行する必要があります。カスタム・プラットフォーム内のボード・バリエーションは、すべてのテストプロセスを経ていなければなりません。

1.5 Generating the Rapid Prototyping Library インテル FPGA SDK for OpenCL Custom Platform Rapid Prototyping Support (Early Access Program)

Altera SDK for OpenCL [1] (AOCL [2]), you must generate a library of precompiled FPGA designs. Create the Rapid Prototyping library after your initial board design partition achieves timing closure and you finalize the design. The Rapid Prototyping feature enables AOCL users to compile and test a limited set of OpenCL kernels without compiling to hardware.

---

[1] OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission of the Khronos Group™.

The following files and components are necessary for generating a Rapid Prototyping library:

1. Timing-closed board design partition.
2. Prototyping templates package available with the Custom Platform Toolkit in the *ALTERAOCLSDKROOT*/board/custom_platform_toolkit directory, where *ALTERAOCLSDKROOT* points to the location of the AOCL installation). The directory structure is as follows:
   - prototyping_templates/board_package—Contains the .txt files that you transfer to the board directory of the Custom Platform.
   - prototyping_templates/kernels—Contains the .cl files that you use to generate the library.

To generate the Rapid Prototyping library, perform the following tasks:

1. Create the Rapid Prototyping board subdirectory within the same directory as the board, and name it <*board_name*>_vfabric.

   For example, if the timing-closed board design is located in <*path_to_s5_net*>/hardware/s5_net, the Rapid Prototyping board directory is <*path_to_s5_net*>/hardware/s5_net_vfabric.
2. Copy all the files in the prototyping_templates/board_package directory of the Custom Platform Toolkit into the AOCL Rapid Prototyping board directory created in Step 1.
3. Compile each .cl file in the prototyping_templates/kernels directory of the Custom Platform Toolkit by invoking the following command:

   ```
   aoc -march=prototype --create-template --reuse-existing-templates [--profile] --board <board_name> <kernel_filename>.cl
   ```

   This command compiles a Rapid Prototyping design for each of the kernel files in the prototyping_templates/kernels directory. It also creates a <*kernel_filename*> subdirectory that contains necessary intermediate files. For example, compiling var1.cl generates a Rapid Prototyping design from the var1.txt file in the Rapid Prototyping board directory.

   You must compile some of the .cl kernel source files with the --profile インテル FPGA SDK for OpenCL オフライン・コンパイラー (AOC) option to instrument the Verilog code with performance counters (see table below). After the compilation completes successfully, copy each output file into the Rapid Prototyping board directory and rename it in the following manner:
   - Rename fpga.bin to <*kernel_filename*>.fpga.bin
   - Rename acl_quartus_report.txt to <*kernel_filename*>.acl_quartus_report.txt

(2) The インテル FPGA SDK for OpenCL is based on a publishedTo support the Rapid Prototyping feature of the Khronos Specification, and has passed the Khronos Conformance Testing Process. Current conformance status can be found at www.khronos.org/conformance.

表 4. Rapid Prototyping Kernel Source Files and Corresponding Names of Output Binary and Report Files

Kernel Source File	Include --profile flag in aoc command?	Files to Copy from the prototyping_templates/kernels directory	Destination Filenames
var1.cl	Yes	fpga.bin, acl_quartus_report.txt	var1.fpga.bin, var1.acl_quartus_report.txt
var2.cl	Yes	fpga.bin, acl_quartus_report.txt	var2.fpga.bin, var2.acl_quartus_report.txt
var3.cl	No	fpga.bin, acl_quartus_report.txt	var3.fpga.bin, var3.acl_quartus_report.txt
var4.cl	Yes	fpga.bin, acl_quartus_report.txt	var4.fpga.bin, var4.acl_quartus_report.txt
var5.cl	Yes	fpga.bin, acl_quartus_report.txt	var5.fpga.bin, var5.acl_quartus_report.txt
var6.cl	No	fpga.bin, acl_quartus_report.txt	var6.fpga.bin, var6.acl_quartus_report.txt
var7.cl	No	fpga.bin, acl_quartus_report.txt	var7.fpga.bin, var7.acl_quartus_report.txt
var8.cl	No	fpga.bin, acl_quartus_report.txt	var8.fpga.bin, var8.acl_quartus_report.txt
var9.cl	No	fpga.bin, acl_quartus_report.txt	var9.fpga.bin, var9.acl_quartus_report.txt
var10.cl	Yes	fpga.bin, acl_quartus_report.txt	var10.fpga.bin, var10.acl_quartus_report.txt
var11.cl	Yes	fpga.bin, acl_quartus_report.txt	var11.fpga.bin, var11.acl_quartus_report.txt

4. After you compile the `.cl` files successfully, copy the `sys_description.txt` file from any *<kernel_filename>* subdirectory and paste it into the Rapid Prototyping board directory.

   `sys_description.txt` ファイルは、生成されたすべてのテンプレートに共通です。

5. Depending on the size of the board interface partition, some templates that you generate from the kernel files in the `prototyping_templates/kernels` directory might not fit on the board. In that case, delete the corresponding `.txt` file from the Rapid Prototyping board directory.
6. Rename files as necessary to ensure that you number the templates continuously.

   For example, if the `var9.cl` file fails to compile but `var10.cl` compiles successfully, after you delete `var9.txt` from the Rapid Prototyping board directory, you must rename the following files to begin with var9 instead of var10:

- var10.txt
- var10.fpga.bin
- var10.acl_quartus_report.txt

7. Update the num_templates.txt file to the number of templates that compile successfully.

After you complete all the compilations, verify that the <*board_name*>_vfabric directory contains the following files:

表 5. Files in the <board_name>_vfabric Directory After Compilation Completes

Filename	Description
num_templates.txt	Contains the number of templates that compile successfully.
sys_description.txt	A common file that all generated templates use. If the file is missing, copy the sys_description.txt file from any <*kernel_filename*> subdirectory that the AOC creates during compilation to the <*board_name*>_vfabric directory.
var<*N*>.txt	Contains the specification of the generated template. There is one such file for each template that compiles successfully.
var<*N*>.fpga.bin	The precompiled fpga.bin file that contains the generated template. There is one such file for each template that compiles successfully.
var<*N*>.acl_quartus_report.txt	The compilation information for the generated template. There is one such file for each template that completes successfully.

1.6 インテル FPGA SDK for OpenCL Preferred Board ステータスへの適用

カスタム・プラットフォームとサポートされる FPGA ボードを インテル FPGA SDK for OpenCL Preferred Board Partner Program に登録すると、Quartus Prime デザインスイート のバージョン間で継続的にされる内部テストの恩恵を受けることができます。インテル がテストしたカスタム・プラットフォームとボードは、後の Quartus Prime デザインスイート のバージョンとの上位向互換性が高くなります。

カスタム・プラットフォームとサポートされる FPGA ボードで インテル FPGA SDK for OpenCL Preferred Board ステータスを得るには、以下のデータを生成し、インテル にそれらを提出しなければなりません。

1. aocl board-xml-test コマンドの呼び出しからの出力。
2. aoc --list-boards コマンドの呼び出しからの出力。
3. ホストコンパイル、ホスト実行、およびすべての Quartus Prime リポートファイル (.rpt) からの出力。また、カスタム・プラットフォームの各ボード場合は、次のテストからの acl_quartus_report.txt ファイルがあります。
   a. すべてのテストは、*ALTERAOCLSDKROOT* が インテル FPGA SDK for OpenCL インストレーションの位置を参照する *ALTERAOCLSDKROOT*/board/custom_platform_toolkit/tests ディレクトリー内に含まれています。
   b. アルテラのウェブサイトの OpenCL デザイン例のページに従って、次の例をコンパイルします。

i. Vector Addition
ii. Matrix Multiplication
iii. FFT (1D)
iv. FFT
v. Sobel Filter
vi. Finite Difference Computation (3D)

4. カスタム・プラットフォームの各ボードには、次の要約があります。
   a. カスタム・プラットフォーム・ツールキット (`/tests/boardtest`) の boardtest テストにより報告された `HOST-TO-MEMORY BANDWIDTH`
   b. boardtest テストにより報告された `KERNEL-TO-MEMORY BANDWIDTH`
   c. カスタム・プラットフォーム・ツールキット (`/tests/swapper`) の swapper テストにより通知されたスワップアンド実行の `Throughput`
   d. カスタム・プラットフォーム・ツールキット (*`ALTERAOCLSDKROOT`*`/board/custom_platform_toolkit/tests/blank`) の blank テストから `acl_quartus_report.txt` ファイルにより報告された `Actual clock freq`

   重要: ローカル配線リソースの消費を抑えるためにグローバル配線を使用します。グローバル配線の使用は、タイミング制約を満たしてカーネルの性能 (Fmax) の向上に役立つため、必要不可欠です。グローバル配線またはリージョナル配線は、5000 以上のファンアウトを有するすべてのネット、または、カーネルクロック、クロックとリセットの 2 つに対して使用します。Compilation Report の Fitter セクションの下にある Resource サブセクションの Non-Global High Fan-Out Signals レポートを確認してください。

