Altera SDK for OpenCL

Custom Platform Toolkit User Guide

Last updated for Quartus Prime Design Suite: 15.1
UG-OCL007
2015.11.02

101 Innovation Drive
San Jose, CA 95134
www.altera.com

Subscribe
Send Feedback

Contents

Altera SDK for OpenCL Custom Platform Toolkit User Guide

1

2015.11.02

UG-OCL007 Subscribe Send Feedback

The *Altera SDK for OpenCL Custom Platform Toolkit User Guide* outlines the procedure for creating an Altera® Software Development Kit (SDK) for OpenCL™ (AOCL) Custom Platform.

The Altera SDK for OpenCL[1][2] Custom Platform Toolkit provides the necessary tools for implementing a fully functional Custom Platform. The Custom Platform Toolkit is available in the ***ALTERAOCLSDKROOT*/board** directory, where the environment variable *ALTERAOCLSDKROOT* points to the location of the AOCL installation.

The goal is to enable an AOCL user to target any given Custom Platform seamlessly by performing the following tasks:

1. Acquire an accelerator board and plug it into their system.
2. Acquire the Custom Platform and unpack it to a local directory.
3. Set the environment variable *AOCL_BOARD_PACKAGE_ROOT* to point to this local directory.
4. Set the environment variable *QUARTUS_ROOTDIR_OVERRIDE* to point to installation directory of the Quartus® Prime Standard Edition software or the Quartus Prime Pro Edition software, depending on the target device.
5. Invoke the `aocl install` utility command.
6. Compile the OpenCL kernel and build the host application.
7. Set environment variables to point to the location of the memory-mapped device (MMD) library.
   - For Windows systems, set the *PATH* environment variable.
   - For Linux systems, set the *LD_LIBRARY_PATH* environment variable.
8. Run the host application.

Prerequisites for the Altera SDK for OpenCL Custom Platform Toolkit

The *Altera SDK for OpenCL Custom Platform Toolkit User Guide* assumes that you have prior hardware design knowledge necessary for using the Custom Platform Toolkit to create an Altera SDK for OpenCL Custom Platform.

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

(2) The Altera SDK for OpenCL is based on a published Khronos Specification, and has passed the Khronos Conformance Testing Process. Current conformance status is available at **www.khronos.org/conformance**.

101 Innovation Drive, San Jose, CA 95134

You must have experiences in the following hardware design areas:

- Quartus Prime software design with Qsys, HDL and Tcl
- Altera intellectual property (IP) necessary to communicate with the physical interfaces of the board
- High speed design, timing analysis and Synopsys Design Constraints (SDC) constraints
- FPGA architecture, including clock and global routing, floorplanning, and I/O
- Team-based design (that is, incremental compilation)

You must install the Quartus Prime software, the relevant device support file(s), and the AOCL on your machine. Depending on the target device, you must install the Quartus Prime Standard Edition software, the Quartus Prime Pro Edition software, or both. Refer to the *Altera SDK for OpenCL Getting Started Guide* for installation instructions.

You have the following Custom Platform design options:

- Refer to the information in this document to create a Custom Platform from the templates available in the Custom Platform Toolkit.
- Refer to the information in this document and the *Altera Stratix V Network Reference Platform Porting Guide* to create a Custom Platform by modifying relevant files in s5_net.

  Download the Altera Stratix® V Network Reference Platform (s5_net) from the Altera SDK for OpenCL FPGA Platforms page on the Altera website. The link for the download is under **Custom**.
- Refer to the information in the following documents to create a Custom Platform by modifying relevant files in the Cyclone® V SoC Development Kit Reference Platform (c5soc), available with the AOCL:

  1. *Altera SDK for OpenCL Custom Platform Toolkit User Guide*
  2. *Altera Cyclone V SoC Development Kit Reference Platform Porting Guide*
  3. *Cyclone V SoC Development Board Reference Manual*

Related Information

- **Altera SDK for OpenCL Getting Started Guide**
- **Altera Stratix V Network Reference Platform Porting Guide**
- **Altera Cyclone V SoC Development Kit Reference Platform Porting Guide**
- **Cyclone V SoC Development Board Reference Manual**
- **Altera SDK for OpenCL FPGA Platforms page on the Altera website**

Overview of the AOCL Custom Platform

An Altera SDK for OpenCL Custom Platform is a collection of tools and libraries necessary for the communication between the Altera Offline Compiler (AOC) and the FPGA boards.

Currently, the AOC targets a single Custom Platform at a time.

The environment variable *AOCL_BOARD_PACKAGE_ROOT* points to the path of the **board_env.xml** board environment eXtensible Markup Language (XML) file within a Custom Platform. A given Custom Platform installation can include several board variants of the same board interface. You might have different FPGA parts, or you might want to support different subsets of board interfaces. Colocating the board variants allows simultaneous communication with different boards in a multiple-device environment.