5. 必要なボード数をインハウス・レグレッション・テストのために インテル に提出します。レグレッション・テストは、カスタム・プラットフォームがサポートする各動作システム上の各ボードバリアントに対して、革新的な実験でテストを行います。ボードを提出する前に次の手順のテストを行ってください。
   a. ボードを物理マシンにインストールします。
   b. マシンをブートし、`aocl install` ユーティリティー・コマンドを呼び出します。
   c. `aocl diagnose` コマンドを呼び出します。
   d. SDK のテストプログラムを実行します。テスターは、`program` ユーティリティーの関数を明確にするために、`aocl program` *`<device_name>`* *`<kernel_filename>`*`.cl` コマンドも呼び出します。

関連情報

インテル FPGA のウェブサイトの OpenCL のデザイン例ページ

1.7 出荷時の推奨

インテル 検証済みのボードを インテル FPGA SDK for OpenCL ユーザーに出荷する前に、hello_world OpenCL デザイン例でボードのフラッシュメモリーをプログラミングします。`hello_world.aocx` ハードウェア・コンフィグレーション・ファイルを使用したボードのフラッシュメモリーのプログラミングは、SDK ユーザーがボードに電源投入して作業中のカーネルを観察することを可能にします。

アルテラのウェブサイトの OpenCL デザイン例のページから hello_world OpenCL デザイン例をダウンロードします。

より詳しい情報は、hello_world OpenCL デザイン例で使用可能な README.txt ファイルと *インテル FPGA SDK for OpenCL スタートガイドの FPGA のフラッシュメモリーのプログラミング*の項を参照してください。

関連情報

- アルテラのウェブサイトの OpenCL のデザイン例ページ
- Windows 用 FPGA のフラッシュメモリーのプログラミング
- Linux 用 FPGA のフラッシュメモリーのプログラミング

1.8 改訂履歴

表 6. インテル FPGA SDK for OpenCL カスタム・プラットフォーム・ツールキット・ユーザーガイドの AOCL カスタム・プラットフォーム・デザインの章における改訂履歴

日付	バージョン	変更内容
2016 年 10 月	2016.10.31	• アルテラ SDK for OpenCL から「 インテル FPGA SDK for OpenCL 」へ変更。 • アルテラ・オフライン・コンパイラーから「 インテル FPGA SDK for OpenCL オフライン・コンパイラー」へ変更。 • board_env.xml および board_spec.xml ファイルのコード例を、Arria 10 GX FPGA Development Kit Reference Platform で対応するファイルの 16.1 バージョンに更新。 • 「*Altera Client Driver のセットアップ*」のテキストを更新。 • 「*ハードウェア・デザインのテスト*」で、Windows の手順 5 の make コマンドを make -f Makefile.windows に変更。
2016 年 5 月	2016.05.02	• 「*board_spec.xml ファイルの作成*」で、board_spec.xml の XML コード例を現在のバージョンに更新し、.xml ファイル例に沿う手順でエンベデッド例を更新。 • 項「*AOCL ユーティリティー・サポートの提供*」で、program ユーティリティーの実装の要求を更新。 • 「*Altera Client Driver のセットアップ*」で、.acd ファイルの Linux ディレクトリーを/opt/Altera/OpenCL_boards/から/opt/Altera/OpenCL/Boards/に変更。
2015 年 11 月	2015.11.02	• 表記を *Quartus II* から「*Quartus Prime*」へ変更。 • 表記を *Altera Complete Design Suite* から「 *Quartus Prime デザインスイート* 」へ変更。 • 項「*AOCL ユーティリティー・サポートの提供*」で、diagnose ユーティリティーのサポート要件を更新。 • 項「*board_env.xml ファイルの作成*」で、mmdlib XML エレメントを board_env.xml ファイルに含まれるエレメントのリストに追加。
2015 年 5 月	15.0.0	• 項「*Altera Client Driver のセットアップ*」を追加。

continued...

日付	バージョン	変更内容
2014 年 12 月	14.1.0	• カスタム・プラットフォーム・ツールキットが *ALTERAOCLSDKROOT*/board ディレクトリーで使用可能なことを明記。 • 項「*AOCL ユーティリティー・サポートの提供*」で、uninstall ユーティリティー実行可能ファイルを追記。 • board_env.xml と board_spec.xml ファイルのバージョン属性がカスタム・プラットフォームの開発に使用する AlteraComplete Design Suite と Altera SDK for OpenCL のバージョンに一致している必要があることを明記。 • 項「 *board_spec.xml ファイルの作成*」で、compile eXtensible Markup Language エレメントと board_spec.xml ファイルにそれらに関連する属性が含まれている説明を追記。 • 項「*前方互換性でのカスタム・プラットフォームの自動移行*」と「*自動移行のカスタマイズ*」で、カスタム・プラットフォームの自動移行の情報を追加。 • 項「*Rapid Prototyping Library の生成*」を削除。
2014 年 10 月	14.0.1	• 既存の資料を 2 つの章に再編成。
2014 年 6 月	14.0.0	• 初版

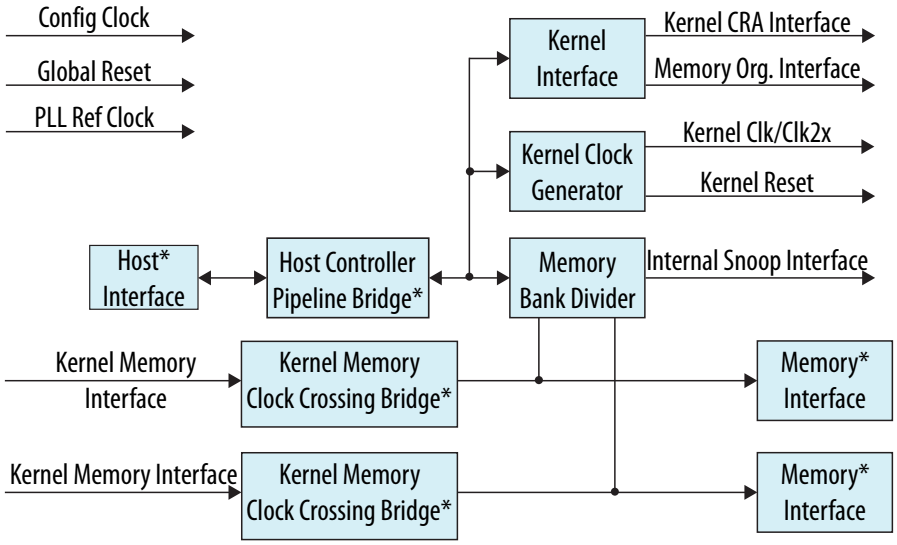
2 インテル FPGA SDK for OpenCL カスタム・プラットフォーム・ツールキット・リファレンス・マニュアル

インテル FPGA SDK for OpenCL カスタム・プラットフォーム・ツールキット・リファレンス・マニュアル の章では、カスタム・プラットフォームの実装を補助する補足情報を提供します。

2.1 ボード Qsys サブシステム

ボード・ハードウェアでデザインする際、すべての非カーネルロジックを含むトップレベルの Qsys システム (`system.qsys`) 内に Qsys サブシステムを作成するオプションがあります。ボード Qsys サブシステムは、新しいアクセラレーター・ボードにおいてメイン・デザイン・エントリー・ポイントです。OpenCL ホストのインスタンス化とグローバルメモリーのインターフェイスが行われる場所です。ボードデザインは、外部メモリーの最小 128 KB (キロバイト) を有します。すべての Avalon メモリーマップド (Avalon-MM) スレーブ・インターフェイス (ブロック RAM など) は、潜在的にメモリー・インターフェイスになり得ます。

下の図は、ボードシステムの実装を詳しく示しています。



注意: アスタリスク (*) で示されたブロックは、ボード Qsys サブシステムに追加する必要があるブロックです。

OpenCL ホスト通信インターフェイスとグローバルメモリー・インタフェースは、ボードシステムのメイン・コンポーネントです。MMD (メモリー・マップド・デバイス) レイヤーは、何らかの媒体を介して、この Qsys システムのインスタンス化された IP (Intellectual Property) コアと通信します。

例えば、MMD レイヤーは PCI Express (PCIe) ベースのホスト・インターフェイス上で実行され、ホスト・インターフェイスは FPGA 上の インテル PCIe エンドポイントから Avalon インターフェイスの要求を生成します。

ボード Qsys サブシステム内では、OpenCL カーネルで使用可能なグローバルメモリー・システムを定義することもできます。グローバルメモリー・システムは異なるタイプのメモリー・インターフェイスから構成されています。各メモリータイプは、1、2、4、または 8 バンクの物理メモリーで成っています。すべての特定のメモリータイプのバンクは、バイトで同じサイズで、同等のインターフェイスを有していなければなりません。万一、ストリーミング I/O がある場合は、Qsys システムに対応する IP もボード 含める必要があります。加えて、チャネル・インターフェイスを記述するために、`board_spec.xml` ファイルを更新しなければなりません。