An AOCL Custom Platform contains the following components:

- **Quartus Prime skeleton project**—A Quartus Prime project for your board, which the AOC modifies to include the compiled kernel. This project must include a post-place-and-route partition for all logic not controlled by the kernel clock.
- **Board installation setup**—A description of your board and its various components.
- **Generic I/O interface**—An MMD software library that implements basic I/O between the host and the board.
- **Board utilities**—An implementation of AOCL utilities for managing the accelerator board, including tasks such as installing and testing the board.

Directories and Files in an AOCL Custom Platform

Populate your Altera SDK for OpenCL Custom Platform with files, libraries and drivers that allow an OpenCL kernel to run on the target FPGA board.

Table 1-1: Contents within the Top-Level Custom Platform Directory

Content	Description
board_env.xml	The XML file that describes the board installation to the AOCL.
<hardware>	Directory containing the Quartus Prime projects for the supported boards within a given Custom Platform. Specify the name of this directory in the **board_env.xml** file. Within this directory, the AOCL assumes that any subdirectory containing a **board_spec.xml** file is a board.
include	Directory containing board-specific header files.
source	Directory containing board-specific files, libraries and drivers.
platform	Directory containing platform-specific (for example, x86_64 Linux) drivers and utilities.

Recommendations for Structuring the Custom Platform Directory

For ease of use, consider adopting the Altera®-recommended directory structure and naming convention when you create an Altera SDK for OpenCL™ Custom Platform.

- Make the ***ALTERAOCLSDKROOT*/board** directory the location of the board installation, where *ALTERAOCLSDKROOT* points to the location of the AOCL installation.

  Attention: Do not remove any existing subdirectories from the ***ALTERAOCLSDKROOT*/board** directory.
- Create a ***<board_vendor_name>*** subdirectory within the ***ALTERAOCLSDKROOT*/board** directory to store the Custom Platform.
- Store the contents of a given Custom Platform in a ***ALTERAOCLSDKROOT*/board/<*board_vendor_name*>/<*board_family_name*>** subdirectory.
- Assign unique names to software libraries (for example, **lib<*board_vendor_name*>_<*board_family_name*>.so**) to avoid name collisions.

For example, if you (ABC Incorporated) create a Custom Platform for a family of boards named XYZ, set up your Custom Platform such that the AOCL user can access XYZ by performing the following tasks:

1. Install the XYZ Custom Platform in ***ALTERAOCLSDKROOT*/board/ABC/XYZ**, where *ALTERAOCLSDKROOT* is the environment variable that points to the absolute path to the AOCL installation package.
2. Set the *AOCL_BOARD_PACKAGE_ROOT* environment variable to point to ***ALTERAOCLSDKROOT*/board/ABC/XYZ**.

Custom Platform Automigration for Forward Compatibility

The automigration feature updates an existing Altera-registered Custom Platform for use with the current version of the Quartus Prime Design Suite® (QPDS) and the Altera SDK for OpenCL.

Important: Automigration is more likely to complete successfully if your Custom Platform resembles an Altera Reference Platform as closely as possible.

The following information applies to a Custom Platform that is version 14.0 and beyond:

1. To update a Custom Platform for use with the current version of the QPDS, which includes the AOCL, do not modify your Custom Platform. The automigration capability detects the version of your Custom Platform based on certain characteristics and updates it automatically.
2. If you have modified a Custom Platform and you want to update it for use with the current version of the QPDS, which includes the AOCL, implement all features mandatory for the current version of the Custom Platform. After you modify a Custom Platform, automigration can no longer correctly detect its characteristics. Therefore, you must upgrade your Custom Platform manually.

A successfully-migrated Custom Platform will preserve its original functionality. In most cases, new features in a new QPDS or AOCL version will not interfere with Custom Platform functionality.

When the Altera Offline Compiler compiles a kernel, it probes the **board_spec.xml** file for the following information:

1. The version of the Custom Platform, as specified by the `version` attribute of the `board` XML element.
2. The platform type, as specified by the `platform_type` parameter of the `auto_migrate` attribute within the `compile` XML element.

Based on the information, the AOCL names a set of fixes it must apply during Custom Platform migration. It applies the fixes to the Quartus Prime project that the AOC uses to compile the OpenCL kernel. It also generates an **automigration.rpt** report file in the AOCL user's current working directory describing the applied fixes.

The automigration process does not modify the installed Custom Platform.

Note: If automigration fails, contact your local Altera field applications engineer for assistance.

Customizing Automigration

You and the Altera SDK for OpenCL user both have the ability to disable the automigration of an installed Custom Platform. In addition, you may choose which named fixes, identified by the AOCL, you want to apply to your Custom Platform.

1. Disable automigration in one of the following manners:

- If you are a board developer, within the `compile` XML element in the **board_spec.xml** file, set the `platform_type` parameter of the `auto_migrate` attribute to `none`.
- If you are an AOCL user, invoke the `aoc --no-auto-migrate` command.

2. To explicitly include or exclude fixes that the AOCL identifies, in the **board_spec.xml** file, subscribe or unsubscribe to each fix by listing it in the `include fixes` or `exclude fixes` parameter, respectively. The `include fixes` and `exclude fixes` parameters are part of the `auto_migrate` attribute within the `compile` element. When listing multiple fixes, separate each fix by a comma.

   Refer to the **automigration.rpt** file for the names of the fixes that you specify in the `include fixes` and `exclude fixes` parameters.

Creating an AOCL Custom Platform

The following topics outline the tasks you must perform to create a Custom Platform for use with the Altera SDK for OpenCL.

1. **Designing the Board Hardware** on page 1-5
   To design an accelerator board for use with the Altera SDK for OpenCL, you must create all the board and system components, and the files that describe your hardware design to the Altera Offline Compiler.
2. **Creating the Board XML Files** on page 1-10
   Your Custom Platform must include the XML files that describe your Custom Platform and each of your hardware system to the Altera SDK for OpenCL.
3. **Creating the MMD Library** on page 1-15
   Your Custom Platform requires an MMD layer necessary for communication with the accelerator board.
4. **Setting Up the Altera Client Driver** on page 1-17
   The ACD allows the AOCL to automatically find and load the Custom Platform libraries at host runtime.
5. **Providing AOCL Utilities Support** on page 1-18
   Each Custom Platform you develop for use with the Altera SDK for OpenCL must support a set of AOCL utilities. These utilities enable users to manage the accelerator board through the AOCL.
6. **Testing the Hardware Design** on page 1-20
   After you create the software utilities and the MMD layer, and your hardware design achieves timing closure, test the design.

Designing the Board Hardware

To design an accelerator board for use with the Altera SDK for OpenCL, you must create all the board and system components, and the files that describe your hardware design to the Altera Offline Compiler.

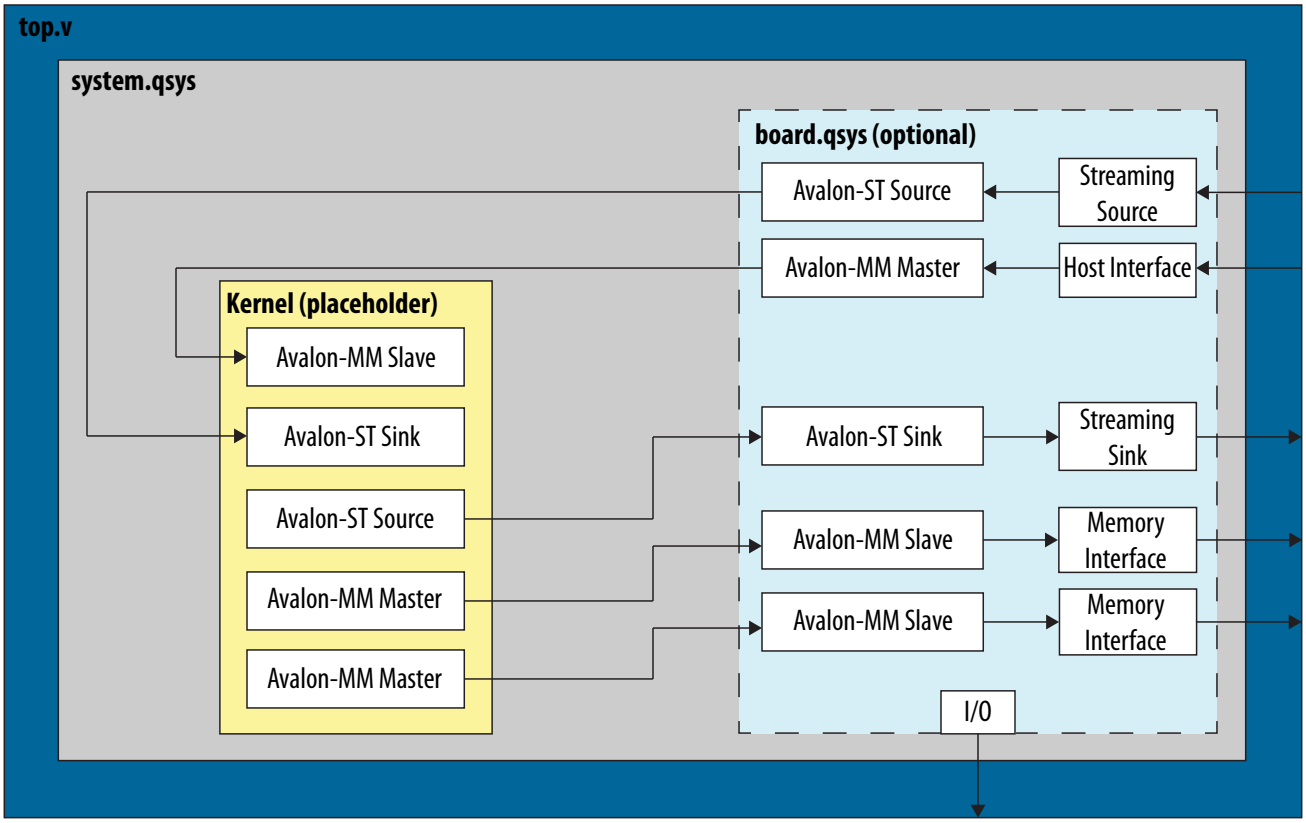
Each board variant in the Custom Platform consists of a Quartus Prime project, and a **board_spec.xml** XML file that describes the system to the AOC. The **board_spec.xml** file describes the interfaces necessary to connect to the kernel. The AOC generates a custom circuit based on the data from the **board_spec.xml** file. Then it incorporates the OpenCL kernel into the Qsys system you create for all nonkernel logic.