2.1.1 インテル FPGA SDK for OpenCL 固有の Qsys システム・コンポーネント

ボードロジックの Qsys システムは、ホスト通信とグローバルメモリー・インターフェイスをインスタンス化する機能を実装するために必要な インテル FPGA SDK for OpenCL 固有のコンポーネントを含んでいます。

ボード Qsys システムは、OpenCL カーネルを制御するために Avalon-MM マスターをエクスポートする必要があります。また、カーネルがグローバルメモリー・インターフェイスとして使用する 1 つ以上の Avalon-MM スレーブポートもエクスポートしなければなりません。SDK の *ALTERAOCLSDKROOT*`/ip/board` ディレクトリーは、*ALTERAOCLSDKROOT* が SDK インストレーションの位置を参照する SDK 固有の Qsys システム・コンポーネントを持つライブラリーを含んでいます。これらのコンポーネントは、Avalon-MM インターフェイス、プログラマブル・バンクの編成、キャッシュ・スヌーピング、およびアルテラの保証されたタイミング・クロージャーのサポートなどの機能を実装するために必要です。

1. 32 ページの OpenCL Kernel Clock Generator
2. 33 ページの OpenCL Kernel Interface
3. 34 ページの OpenCL Memory Bank Divider

2.1.1.1 OpenCL Kernel Clock Generator

OpenCLKernel Clock Generator は、OpenCL カーネルで使用するクロック出力とクロック 2 つの出力を生成する Qsys コンポーネントです。Avalon-MM スレーブ・インターフェイスは、PLL (フェーズ・ロック・ループ) の再プログラミングとカーネルクロック・ステータスの情報に有効です。

表 7. OpenCL Kernel Clock Generator コンポーネントのパラメーター設定

パラメーター	説明
REF_CLK_RATE	カーネル PLL(つまり、`pll_refclk`) をドライブするリファレンス・クロックの周波数です。
KERNEL_TARGET_CLOCK_RATE	Quartus Prime ソフトウェアがコンパイル中に達成しようとする周波数です。デフォルト設定でこのパラメーターを保持してください。

表 8. OpenCL Kernel Clock Generator コンポーネントでの信号とポート

信号とポート。	説明
pll_refclk	カーネル PLL のためのリファレンス・クロックです。このクロックの周波数は、**REF_CLK_RATE** コンポーネント・パラメーターで指定する周波数と一致している必要があります。
clk	ホスト・コントロール・インターフェイスで使用されるクロックです。clk のクロックレートは Slow です。
reset	PLL とコントロール・ロジックをリセットするリセット信号です。PLL をリセットすると、カーネルクロックは一時的に無効になります。このリセット信号をシステムのパワーオンリセット信号に接続します。
CTRL	OpenCL ホスト・インターフェイスに接続し、OpenCL カーネルに基づく周波数を調整するために使用されるスレーブポートです。
kernel_clk kernel_clk2x	カーネルクロックと 2 倍速で動作する kernel_clk と kernel_clk の 2 つのバリアントです。kernel_clk2x 信号は、このインターフェイスから直接エクスポートされます。kernel_clk は内部 Qsys 接続を有するため、クロックソース・コンポーネントを使用してエクスポートします。また、クロックソースを使用して、カーネルリセットをエクスポートすることも可能です。加えて、追加したすべての I/O を除き、kernel_clk とともにボード Qsys システム・インタフェースですべてのロジックをクロックします。
kernel_pll_locked	(オプション) リファレンス・クロック上で PLL がロックされている場合、信号の値は 1 です。ホスト・インターフェイスは通常この信号を管理します。ただし、この信号はボード Qsys システムで使用可能になります。

2.1.1.2 OpenCL Kernel Interface

OpenCL Kernel Interface は、ホスト・インターフェイスが OpenCL カーネルへのアクセスと制御が可能な Qsys コンポーネントです。

表 9. OpenCL Kernel Interface コンポーネントのパラメーターの設定

パラメーター	説明
Number of global memory systems	ボードデザインのグローバル・メモリー・タイプの数です。

表 10. OpenCL Kernel Interface コンポーネントの信号とポート

信号とポート	説明
clk	ホスト・コントロール・インターフェイスで使用されるクロック入力です。clk のクロックレートはスローにすることができます。
reset	このリセット入力はコントロール・インターフェイスをリセットします。また、すべてのカーネルロジックをリセットする kernel_reset 信号をトリガーします。
kernel_ctrl	このスレーブポートを使用し、OpenCL ホスト・インターフェイスに接続します。このインターフェイスはカーネルの引数を設定し、カーネルの実行を開始する低速インターフェイスです。
kernel_clk	このクロック入力をドライブする OpenCL Kernel Clock Generator からの kernel_clk 出力です。
kernel_cra	この Avalon-MM マスター・インターフェイスは、インテル FPGA SDK for OpenCL オフライン・コンパイラーで生成されたカーネルと直接通信します。Avalon-MM インターフェイスを OpenCL Kernel Interface にエクスポートし、board_spec.xml ファイルに名前を付けます。
sw_reset_in	必要に応じて、OpenCL ホスト・インターフェイスは、kernel_ctrl インターフェイスを介してカーネルをリセットします。ボードデザインがカーネルリセットを要求する場合、このリセット入力を介してリセットができます。その他の場合、インターフェイスをグローバル・パワーオンリセットに接続します。

continued...

信号とポート	説明
kernel_reset	このリセット出力を使用して、カーネルとカーネルと通信するその他のハードウェアをリセットします。 ***警告:*** このリセットは MMD のオープンとクローズ間のコールで発生します。したがって、MMD の動作で必要なすべてのリセットは実行しないでください。
sw_reset_export	このリセット出力は kernel_reset と同じですが、clk インターフェイスに同期されています。この出力を使用して kernel_clk クロックドメインにないロジックをリセットしますが、カーネルをリセットするたびにリセットを行う必要があります。
acl_bsp_memorg_host	これらの信号は、メモリー・インターフェイスに使用されます。 OpenCL Kernel Interface コンポーネントのパラメーター・エディターで指定するグローバルメモリー・システムの数に基づいて、Quartus Prime ソフトウェアは、それぞれ異なる 16 進数の接尾辞を有するこのシグナルに対応した数のコピーを作成します。各信号を、各グローバルメモリー・システム (DDR など) に関連付けられた OpenCL Memory Bank Divider コンポーネントに接続します。次に、board_spec.xml ファイルにある global_mem エレメント の config_addr 属性に、16 進数の接尾辞をリストします。
kernel_irq_from_kernel	カーネルからの割り込み入力です。この信号は、board_spec.xml ファイルにエクスポートされ、名前が付けられます。
kernel_irq_to_host	カーネルからの割り込み出力です。この信号はホスト・インターフェイスに接続します。

2.1.1.3 OpenCL Memory Bank Divider

OpenCLMemory Bank Divider は、Avalon-MM スレーブポート上のホスト・インターフェイスからの受信要求を受け取り、適切なバンク・マスター・ポートに配線する Qsys コンポーネントです。このコンポーネントは、ホストとグローバルメモリー・インターフェイス間のパス上に存在していなければなりません。加えて、カーネルとグローバルメモリー・インターフェイス間のパスの外側に存在している必要があります。

表 11. OpenCL Memory Bank Divider でのパラメーター設定

パラメーター	説明
Number of banks	ボードに含まれる各グローバルメモリー・タイプごとでのメモリーバンクの数です。
Separate read/write ports	このパラメーターを有効にすると、各バンクにリード動作用とライト動作用のポートが 1 つずつあります。
Add pipeline stage to output	このパラメーターを有効にし、潜在的なタイミング改善を可能にします。
Data Width	メモリーへのデータパスの幅 (ビット単位) です。
Address Width (total addressable)	すべてのグローバルメモリーのアドレス指定に必要なアドレスビットの総数です。
Burst size (maximum)	board_spec.xml ファイルにある global_mem エレメントの interface 属性で定義されている maxburst の値です。
Maximum Pending Reads	コンポーネントが waitrequest 信号をアサートせずに処理が可能な保留中のリード送信の最大数です。 ***注意:*** 高い **Maximum Pending Reads** 値は、Qsys はコンポーネントのマスターとスレーブの間で多くのデバイスリソースを消費するディープ応答 FIFO バッファーを挿入します。また、ホストとメモリー・インターフェイス間の達成可能な帯域幅も増加します。
Split read/write bursts on burst word boundary	バースト・ワード・バウンダリーでのリードとライトのバーストの分割を有効にします。 **Number of banks** パラメーターの値を 1 より大きく、ホスト・コントローラーがスレーブポートに送信するバーストのリードとライトがバースト・ワード・バウンダリーを交差する場合は、このパラメーターを有効にします。