You must preserve the design of all nonkernel logic. You can preserve your design in the Quartus Prime software via one of the following methods:

- Create a design partition containing all nonkernel logic under a single HDL hierarchy and then export the partition. For example, you may create and export a **board.qsys** Qsys subsystem (see figure below). The top-level **system.qsys** Qsys system can then instantiate this exported board Qsys subsystem.
- Implement the Configuration via Protocol (CvP) configuration scheme, which preserves all logic outside a design partition. In this case, you only need to create a partition around the kernel logic. You may place all nonkernel logic into a single top-level Qsys system file (for example, **system.qsys**).

You must design all the components and the **board_spec.xml** file that describe the system to the AOCL.

Figure 1-1: Example System Hierarchy with a Board Qsys Subsystem



1. **Creating the Board Qsys System** on page 1-6
   To create your board system in a Qsys subsystem, you may modify the **board.qsys** template in the Custom Platform Toolkit.
2. **Establishing Guaranteed Timing Flow** on page 1-9
   Deliver a design partition for nonkernel logic that has a clean timing closure flow as part of your Custom Platform.

Creating the Board Qsys System

When designing your board hardware, you have the option to create a Qsys subsystem within **system.qsys** that contains all the board logic. In addition to organizing your design code, having this subsystem allows you to create a Quartus Prime partition that you can preserve. To create your board system in a Qsys subsystem, you may modify the **board.qsys** template in the Custom Platform Toolkit.

An implementation of a board Qsys subsystem might include the following components:

- Proper reset sequencing
- Altera SDK for OpenCL-specific components
- Host-to-FPGA communication IP
- Memory IP used for AOCL global memory
- Streaming channels to board-specific interfaces

Refer to *The Board Qsys System* section for more information.

Templates of the following hardware design files are available in the ***ALTERAOCLSDKROOT*/board/custom_platform_toolkit/board_package/hardware/template** directory:

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

Template of the **post_flow.tcl** file is available in the ***ALTERAOCLSDKROOT*/board/custom_platform_toolkit/board_package/hardware/template/scripts** directory of the Custom Platform Toolkit.

To create nonkernel logic, perform the following tasks in the **system.qsys** top-level Qsys system or in a board Qsys subsystem:

1. In Qsys, add your host and memory IPs to the Qsys system, and establish all necessary connections and exports.

   Attention: You might need to acquire separate IP licenses. For a list of available licensed and unlicensed IP solutions, visit the All Intellectual Property page of the Altera website. For more information about each IP, click the link in the Product Name column to navigate to the product page.

   a. Connect your host interface clock such that it drives `por_reset_controller/clk`. Your design's global reset and clock inputs are fed to a reset counter (`por_reset_counter`). This reset counter then synchronizes to the host interface clock in the Merlin Reset Controller (`por_reset_controller`).

      The `por_reset_counter` ACL SW Reset component implements the power-on reset. It resets all the device hardware by issuing a reset for a number of cycles after the FPGA completes its configuration.

   b. Modify the parameters of the `pipe_stage_host_ctrl` Avalon® Memory-Mapped (Avalon-MM) Pipeline Bridge component such that it can receive requests from your host IP. Connect your host interface's Avalon-MM master port to the `s0` port of `pipe_stage_host_ctrl`. Connect the `m0` port of `pipe_stage_host_ctrl` to all the peripherals that must communicate with your host interface, including the OpenCL Kernel Clock Generator and the OpenCL Kernel Interface components.

   c. Adjust the number of `clock_cross_kernel_mem_<N>` Avalon-MM Clock Crossing Bridge components to match the number of memory interfaces on your board. This component performs clock crossing between the kernel and memory interfaces. Modify the parameters of each component so that they are consistent with the parameters of the OpenCL Memory Bank Divider component and the `interface` attribute described in **board_spec.xml**. Connect the `m0` master, clock, and reset ports of `clock_cross_kernel_mem_<N>` (that is, `m0`, `m0_clk`, and `m0_reset`, respectively) to your memory IP.