表 12. OpenCL Memory Bank Divider コンポーネントでの信号とポート

信号とポート	説明
clk	バンク・ディバイダー・ロジックが使用するこのクロック入力です。ホストおよびメモリー・インターフェイスの IP が異なるクロックを有する場合、clk クロックレートは、2 つの IP クロックの一番遅いものよりも遅くないようにします。
reset	ボードをパワーオンリセットに接続するリセット入力です。
s	ホスト・インターフェイス・コントローラーに接続するスレーブポートです。
kernel_clk	このクロック入力をドライブする OpenCL Kernel Clock Generator からの kernel_clk 出力です。
kernel_reset	このクロック入力を駆動する OpenCL Kernel Interface からの kernel_reset 出力です。
acl_bsp_snoop	この Avalon Streaming (Avalon-ST) ソースをエクスポートします。board_spec.xml ファイルにある interfaces インターフェイスには、デフォルトメモリー (acl_internal_snoop) のスヌープ・インターフェイスのみが記述されています。ヘテロジニアス・メモリー・デザインがある場合、これらのタスクは、デフォルトメモリーに関連付けられた OpenCL Memory Bank Divider コンポーネントに対してのみ実行してください。 *重要:* Qsys でビルドしたメモリーシステムは、acl_bsp_snoop の幅を改変します。acl_bsp_snoop の幅と一致するように、board_spec.xml ファイルにある channels エレメント内の streamsource インターフェイスの幅を更新する必要があります。 *重要:* board_spec.xml ファイルでは、interfaces エレメント内の streamsource カーネル・インターフェイスで指定されたスヌープ・インターフェイス (acl_internal_snoop) の幅を更新します。幅の更新は、board_spec.xml にある global_mem インターフェイス・エントリーが、デフォルトメモリーで対応する OpenCL Memory Bank Divider コンポーネントからの bank<*N*> Avalon-MM マスターの特性と一致している必要があります。
acl_bsp_memorg_host	OpenCL Kernel Interface の acl_bsp_memorg_host インターフェイスに接続するコンデュイットです。
bank1, bank2, ..., bank8	OpenCL Memory Bank Divider で使用可能なメモリーマスター数は、ユニットがインスタンス化されたときに含まれているメモリーバンク数によって異なります。board_spec.xml ファイルで指定された対応するカーネル・メモリー・インターフェイスの開始アドレスと同じ順序で、各バンクを各メモリー・インターフェイスに接続します。 例えば、アドレス 0 で始まる global_mem インターフェイスは、OpenCL Memory Bank Divider からのメモリーマスター bank1 に対応している必要があります。

関連情報

- 39 ページの channels
- 39 ページの interfaces
- 37 ページの global_mem
- 33 ページの OpenCL Kernel Interface

2.2 Board_spec.xml ファイル内の XML エレメント、属性、およびパラメーター

この項では、board_spec.xml ファイルに含める必要があるメタデータについて説明します。

36 ページの board
36 ページの device
37 ページの global_mem
38 ページの host

39 ページの channels
39 ページの interfaces
40 ページの interface
41 ページの compile

2.2.1 board

board_spec.xml ファイルの board エレメントは、アクセラレーター・ボードのバージョンと名前を提供します。

eXtensible Markup Language (XML) コード例：

```
<board version="<version>" name="<Custom_Platform_name>">
...
</board>
```

表 13. board エレメントでの属性

属性	説明
version	ボードのバージョンです。ボードのバージョンは、カスタム・ラットフォームの開発に使用する Quartus ソフトウェアのバージョンと一致していなければなりません。
name	アクセラレーター・ボード名で、board_spec.xml ファイルに既存するディレクトリー名と一致している必要があります。名前は、文字、数字、アンダースコア (_)、ハイフン (-)、ピリオド (.) (例：a10_ref) のみでの組み合わせでなければなりません。

2.2.2 device

board_spec.xml ファイルの device エレメントは、ボードデザインが使用するデバイスモデルとリソースを提供します。

XML コード例：

```
<device device_model="5sgsed8k2f40c2_dm.xml">
   <used_resources>
     <alms num="45000"/>
     <!-- ALMs used in final placement - ALMs used for registers -->
     <ffs num="117500"/>
     <dsps num="0"/>
     <rams num="511"/>
   </used_resources>
 </device>
```

表 14. device エレメントでの属性

属性	説明
device_model	アクセラレーター・ボード上で使用可能な FPGA リソースを記述するデバイスモデルのファイル名です。
used_resources	カーネルがない場合に、ボードデザインが消費する ALM (アダプティブ・ロジック・モジュール) 、フリップフロップ、DSP (デジタル・シグナル・プロセッサー) ブロック、および RAM ブロックの数を インテル FPGA SDK for OpenCL に通知します。すべてのボードロジックの周囲に定義されたパーティションを作成した場合、使用されたリソースデータは Fitter Report の Partition Statistics セクションから取得できます。

continued...

属性	説明
	次のパラメーターから情報を抽出します。 • alms num—これらのレジスターのみに使用されている ALM の数以外で使用されるロジック ALM の数です。この値は、Fitter Partition Statistics の [A] パートからの [a]+[b]+[d] に対応していなければなりません。 • ffs num—フィリップフロップの数です。 • dsps num—DSP ブロックの数です。 • rams num—RAM ブロックの数です。

2.2.3 global_mem

board_spec.xml ファイルの global_mem と interface エレメントは、カーネルに接続するメモリー・インターフェイスの情報を提供します。

XML コード例：

```
<!-- DDR3-1600 -->
<global_mem name="DDR" max_bandwidth="25600" interleaved_bytes="1024"
 config_addr="0x018">
  <interface name="board" port="kernel_mem0" type="slave" width="512"
maxburst="16"
   address="0x00000000" size="0x100000000" latency="240"/>
  <interface name="board" port="kernel_mem1" type="slave" width="512"
maxburst="16"
   address="0x100000000" size="0x100000000" latency="240"/>
</global_mem>

<!-- QDRII -->
  <global_mem name="QDR" max_bandwidth="17600" interleaved_bytes="8"
   config_addr="0x100">
    <interface name="board" type="slave" width="64" maxburst="1"
     address="0x200000000" size="0x1000000" latency="150" addpipe="1">
      <port name="kernel_qdr0_r" direction="r"/>
      <port name="kernel_qdr0_w" direction="w"/>
    </interface>
    <interface name="board" type="slave" width="64" maxburst="1"
     address="0x201000000" size="0x1000000" latency="150" addpipe="1">
      <port name="kernel_qdr1_r" direction="r"/>
      <port name="kernel_qdr1_w" direction="w"/>
    </interface>
    <interface name="board" type="slave" width="64" maxburst="1"
     address="0x202000000" size="0x1000000" latency="150" addpipe="1">
      <port name="kernel_qdr2_r" direction="r"/>
      <port name="kernel_qdr2_w" direction="w"/>
    </interface>
    <interface name="board" type="slave" width="64" maxburst="1"
     address="0x203000000" size="0x1000000" latency="150" addpipe="1">
      <port name="kernel_qdr3_r" direction="r"/>
      <port name="kernel_qdr3_w" direction="w"/>
    </interface>
  </global_mem>
```

注意: カーネルがアクセスする各グローバルメモリー内には、特性を記述する interface エレメントを 1 つ含める必要があります。

表 15. global_mem エレメントでの属性

属性	説明
name	インテル FPGA SDK for OpenCL ユーザーがメモリータイプの識別に使用する名前です。それぞれの名前は一意、かつ 32 文字未満でなければなりません。
max_bandwidth	現在のコンフィグレーションに結合されたすべてのグローバルメモリー・インターフェイスの単位 Mb/s (メガバイト / 秒) での最大帯域幅です。インテル FPGA SDK for OpenCL オフライン・コンパイラーは、max_bandwidth を使用し、アプリケーションとボードに適したアーキテクチャーを選択します。 メモリーのデータシートから帯域幅の値を計算します。 800 MHz で動作中の 64 ビット DDR3 インターフェイスでの max_bandwidth の計算例は次のとおりです。 max_bandwidth = 800 MHz x 2 x 64 ビット ÷ 8 ビット = 12800 MB/s ブロック RAM は、グローバルメモリーとして外部メモリーの代わり、もしくは外部メモリーと結合して、使用することができます。ブロック RAM の max_bandwidth 計算式は、max_bandwidth = ブロック RAM スピード x (ブロック RAM インターフェイスのサイズ ÷ 8 ビット) です。 100 MHz で動作中の 512 ビット・ブロック RAM での max_bandwidth の計算例は次のとおりです。 max_bandwidth = 100 MHz x 512 ビット ÷ 8 ビット = 6400 MB/s
interleaved_bytes	特定のグローバルメモリー・システムで複数のインターフェイスをインスタンス化する際、board_spec.xml ファイル内の interleaved_bytes 属性を含めます。この属性は、オフライン・コンパイラーがインターフェイス間で分配するデータのサイズを制御します。 オフライン・コンパイラーは、現在、1 つのフルバーストのサイズより細かくないバンク間でデータをインターリーブすることができます。この属性は、バイト単位でこのサイズを指定し、通常、burst_size x width_bytes で計算されます。interleaved_bytes の値は、ホスト・インターフェイスとカーネルにおいて等しくなければなりません。したがって、この観点から、OpenCL Memory Bank Divider のコンフィグレーションは、エクスポートされたカーネル・スレーブ・インターフェイスと一致していなければなりません。 ブロック RAM の場合、interleaved_bytes はインターフェイスの幅 (単位バイト) と等しくなります。
config_addr	ホスト・ソフトウェアがメモリーの設定に使用する ACL Mem Organization Control Qsys コンポーネント (mem_org_mode) のアドレスです。ボードにホモジニアス・メモリーがある場合、この属性を省略することができます。ソフトウェアは、このコンポーネントのためにデフォルトアドレス (0x18) を使用します。ボードにヘテロジニアス・メモリーがある場合、各メモリータイプのボードシステムに mem_org_mode コンポーネントがあります。 config_addr 属性を mem_org_mode コンポーネントのベースアドレス値に入力して設定します。
default	デフォルトのメモリー・インターフェイスとして、このオプションの属性を含めて値 1 をグローバルメモリーに割り当てて設定します。 この属性を実装しない場合、board_spec.xml ファイルで定義されている最初のメモリータイプがデフォルトのメモリー・インターフェイスになります。
interface	各インターフェイスで指定すべきパラメーターについては、*interface* の項を参照してください。