Important: Connect `m0_reset` in such a way that assertion of `kernel_reset` from the OpenCL Memory Bank Divider component triggers this reset.

2. Customize the AOCL-specific Qsys system components..

   Attention: If you use the **board.qsys** system template to create a Qsys subsystem, note that it is preconfigured with the necessary connections between the AOCL-specific system components and the appropriate interfaces exported to match the **board_spec.xml** file. Altera recommends that you preserve the preconfigured connections as much as possible.

   a. In Qsys, click **Tools** > **Options**. In the **Options** dialog box, add ***ALTERAOCLSDKROOT*/ip/board** to the **Qsys IP Search Path** and then click **Finish**.
   b. Instantiate the OpenCL Kernel Clock Generator component. Specify the component parameters, and connect the signals and ports as outlined in the *OpenCL Kernel Clock Generator* section.
   c. Instantiate the OpenCL Kernel Interface component. Specify the component parameters, and connect the signals and ports as outlined in the *OpenCL Kernel Interface* section.
   d. For each global memory type, instantiate the OpenCL Memory Bank Divider component. Specify the component parameters, and connect the signals and ports as outlined in the *OpenCL Memory Bank Divider* section.

      Attention: Set the parameters such that the resulting bank masters have the equivalent address bits and burst widths as those from the kernel, as defined in the `interface` attribute of the `global_mem` element in the **board_spec.xml** file. For each memory bank, Qsys generates a master that inherits the same characteristics as your specifications.

3. If you choose to create a Qsys subsystem for the nonkernel logic, export any necessary I/Os to the top-level **system.qsys** Qsys system.
4. Edit the top-level **top.v** file to instantiate **system.qsys** and connect any board-specific I/Os.
5. Set up the **top.qpf** Quartus Prime project with all the necessary settings for your board design.
6. Modify the **post_flow.tcl** file to include the Tcl code that generates the **fpga.bin** file during Quartus Prime compilation.

   The **fpga.bin** file is necessary for programming the board.
7. Edit the **board_spec.xml** file to include board-specific descriptions.

Related Information

- **All Intellectual Property page on the Altera website**
- **OpenCL Kernel Clock Generator** on page 2-2
- **OpenCL Kernel Interface** on page 2-3
- **OpenCL Memory Bank Divider** on page 2-4
- **The Board Qsys Subsystem** on page 2-1

General Quality of Results Considerations for the Exported Board Partition

When generating a post-place-and-route partition, take into account several design considerations for the exported board partition that might have unexpected consequences on the Altera SDK for OpenCL compilation results. The best approach to optimizing the board partition is to experiment with a range of different OpenCL kernels.

The list below captures some of the parameters that might impact the quality of AOCL compilation results:

- Resources Used

  Minimize the number of resources the partition uses to maximize the resources available for the OpenCL kernels.

- Kernel Clock Frequency

  Altera recommends that the kernel clock has a high clock constraint (for example, greater than 350 MHz for a Stratix V device). The amount of logic in the partition clocked by the kernel clock should be relatively small. This logic should not limit the kernel clock speed for even the simplest OpenCL kernels. Therefore, at least within the partition, the kernel clock should have a high clock constraint.

- Host-to-Memory Bandwidth

  The host-to-memory bandwidth is the transfer speed between the host processor to the physical memories on the accelerator card. To measure this memory bandwidth, compile and run the host application included with the Custom Platform Toolkit.

- Kernel-to-Memory Bandwidth

  The kernel-to-memory bandwidth is the maximum transfer speed possible between the OpenCL kernels and global memory.

  To measure this memory bandwidth, compile and run the host program included in the **/tests/boardtest/host** directory of the Custom Platform Toolkit.

- Fitter Quality of Results (QoR)

  To ensure that OpenCL designs consuming much of the device's resources can still achieve high clock frequencies, region-constrain the partition to the edges of the FPGA. The constraint allows OpenCL kernel logic to occupy the center of the device, which has the most connectivity with all other nodes.

  Test compile large designs to ensure that other Fitter-induced artifacts in the partition do not interfere with the QoR of the kernel compilations.

- Routability

  The routing resources that the partition consumes can affect the routability of a compiled OpenCL design. A kernel might use every digital signal processing (DSP) block or memory block on the FPGA; however, routing resources that the partition uses might render one of these blocks unroutable. This routing issue causes compilation of the Quartus Prime project to fail at the fitting step. Therefore, it is imperative that you test a partition with designs that use all DSP and memory blocks.