関連情報

40 ページの interface

2.2.4 host

board_spec.xml ファイルの host エレメントは、ホストからカーネルまでのインターフェイスの情報を提供します。

XML コード例 :

```
<host>
  <kernel_config start="0x00000000" size="0x0100000"/>
</host>
```

表 16. host エレメントでの属性

属性	説明
kernel_config	kernel_interface モジュール上の kernel_cra マスターの観点から、カーネルがどのオフセットにあるかを インテル FPGA SDK for OpenCL オフライン・コンパイラーに通知する属性です。 start: カーネルの開始アドレスです。この属性の値は、kernel_cra マスターがカーネル以外のものをマスターすべきではないため、通常は 0 です。 size: このパラメーターではデフォルト値 0x0100000 を保持してください。

2.2.5 channels

インテル FPGA SDK for OpenCL は明示的に名付けられた *channels* を介し、カーネルと I/O の間で直接データ・ストリーミングをサポートします。アクセラレーター・ボードがカーネルから I/O までの直接アクセスのチャネルを提供する場合、board_spec.xml ファイル内の channels エレメントを含めます。channels エレメントでは、Avalon-ST の仕様を使用して実装されているすべてのチャネル・インターフェイスが識別されなければなりません。interface 属性を介する各チャネル・インターフェイスを指定します。各インターフェイスに指定すべきパラメーターについては、*interface* のセクションを参照してください。チャネル・インターフェイスはデーターのみをサポートし、Avalon-ST 信号を有効にして準備します。I/O チャネルのデフォルトは、インターフェイス・レベルでの順序は、8 ビット・シンボル、およびビッグ・エンディアンです。

XML コード例：

```
<channels>
  <interface name="udp_0" port="udp0_out"  type="streamsource" width="256"
   chan_id="eth0_in"/>
  <interface name="udp_0" port="udp0_in"  type="streamsink" width="256"
   chan_id="eth0_out"/>
  <interface name="udp_0" port="udp1_out"  type="streamsource" width="256"
   chan_id="eth1_in"/>
  <interface name="udp_0" port="udp1_in"  type="streamsink" width="256"
   chan_id="eth1_out"/>
</channels>
```

関連情報

40 ページの interface

2.2.6 interfaces

board_spec.xml ファイルの interfaces エレメントは、OpenCL カーネルに接続してこれらの動作を制御するカーネル・インターフェイスを記述します。このエレメントでは、master、irq および streamsource の各インターフェイス型を 1 つを含んでいます。各インターフェイスで指定すべきパラメーターについては、*interface* のセクションを参照してください。

XML コード例：

```
<interfaces>
  <interface name="board" port="kernel_cra" type="master" width="64"
misc="0"/>
  <interface name="board" port="kernel_irq" type="irq" width="1"/>
  <interface name="board" port="acl_internal_snoop" type="streamsource"
   enable="SNOOPENABLE" width="31" clock="board.kernel_clk"/>
  <kernel_clk_reset clk="board.kernel_clk" clk2x="board.kernel_clk2x"
reset="board.kernel_reset"/>
</interfaces>
```

master、irq、および streamsource インターフェイスに加えて、デザインがボードロジックを有する個別の Qsys サブシステムを含んでいる場合、そこからエクスポートされるカーネルクロックとリセット・インターフェイスも interfaces エレメントの一部になります。kernel_clk_reset 属性とそれらに対応するパラメーターでこれらのインターフェイスを指定します。

表 17. kernel_clk_reset 属性でのパラメーター

重要: Qsys 接続形式 (つまり、<*instance_name*>.<*interface_name*>) でのカーネルクロックとリセット・インターフェイスに名前を付けます。

例 : board.kernel_clk

属性	説明
clk	カーネル・クロック・インターフェイスでの Qsys 名です。OpenCL Kernel Clock Generator コンポーネントからの kernel_clk 出力はこのインターフェイスをドライブします。
clk2x	カーネル・クロック・インターフェイスでの Qsys 名です。OpenCL Kernel Clock Generator コンポーネントからの kernel_clk2x 出力はこのインターフェイスをドライブします。
reset	カーネルリセットでの Qsys 接続です。OpenCL Kernel Interface コンポーネントからの kernel_reset 出力はこのインターフェイスをドライブします。

関連情報

40 ページの interface

2.2.7 interface

board_spec.xml ファイルでは、各グローバルメモリー、チャネル、またはカーネル・インターフェイスは個別のインターフェイスから構成されています。global_mem、channels、および interfaces XML エレメントは、各インターフェイスでの interface 属性を含み、対応するパラメーターを指定します。

表 18. interface XML 属性でのパラメーター

パラメーター	可能なインターフェイス	説明
name	すべて	global_mem 用 : Qsys コンポーネントのインターフェイス名です。 channels 用 : チャネル・インターフェイスを有する Qsys コンポーネントのインターフェイス名です。 interfaces 用 : カーネル・インターフェイスが存在するエンティティー名 (board など) です。
port		global_mem 用 :interface 属性に対応する Qsys コンポーネント内の Avalon-MM インターフェイス名です。 channels 用 : Qsys コンポーネント内のストリーミング・インターフェイス名です。 interfaces 用 :OpenCL Kernel Interface Qsys コンポーネントへのインターフェイス名です。例えば、kernel_cra は、Avalon-MM インターフェイスで、kernel_irq は割り込みです。
type		global_mem 用 :slave を設定します。 channels 用 : • カーネルにデータを提供するストリームソース用に streamsource を設定します。 • カーネルからデータを消費するストリーム・シンク・インターフェイス用に streamsink を設定します。 interfaces 用 :master、irq、または streamsource のいずれかを設定します。
width		global_mem 用 : メモリー・インターフェイスでのビット単位の幅です。

continued...

パラメーター	可能なインターフェイス	説明
		channels 用：チャネル・インターフェイスでのビット数です。 interfaces 用：カーネル・インターフェイスでのビット単位の幅です。
maxburst	global_mem	スレーブ・インターフェイスにおけるバーストの最大サイズです。 ***注意:*** width ÷ 8 x maxburst の値は、interleaved_bytes の値と同等です。
address		ホスト・インターフェイス側のアドレスに対応するメモリー・インターフェイスの開始アドレスです。 例えば、アドレス 0 は、OpenCL Memory Bank Divider からの bank1 メモリーマスターに対応している必要があります。また、すべての 0 以外の開始アドレスは、前メモリーの終了アドレスに隣接していなければなりません。
size		メモリー・インターフェイスのビット単位のサイズです。すべてのメモリー・インターフェイスのサイズは同等でなければなりません。
latency		要求応答用のメモリー・インターフェイスでのナノ秒 (ns) 単位の時間を特定する整数です。レイテンシーは、ボードシステムにメモリーのリード要求を発行してからカーネルにメモリーデータを返すまでのラウンドトリップ・タイムです。 例えば、クロック・クロッシングで 200 MHz で動作中の DDR3 メモリー・コントローラーは、およそ 240 ns のレイテンシーです。
latency_type		メモリー・インターフェイスが可変レイテンシーである場合、このパラメーターを average に設定し、指定されたレイテンシーが平均的な場合と見なされることを示します。完全なカーネルからメモリー間のパスに固定レイテンシーが保証されている場合、このパラメーターを fixed に設定します。
chan_id	channels	チャネル・インターフェイスの認識に使用され、最大 128 文字までの文字列です。
clock	interfaces	streamsource カーネル・インターフェイス型では、パラメーターはスヌーブストリームが使用するクロック名を指定します。

2.2.8 compile

board_spec.xml ファイルの compile エレメントとそれに関連する属性およびパラメーターでは、Quartus Prime のコンパイル、レジストレーション、および自動移行の一般的な制御について説明します。

XML コード例：

```
<compile project="top" revision="top" qsys_file="none" generic_kernel="1">
    <generate cmd="echo"/>
    <synthesize cmd="quartus_sh -t import_compile.tcl"/>
    <auto_migrate platform_type="a10_ref" >
        <include fixes=""/>
        <exclude fixes=""/>
    </auto_migrate>
</compile>
```

属性	説明
project	Quartus Prime ソフトウェアがコンパイルする予定の Quartus Prime プロジェクト・ファイル名 (.qpf) です。
revision	Quartus Prime ソフトウェアが インテル FPGA SDK for OpenCL オフライン・コンパイラー実行ファイル (.aocx) を生成するためにコンパイルする Quartus Prime プロジェクト内のリビジョン名です。
qsys_file	OpenCL カーネルが組み込まれている Qsys ファイル名です。 Quartus Prime ソフトウェアがデザインでトップレベルの .qsys ファイルの作成を必要としない場合、オプションで「none」の値を qsys_file に割り当てることができます。

continued...

属性	説明
generic_kernel	オフライン・コンパイラーにすべての OpenCL コンパイルで共通の Verilog インターフェイスを生成させる場合、この値を 1 に設定します。この設定は、Configuration via Protocol (CvP) フローにあるような、カーネルの周りに Quartus Prime デザイン・パーテンションを設定する場合に必要です。
generate_cmd	OpenCL カーネルが組み込まれている Qsys システムで Verilog ファイルを生成するような完全なコンパイル準備に必要なコマンドです。
synthesize_cmd	カスタム・プラットフォームから fpga.bin ファイルを生成するために必要なコマンドです。通常、このコマンドは Quartus Prime ソフトウェアに完全にコンパイルを実行するように指示します。
auto_migrate	• platform_type—カスタム・プラットフォームから得た インテル FPGA Reference Platform で参照される値に基づき、この値を選択します。有効な値は、none、s5_net、c5soc、および a10_ref です。 • include fixes—カスタム・プラットフォームに適用させる名前付き修正のコンマ区切りリストです。 • exclude fixes—カスタム・プラットフォームに適用させない名前付き修正のコンマ区切りリストです。

2.3 MMD API の概要

MMD インターフェイスは、すべての MMD アプリケーション・プログラミング・インターフェイス (API) 関数の累積です。

重要: これらの関数、引数、および戻り値についてのすべての情報は、aocl_mmd.h ファイルを参照してください。aocl_mmd.h ファイルは、インテル FPGA SDK for OpenCL カスタム・プラットフォーム・ツールキットの一部です。MMD レイヤーのファイルは、オペレーティング・システム固有の実装に含まれています。

42 ページの aocl_mmd_get_offline_info
44 ページの aocl_mmd_get_info
45 ページの aocl_mmd_open
45 ページの aocl_mmd_close
46 ページの aocl_mmd_read
47 ページの aocl_mmd_write
47 ページの aocl_mmd_copy
48 ページの aocl_mmd_set_interrupt_handler
49 ページの aocl_mmd_set_status_handler
50 ページの aocl_mmd_yield
50 ページの aocl_mmd_shared_mem_alloc
51 ページの aocl_mmd_shared_mem_free
51 ページの aocl_mmd_reprogram

2.3.1 aocl_mmd_get_offline_info

aocl_mmd_get_offline_info 関数は、requested_info_id 引数に指定されたボードに関するオフライン情報を入手します。この関数はデバイスに起因せず、aocl_mmd_open()コールからのハンドルが不要なため、オフラインです。

構文

```
int aocl_mmd_get_offline_info( aocl_mmd_offline_info_t requested_info_id,
                               size_t param_value_size,
                               void* param_value,
                               size_t* param_size_ret )
```

関数の引数

1. `requested_info_id`—コール元に返すオフラインデバイス情報が示す `aocl_mmd_offline_info_t` 型の列挙値です。

表 19. requested_info_id 引数での可能な列挙値

名前	説明	型
AOCL_MMD_VERSION	MMD レイヤーのバージョンです。	char*
AOCL_MMD_NUM_BOARDS	候補ボードの数です。	int
AOCL_MMD_BOARD_NAMES	使用可能なボードの数です。 ***注意:*** 各ボード名はセミコロン (;) で区切ります。	char*
AOCL_MMD_VENDOR_NAME	ボードベンダーの名前です。	char*
AOCL_MMD_VENDOR_ID	整数ボードベンダー ID です。	int
AOCL_MMD_USES_YIELD	値 0 は、インテル FPGA SDK for OpenCL のホストランタイムにユーザーのプロセスの中断を指示します。ホストランタイムは MMD レイヤーからイベント更新 (割り込みなど) を受信するとプロセスを再開します。 値 1 は、イベントが終了するまでの待機中、SDK のホストランタイムに `aocl_mmd_yield` 関数の継続的なコールを指示します。 ***注意:*** `aocl_mmd_yield` 関数が現在のスレッドを中断しない場合は、AOCL_MMD_USES_YIELD の 1 への設定は CPU の高使用率を引き起こす可能性があります。	int
AOCL_MMD_MEM_TYPES_SUPPORTED	カスタム・プラットフォームがサポートするすべてのメモリータイプをリストしているビットフィールドです。このフィールドに次の列挙値を組み合わることができます。 AOCL_MMD_PHYSICAL_MEMORY—カスタム・プラットフォームは、物理メモリー (DDR や QDR など) と直接通信する IP を含んでいます。 AOCL_MMD_SVM_COARSE_GRAIN_BUFFER—カスタム・プラットフォームは、OpenCL `cl_mem` オブジェクトでの共有仮想メモリー (SVM) ポインターデータのキャッシュと、ホスト・プロセッサーと FPGA 間でキャッシュを同期するための明示的なユーザー関数コールの要件の両方をサポートしています。 *注意:* 現在、インテル は AOCL_MMD_SVM_FINE_GRAIN_SYSTEM サポートのサブセットを除いたこの SVM のレベルをサポートしていません。 AOCL_MMD_SVM_FINE_GRAIN_BUFFER—カスタム・プラットフォームは、個別バイトでの SVM ポインターデータのキャッシングをサポートしています。カスタム・プラットフォームは、ホスト・プロセッサーと FPGA の間のキャッシュを同期するために、ポインター割り当ての最中に収集したホストランタイムからの情報を必要とします。SVM ポインターがこの追加データを受信した後、board インターフェイスはホスト・プロセッサーと FPGA の間のキャッシュを自動的に同期します。 *注意:* 現在、インテル は AOCL_MMD_SVM_FINE_GRAIN_SYSTEM サポートのサブセット以外のこの SVM レベルをサポートしていません。	int (ビットフィールド)

continued...

名前	説明	型
	AOCL_MMD_SVM_FINE_GRAIN_SYSTEM—カスタム・プラットフォームは、個別バイトの SVM ポインターデータのキャッシュをサポートし、加えて、ホスト・プロセッサーと FPGA の間のキャッシュを同期するためのホストランタイムからの追加情報を必要としません。board インターフェイスは、すべての SVM ポインターにおいてホスト・プロセッサーと FPGA の間のキャッシュを自動的に同期します。 ***注意:*** この SVM レベルでの インテル のサポートは、暫定的なものです。機能のいくつかは完全にサポートされない場合があります。	

2. param_value_size—param_value フィールドのビット単位のサイズです。size_t 値は、列挙が定義で示す予期される応答の型のサイズと一致していなければなりません。

   例えば、AOCL_MMD_NUM_BOARDS が int 型の値を返した場合、param_value_size を sizeof (int)に設定します。param_size_ret 引数で返されたバイト数と同じ数が表示されなければなりません。
3. param_value—返された情報を受け取る変数への void*ポインターです。
4. param_size_ret—返されるデータのバイト数を受け取る size_t*型のポインター引数です。

戻り値

負の戻り値はエラーを示します。

2.3.2 aocl_mmd_get_info

aocl_mmd_get_info 関数は、requested_info_id 引数に指定されたボードに関する情報を取得します。

構文

```
int aocl_mmd_get_info( int handle,
                       aocl_mmd_info_t requested_info_id,
                       size_t param_value_size,
                       void* param_value,
                       size_t* param_size_ret );
```

関数の引数

1. handle—aocl_mmd_open() コールから入手したボードのハンドルを表す正の int 値です。
2. requested_info_id—コール元に返すデバイス情報が示す aocl_mmd_offline_info_t 型の列挙値です。

表 20. requested_info_id 引数での正の列挙値

名前	説明	型
AOCL_MMD_NUM_KERNEL_INTERFACES	カーネル・インターフェイスの数です。	int
AOCL_MMD_KERNEL_INTERFACES	カーネル・インターフェイスです。	int*
AOCL_MMD_PLL_INTERFACES	カーネル・クロック・ハンドルです。	int*
AOCL_MMD_MEMORY_INTERFACE	グローバルメモリー・ハンドルです。	int

continued...