Establishing Guaranteed Timing Flow

Deliver a design partition for nonkernel logic that has a clean timing closure flow as part of your Custom Platform.

1. Create a placed and routed design partition using the incremental compilation feature of the Quartus Prime software. This is the design partition for nonkernel logic.

For more information on how to use the incremental compilation feature to generate a timing-closed design partition, refer to the *Quartus Prime Incremental Compilation for Hierarchical and Team-Based Design* chapter in Volume 1 of the *Quartus Prime Standard Edition Handbook*.

2. Import the post-fit partition from Step 1 into the top-level design as part of the compilation flow.
3. Run the ***ALTERAOCLSDKROOT*/ip/board/bsp/adjust_plls.tcl** script as a post-flow process, where *ALTERAOCLSDKROOT* points to the path of the Altera SDK for OpenCL installation.

   The **adjust_plls.tcl** script determines the maximum kernel clock frequency and stores it in the `pll_rom` on-chip memory of the OpenCL Kernel Clock Generator component.

Related Information

Quartus Prime Incremental Compilation for Hierarchical and Team-Based Design

Creating the Board XML Files

Your Custom Platform must include the XML files that describe your Custom Platform and each of your hardware system to the Altera SDK for OpenCL. You may create these XML files in simple text editors (for example, WordPad for Windows, and vi for Linux).

Creating the board_env.xml File on page 1-10
The **board_env.xml** file describes your Custom Platform to the AOCL.

Creating the board_spec.xml File on page 1-13
The **board_spec.xml** XML file contains metadata necessary to describe your hardware system to the Altera SDK for OpenCL.

Creating the board_env.xml File

For the Altera Offline Compiler to target a Custom Platform, the Altera SDK for OpenCL user has to set the environment variable *AOCL_BOARD_PACKAGE_ROOT* to point to the Custom Platform directory in which the **board_env.xml** file resides.
The **board_env.xml** file describes your Custom Platform to the AOCL.
Together with the other contents of the Custom Platform, the **board_env.xml** file sets up the board installation that enables the AOC to target a specific accelerator board.

A **board_env.xml** template is available in the **/board_package** directory of the Custom Platform Toolkit.

1. Create a `board_env` top-level XML element. Within `board_env`, include the following XML elements:
   - `hardware`
   - `platform`

   Include a `platform` element for each operating system that your Custom Platform supports.
2. Within each `platform` element, include the following XML elements:
   - `mmdlib`
   - `linkflags`
   - `linklibs`
   - `utilbindir`
3. Parameterize each element and corresponding attribute(s) with information specific to your Custom Platform, as outline in the table below:

Table 1-2: Specifications of XML Elements and Attributes in the board_env.xml File

Element	Attribute Description
`board_env`	`version`: The AOCL Custom Platform Toolkit release you use to create your Custom Platform. **Attention:** The Custom Platform version must match the AOCL version you use to develop the Custom Platform. `name`: Name of the board installation directory containing your Custom Platform.
`hardware`	`dir`: Name of the subdirectory, within the board installation directory, that contains the board variants. `default`: The default board variant that the AOC targets when the AOCL user does not specify an explicit argument for the `--board <board_name>` AOC option.
`platform`	`name`: Name of the operating system. Refer to the *Altera SDK for OpenCL Getting Started Guide* and the *Altera RTE for OpenCL Getting Started Guide* for more information.
`mmdlib`	A string that specifies the path to the MMD library of your Custom Platform. To load multiple libraries, specify them in an ordered, comma-separated list. The host application will load the libraries in the order that they appear in the list.
`linkflags`	A string that specifies the linker flags necessary for linking with the MMD layer available with the board. **Tip:** You can use `%a` to reference the AOCL installation directory and `%b` to reference your board installation directory.
`linklibs`	A string that specifies the libraries the AOCL must link against to use the MMD layer available with the board. **Note:** Include the alterahalmmd library, available with the AOCL, in this field because the library is necessary for all devices with an MMD layer.
`utilbindir`	Directory in which the AOCL expects to locate the AOCL utility executables (that is, `install`, `uninstall`, `program`, `diagnose` and `flash`). **Tip:** You can use `%a` to reference the AOCL installation directory and `%b` to reference your board installation directory.

Your **board_env.xml** file should resemble the following example:

```
<?xml version="1.0"?>
<board_env version="15.1" name="a10_ref">
  <hardware dir="hardware" default="a10_ref"></hardware>
  <platform name="linux64">
    <mmdlib>%b/linux64/lib/libaltera_a10_ref_mmd.so</mmdlib>
    <linkflags>-L%b/linux64/lib</linkflags>
    <linklibs>-lalterahalmmd -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>alterahalmmd.lib altera_a10_ref_mmd.lib</linklibs>
    <utilbindir>%b/windows64/libexec</utilbindir>
  </platform>
</board_env>
```

Related Information

- **Prerequisites for the Altera SDK for OpenCL**
- **Prerequisites for the Altera RTE for OpenCL**

Testing the board_env.xml File

After you generate the **board_env.xml** file, test the file within your board installation directory to ensure that the Altera® Offline Compiler recognizes the board installation.