名前	説明	型
AOCL_MMD_TERMPERATURE	温度測定です。	float
AOCL_MMD_PCIE_INFO	PCIe 情報です。	char*
AOCL_MMD_BOARD_NAME	ボードの名前です。	char*
AOCL_MMD_BOARD_UNIQUE_ID	特有ボード ID です。	char*

3. param_value_size—param_value フィールドのバイト単位のサイズです。size_t 値は、列挙が定義で示す予期される応答の型のサイズと一致していなければなりません。

   例えば、AOCL_MMD_TEMPERATURE が float 型の値を返す場合、param_value_size を sizeof (float)に設定します。param_size_ret 引数で返されるバイト数と同じ数が表示されなければなりません。
4. param_value—返された情報を受信する変数への void*ポインターです。
5. param_size_ret—返されるデータのバイト数を受信する size_t*型のポインター引数です。

戻り値

負の戻り値はエラーを示します。

2.3.3 aocl_mmd_open

aocl_mmd_open 関数は、指定されたデバイスを開き、初期化します。

構文

```
int aocl_mmd_open( const char* name );
```

関数の引数

name—この関数は、この const char*文字列と一致する名前でボードを開きます。名前は、通常、AOCL_MMD_BOARD_NAMES オフライン情報で指定されたものと一致します。

OpenCL ランタイムは、まず AOCL_MMD_BOARD_NAMES オフライン情報を照会してオープン可能なボードを識別します。次に、aocl_mmd_open を呼び出し、各ボード名を引数として使用することで、すべての可能なデバイスをオープンしようとします。

重要: 名前は C スタイルの NULL で終了する ASCII 文字列でなければなりません。

戻り値

aocl_mmd_open()が正常に実行されなかった場合、戻り値はボードのハンドルとして機能する正の整数になります。

aocl_mmd_open()が正常に実行されない場合、負の戻り値はエラーを表示します。エラーが発生した場合、OpenCL ランタイムは他の既知のデバイスをオープンします。したがって、オープンコールが正常に行われない場合、MMD レイヤーがアプリケーションを終了しないことが必須です。

2.3.4 aocl_mmd_close

aocl_mmd_close 関数はハンドルを介してオープンされたデバイスをクローズします。

構文

```
int aocl_mmd_close( int handle );
```

関数の引数

handle—aocl_mmd_open()コールで得られたボードのハンドルを表す正の int 値です。

戻り値

aocl_mmd_close()が正常に実行された場合、戻り値は 0 です。

aocl_mmd_close()が正常に実行されなかった場合、負の戻り値はエラーを示します。

2.3.5 aocl_mmd_read

aocl_mmd_read 関数は、シングル・インターフェイスではリード動作です。

構文

```
int aocl_mmd_read( int handle,
                   aocl_mmd_op_t op,
                   size_t len,
                   void* dst,
                   aocl_mmd_interface_t interface,
                   size_t offset );
```

関数の引数

1. handle—aocl_mmd_open()コール元から入手したボードへのハンドルを表す正の int 値です。
2. op—動作の進行状況を追跡するために使用される型 aocl_mmd_op_t の動作オブジェクトです。op が NULL の場合、コールはブロックされ、動作が完了した後にのみ返されます。

   注意: aocl_mmd_op_t は次のとおり定義されます。

   ```
   typedef void* aocl_mmd_op_t;
   ```

3. len—関数を転送するデータのバイト単位のサイズです。型 size_t で len を宣言します。
4. dst—型 void*のデータが書き込まれるホストバッファーです。
5. interface—aocl_mmd_read がアクセスしているインターフェイスへのハンドルです。例えば、グローバルメモリーにアクセスする場合のハンドルは、requested_info_id 引数が AOCL_MMD_MEMORY_INTERFACE の際に返す列挙値 aocl_mmd_get_info()です。interface 引数は型 aocl_mmd_interface_t で、次のいずれかの値を取ります。

名前	説明
AOCL_MMD_KERNEL	カーネル・インターフェイス内のインターフェイスの制御です。
AOCL_MMD_MEMORY	デバイスメモリーへのデータ・インターフェイスです。
AOCL_MMD_PLL	リコンフィグレーションが可能な PLL 用のインターフェイスです。

6. offset—データ転送が開始が生じるインターフェイス内の size_t バイトオフセットです。

戻り値

リード動作が成功した場合、戻り値は 0 です。

リード動作が成功しなかった場合、負の戻り値はエラーを示します。

2.3.6 aocl_mmd_write

aocl_mmd_write 関数は、シングル・インターフェイスでのライト動作です。

構文

```
int aocl_mmd_write( int handle,
                    aocl_mmd_op_t op,
                    size_t len,
                    const void* src,
                    aocl_mmd_interface_t interface,
                    size_t offset );
```

関数の引数

1. handle—aocl_mmd_open()コールで入手したボードのハンドルを表す正の int 値です。
2. op—動作の進行状況を追跡するために使用される型 aocl_mmd_op_t の動作オブジェクトです。op が NULL の場合、コールはブロックされ、動作が完了した後にのみ返されます。

   注意: aocl_mmd_op_t は次のとおり定義されます。

   ```
   typedef void* aocl_mmd_op_t;
   ```

3. len—関数が転送するデータのバイト単位のサイズです。型 size_t で len を宣言します。
4. src—データが読み込まれる型 const void*のホストバッファーです。
5. interface—aocl_mmd_write がアクセスしているインターフェイスへのハンドルです。例えば、グローバルメモリーにアクセスする場合のハンドルは、requested_info_id 引数が AOCL_MMD_MEMORY_INTERFACE の際に返す列挙値 aocl_mmd_get_info()です。interface 引数は型 aocl_mmd_interface_t で、次のいずれかの値を取ります。

名前	説明
AOCL_MMD_KERNEL	カーネル・インターフェイス内のインターフェイスの制御です。
AOCL_MMD_MEMORY	デバイスメモリーへのデータ・インターフェイスです。
AOCL_MMD_PLL	リコンフィグレーションが可能な PLL 用のインターフェイスです。

6. offset—データ転送が開始が生じるインターフェイス内の size_t バイトオフセットです。

戻り値

リード動作が成功した場合、戻り値は 0 です。

リード動作が失敗した場合、負の戻り値はエラーを示します。

2.3.7 aocl_mmd_copy

aocl_mmd_copy 関数は、シングル・インターフェイスでのコピー動作です。

構文

```
int aocl_mmd_copy( int handle,
                   aocl_mmd_op_t op,
                   size_t len,
                   aocl_mmd_interface_t intf,
                   size_t src_offset,
                   size_t dst_offset );
```

関数の引数

1. handle—aocl_mmd_open()コールで入手したボードへのハンドルを表す正の int 値です。
2. op—操作の進行状況を追跡するために使用される型 aocl_mmd_op_t のオペレーション・オブジェクトです。op が NULL の場合、コールはブロックされ、動作が完了した後にのみ返されます。

   注意: aocl_mmd_op_t は次とおり定義されます。

   ```
   typedef void* aocl_mmd_op_t;
   ```

3. len—関数が転送するデータのバイト単位のサイズです。型 size_t で len を宣言します。
4. intf—aocl_mmd_read がアクセスしているインターフェイスへのハンドルです。例えば、グローバルメモリーにアクセスする場合のハンドルは、requested_info_id 引数が AOCL_MMD_MEMORY_INTERFACE の際に返す列挙値 aocl_mmd_get_info()です。interface 引数は aocl_mmd_interface_t 型で、次のいずれかの値を取ります。

名前	説明
AOCL_MMD_KERNEL	カーネル・インタフェース内のインタフェースの制御です。
AOCL_MMD_MEMORY	デバイスメモリーへのデータ・インターフェイスです。
AOCL_MMD_PLL	リコンフィグレーションが可能な PLL のためのインターフェイスです。

5. src_offset—データ転送が開始が生じるソース・インターフェイス内の size_t バイトオフセットです。
6. dst_offset—データ転送が開始されるデスティネーション・インタフェース内の size_t バイトオフセットです。

戻り値

コピー動作が成功した場合、戻り値は 0 です。

コピー動作が失敗した場合、負の戻り値はエラーを示します。

2.3.8 aocl_mmd_set_interrupt_handler

aocl_mmd_set_interrupt_handler 関数は、オープンされたデバイスに対して割り込みハンドラーを設定します。デバイス内部で非同期カーネルイベント関数は、オープンされたデバイスの割り込みハンドラーを設定します。デバイス内部で非同期カーネルイベント (カーネルの完了など) が識別されると、イベントの OpenCL ランタイムに通知するために割り込みハンドラーがコールされます。

注意: このハンドラーが設定されるまで、カーネルからの割り込みは無視してください。

構文

```
int aocl_mmd_set_interrupt_handler( int handle,
                                    aocl_mmd_interrupt_handler_fn fn,
                                    void* user_data );
```

関数の引数

1. handle—aocl_mmd_open()コールで入手したボードへのハンドルを表す正の int 値です。
2. fn—カーネルの割り込みが発生した際に呼び出されるコールバック関数です。fn 引数の型は aocl_mmd_interrupt_handler_fn で、次のとおり定義されます。

```
typedef void (*aocl_mmd_interrupt_handler_fn)( int handle, void*
user_data );
```

3. user_data—void*型のユーザー提供データで呼び出されると fn に渡されます。

戻り値

関数が正常に実行された場合、戻り値は 0 です。

関数が正常に実行されなかった場合、負の戻り値はエラーを示します。

2.3.9 aocl_mmd_set_status_handler

aocl_mmd_set_status_handler 関数は、オープンのデバイスの動作ステータスハンドラーを設定します。動作ステータスハンドラーは、次の状況下で呼び出されます。

- 動作が正常に完了し、ステータスが 0 の場合
- 動作がエラーで完了し、ステータスが負の値になった場合

構文

```
int aocl_mmd_set_status_handler( int handle,
                                 aocl_mmd_status_handler_fn fn,
                                 void* user_data );
```

関数の引数

1. handle—aocl_mmd_open()コールで入手したボードへのハンドルを表す正の int 値です。
2. fn—ステータスの更新が発生した際に呼び出されるコールバック関数です。fn 引数の型は aocl_mmd_status_handler_fn で、次のとおり定義されます。

```
type void (*aocl_mmd_status_handler_fn)( int handle, void* user_data,
aocl_mmd_op_t op, int status );
```

3. user_data—void*型のユーザーが提供したデータで、コールされると fn に渡されます。

戻り値

関数が正常に実行された場合、戻り値は 0 です。

関数が正常に実行されなかった場合、負の戻り値はエラーを示します。

2.3.10 aocl_mmd_yield

aocl_mmd_yield 関数は、ホスト・インターフェイスがアイドル状態の際に呼び出されます。デバイスが特定のイベントを処理するのを待機しているため、ホスト・インターフェイスがアイドル状態になっている可能性があります。

構文

```
int aocl_mmd_yield( int handle );
```

関数の引数

1. handle—aocl_mmd_open()コールで入手したボードへのハンドルを表す正の int 値です。

戻り値

戻り値 0 以外は、yield 関数がダイレクト・メモリー・アクセス (DMA) トランザクションの処理などの適切なデバイス機能に必要な作業を実行したことを示します。

戻り値 0 は、適切なデバイス機能に必要な作業を yield 関数が実行しなかったことを示します。

注意: yield 関数は、実行に必要な作業があると報告される限り、継続的に呼び出されることがあります。

2.3.11 aocl_mmd_shared_mem_alloc

aocl_mmd_shared_mem_alloc 関数は、ホストと FPGA の間で共有メモリーを割り当てます。ホストは、aocl_mmd_shared_mem_alloc で返されたポインターを介して共有メモリーにアクセスします。FPGA は、device_ptr_out を介して共有メモリーにアクセスします。共用メモリーが使用できない場合、aocl_mmd_shared_mem_alloc は NULL を返します。FPGA を再プログラムした後に CPU を再起動しない場合、共有メモリーは保持されます。

構文

```
void * aocl_mmd_shared_mem_alloc( int handle,
                                  size_t size,
                                  unsigned long long *device_ptr_out );
```

関数の引数

1. handle—aocl_mmd_open()コールで入手したボードへのハンドルを表す正の int 値です。
2. size—関数が割り当てる共有メモリーのサイズです。型 size_t で size を宣言します。
3. device_ptr_out—デバイスが共有メモリーにアクセスするために使用するポインター値を受け取る引数です。device_ptr_out 引数の型は unsigned long long で、ホストがデバイスよりも小さいポインターサイズを有する場合にハンドルします。device_ptr_out 引数は NULL 値を設定することはできません。

戻り値

aocl_mmd_shared_mem_alloc が正常に実行された場合、戻り値はホストが共有メモリーにアクセスするために使用するポインター値です。それ以外の場合、戻り値は NULL です。

2.3.12 aocl_mmd_shared_mem_free

aocl_mmd_shared_mem_free 関数は、割り当てられた共有メモリーを解放します。共有メモリーが使用できない場合、この関数は何も行いません。

構文

```
void aocl_mmd_shared_mem_free( int handle,
                               void* host_ptr,
                               size_t size );
```

関数の引数

1. handle—aocl_mmd_open()コールで入手したボードへのハンドルを表す正の int 値です。
2. host_ptr—aocl_mmd_shared_mem_alloc()関数により返された共有メモリーを指すホストポインターです。
3. size—関数が解放する、割り当てられた共有メモリーのサイズです。型 size_t で size を宣言します。

戻り値

aocl_mmd_shared_mem_free 関数には戻り値がありません。

2.3.13 aocl_mmd_reprogram

aocl_mmd_reprogram 関数は、指定されたデバイスでの再プログラム動作です。ホストは、再プログラム動作中に他の OpenCL 動作がデバイス上で実行されていないことを保証していなければなりません。aocl_mmd_reprogram の実行中、カーネルはアイドル状態であり、読み取り、書き込み、およびコピー動作は実行できません。

割り込みを無効にし、size 引数で指定されたサイズを持つ user_data からのデータを使用して FPGA を再プログラミングします。次に、ホストは、aocl_mmd_set_status_handler および aocl_mmd_set_interrupt_handler を再度コールし、割り込みを有効にします。aocl_mmd_reprogram の実行中に割り込みなどのイベントが生じると、競合状態またはデータ破損が発生する可能性があります。

構文

```
int aocl_mmd_reprogram( int handle,
                        void* user_data,
                        size_t size );
```

関数の引数

1. handle—aocl_mmd_open()コールで入手したボードへのハンドルを表す正の int 値です。
2. user_data—コンパイル中に作成される fpga.bin ファイルの void*型のバイナリー内容です。
3. size—user_data のバイト単位のサイズです。size 引数は size_t です。

戻り値

`aocl_mmd_reprogram` が正常に実行された場合、戻り値はホストが共有メモリーへのアクセスに使用するポインターの値です。

2.3.13.1 再プログラミングのサポート

`clCreateProgramWithBinary` フロー（すなわち、オンザフライで再プログラミング）で FPGA をプログラミングする インテル FPGA SDK for OpenCL ユーザーの場合、`aocl_mmd_reprogram` サブルーチンはホスト・アプリケーション内から FPGA を構成するために使用されます。ホストは、FPGA がアイドル状態のときにのみ、このコールが実行されることを保証します。つまり、カーネルが実行されておらず、転送も未処理であることを意味します。MMD レイヤーは、`aocl_mmd_reprogram` の `user_data` 引数のデータを使用してデバイスを再構成する必要があります。

`user_data` 引数のデータは、Quartus Prime のコンパイル時に作成された同じ `fpga.bin` データです。インテル FPGA SDK for OpenCL オフライン・コンパイラーは、コンパイル時に `fpga.bin` の正確な内容を `.aocx` ファイルにパッケージ化します。`fpga.bin` の内容は、オフライン・コンパイラーとは関係がありません。シンプルにファイル内容をホスト経由で渡し、`user_data` 引数を介して `aocl_mmd_reprogram` コールに渡します。

`clCreateProgramWithBinary` 関数について詳しくは、*OpenCL Specification version 1.0* (https://www.khronos.org/) および *インテル FPGA SDK for OpenCL Programming Guide* の *Programming an FPGA via the Host* の項（英語版）を参照してください。

関連情報

- OpenCL Specification version 1.0
- ホスト経由の FPGA のプログラミング

2.4 改訂履歴

表 21. インテル FPGA SDK for OpenCL カスタム・プラットフォーム・ツールキット・ユーザーガイドの インテル FPGA SDK for OpenCL カスタム・プラットフォーム・ツールキット・リファレンス・マニュアルの章における改訂履歴

日付	バージョン	変更内容
2016 年 10 月	2016.10.31	• アルテラ SDK for OpenCL から「インテル FPGA SDK for OpenCL」へ変更。 • アルテラ・オフライン・コンパイラーから「インテル FPGA SDK for OpenCL オフライン・コンパイラー」へ変更。 • 「*OpenCL Memory Bank Divider*」のセクションで、**Split read/write bursts on burst word boundary** パラメーターの情報を追加。
2016 年 5 月	2016.05.02	• 項「*board_spec.xml ファイルでの XML エレメント、属性、およびパラメーター*」の「*global_mem*」のセクションで、`max_bandwidth` 値を決定する計算例を追加。 • 項「*board_spec.xml ファイルでの XML エレメント、属性、およびパラメーター*」の「*interface*」のセクションで、`name` および `port` 属性での `global_mem` 固有の定義を変更。 • `compile` エレメント内の `qsys_file` 属性に `none` の値を割り当てるオプションを追加。 • 「`aocl_mmd_copy`」のセクションの内容を修正。
2015 年 11 月	2015.11.02	• メンテナンス・リリース、および、表記を *Quartus II* から「*Quartus Prime*」へ変更。

continued...

日付	バージョン	変更内容
2015 年 5 月	15.0.0	• メンテナンス・リリース
2014 年 12 月	14.1.0	• 項「*board_spec.xml ファイルでの XML エレメント、属性、およびパラメーター*」で、compile eXtensible Markup Language エレメントおよびそれらに関連する属性とパラメーターの情報を追加。 • 項「*MMD API の概要*」で、aocl_mmd_get_offline_info 関数に requested_info_id 変数の AOCL_MMD_USES_YIELD および AOCL_MMD_MEM_TYPES_SUPPORTED 列挙値に関する情報を追加。
2014 年 10 月	14.0.1	• 既存の資料を 2 つの章に再編成。
2014 年 6 月	14.0.0	• 初版