1. Set the environment variable *AOCL_BOARD_TOOLKIT_ROOT* to point to the Custom Platform subdirectory in which your **board_env.xml** file resides.
2. At the command prompt, invoke the `aocl board-xml-test` command to verify that the Altera SDK for OpenCL can locate the correct field values.
   The AOCL generates an output similar to the one below:

```
board-path       = <path_to_a10_ref>
board-version    = 15.1
board-name       = a10_ref
board-default    = a10_ref
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. Invoke the `aoc --list-boards` command to verify that the AOC can identify and report the board variants in the Custom Platform.
   For example, if your Custom Platform includes two FPGA boards, the AOCL generates an output similar to the one below:

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

The last board installation test takes place when you use the AOC to generate a design for your board.

Related Information

- **Creating the board_env.xml File** on page 1-10
- **Testing the Hardware Design** on page 1-20

Creating the board_spec.xml File

The **board_spec.xml** XML file contains metadata necessary to describe your hardware system to the Altera SDK for OpenCL.

For detailed descriptions on the type of information you must include in the **board_spec.xml** file, refer to the *XML Elements, Attributes, and Parameters in the board_spec.xml File* section. A **board_spec.xml** template is available in the ***ALTERAOCLSDKROOT*/board/custom_platform_toolkit/board_package/hardware/template** directory of the Custom Platform Toolkit.

1. Structure the **board_spec.xml** file to include the following XML elements and attributes:

Table 1-3: XML Elements and Attributes Specified in the board_spec.xml File

Element	Attribute
`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. For the `board` element, specify the board version and the name of the accelerator board. The name of the board must match the name of the directory in which the **board_spec.xml** file resides.

   Important: The board version must match the AOCL version you use to develop the Custom Platform.

   Attention: The board name must contain a combination of only letters, numbers, underscores (_), hyphens (-), or periods (.) (for example: s5_net).
3. For the `device` element, perform the following steps to specify the name of the device model file.
   a. Navigate to the ***ALTERAOCLSDKROOT*/share/models/dm** directory, where *ALTERAOCLSDKROOT* points to the path to the AOCL installation. The directory contains a list of device models files that describe available FPGA resources on accelerator boards.
   b. If your device is listed in the **dm** directory, specify the `device_model` attribute with the name of the device model file. Proceed to Step 4.

For example, `device_model="5sgsed8k2f40c2_dm.xml"`

c. If your device is not listed in the **dm** directory, or if your board uses an FPGA that does not have a device model, create a new device model by performing the tasks described in Steps d to g:

d. Copy a device model from the ***ALTERAOCLSDKROOT*/share/models/dm** directory (for example, **5sgxma7h2fe35c2_dm.xml**).

e. Place your copy of the device model in the Custom Platform subdirectory in which your **board_spec.xml** file resides.

f. Rename the file, and modify the values to describe the part your board uses.

g. In the **board_spec.xml** file, update the `device_model` attribute of the `device` element with the name of your file.

4. For the `device` element, specify the parameters in the `used_resources` attribute to describe the FPGA resources that the board design consumes in the absence of any OpenCL kernel.

   If your design includes a defined partition around all the board logic, you can extract the data from the Partition Statistics section of the Fitter report.

5. For each global memory type, specify the following information:

   a. Name of the memory type.

   b. The combined maximum global memory bandwidth.

      You can calculate this bandwidth value from datasheets of your memories.

   c. The size of the data that the Altera Offline Compiler interleaves across memory banks.

      Note: `interleaved_bytes` = `burst_size` x `width_bytes`

   d. If you have a homogeneous memory system, proceed to Step e. If you have a heterogeneous memory system, for each global memory type, specify the `config_addr` attribute with the base address of the ACL Mem Organization Control Qsys component (mem_org_mode).

   e. If you choose to set a global memory type as default, assign a value of 1 to the optional `default` attribute.

      If you do not include this attribute, the first memory defined in the **board_spec.xml** file becomes the default memory.

   f. Specify the parameters in the `interface` attribute to describe the characteristics of each memory interface.

6. For the `host` element, specify the parameters in the `kernel_config` attribute to describe the offset at which the kernel resides. Determine the start of the offset from the perspective of the `kernel_cra` master in the OpenCL Kernel Interface Qsys component.

7. If your board provides channels for direct OpenCL kernel-to-I/O accesses, include the `channels` element for all channel interfaces. Specify the parameters in the `interface` attribute to describe the characteristics of each channel interface.

8. Include the `interfaces` element to describe the kernel interfaces connecting to and controlling OpenCL kernels. Include one of each interface types (that is `master`, `irq`, and `streamsource`).

   a. Specify the parameters in the `interface` attribute to describe the characteristics of each kernel interface.

      For the `streamsource` interface type, also specify the `clock` attribute with the name of the clock the snoop stream uses. Usually, this clock is the kernel clock.

      Important: Update the width of the snoop interface (`acl_internal_snoop`) specified with the `streamsource` kernel interface. Updating the width ensures that the `global_mem` interface entries in **board_spec.xml** match the characteristics of the `bank<N>` Avalon

Memory-Mapped (Avalon-MM) masters from corresponding OpenCL Memory Bank Divider component for the default memory.

b. Specify the parameters in the `kernel_clk_reset` attribute to include the exported kernel clock and reset interfaces as kernel interfaces.

9. Include the `compile` element and specify its attributes to control the Quartus Prime compilation, registration, and automigration.

Below is the XML code of an example **board_spec.xml** file:

```
<?xml version="1.0"?>
<board version="15.0" name="a10_ref">

  <compile project="top" revision="top" qsys_file="system.qsys" generic_kernel="1">
    <generate cmd="qsys-generate --synthesis=VERILOG system.qsys"/>
    <synthesize cmd="quartus_sh --flow compile top -c top"/>
    <auto_migrate platform_type="a10_ref" >
      <include fixes=""/>
      <exclude fixes=""/>
    </auto_migrate>
  </compile>

  <device device_model="10ax115s2f45i2sges_dm.xml">
    <used_resources>
      <alms num="6566"/> <!-- ALMs used in final placement - ALMs used for
registers -->
      <ffs num="20030"/>
      <dsps num="0"/>
      <rams num="112"/>
    </used_resources>
  </device>

  <!-- DDR4-2133 -->
  <global_mem name="DDR" max_bandwidth="17064" interleaved_bytes="1024"
config_addr="0x018">
    <interface name="board" port="kernel_mem0" type="slave" width="512"
maxburst="16" address="0x00000000" size="0x80000000" latency="240"/>
  </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>
```

Related Information

XML Elements, Attributes, and Parameters in the board_spec.xml File on page 2-6

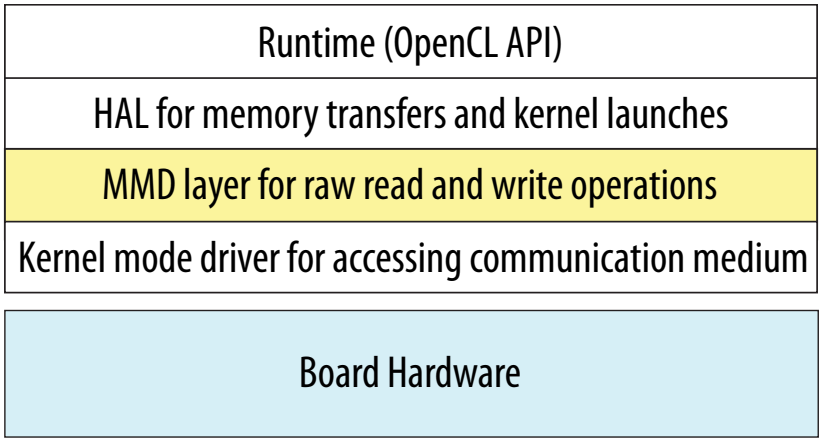
Creating the MMD Library

Your Custom Platform requires an MMD layer necessary for communication with the accelerator board.

You must implement a file I/O-like software interface such as open, read, write, and close to communicate with the accelerator board over any medium. The result of your implementation is a set of linker arguments that allows an OpenCL host application to link against the MMD layer of the target board. A dynamic link library (DLL) that fully implements the MMD layer is also necessary for the communication.

Figure 1-2: AOCL Software Architecture

This figure depicts the four layers of the Altera SDK for OpenCL software architecture: runtime, hardware abstraction layer (HAL), MMD layer, and kernel mode driver.



The following tasks outline the procedure for creating an MMD library for use with PCI Express® (PCIe®).

1.
2. Name a new library file that implements the MMD layer in the following manner:
   <*board_vendor_name*>_<*board_family_name*>[_<*unique_string*>]_mmd.<*a|so|lib|dll*>

   Where:

   <*board_vendor_name*> is the entity responsible for the accelerator board.
   <*board_family_name*> is the board family name that the library supports.
   <*unique_string*> is a designation that you create. Altera recommends that you include information such as revision and interface type.
   <*a|so|lib|dll*> is the file extension. It can be an archive file (**.a**), a shared object file (**.so**), a library file (**.lib**), or a dynamic link library file (**.dll**).

   Example library file name: **altera_svdevkit_pcierev1_mmd.so**
3. Include the ***ALTERAOCLSDKROOT*/board/custom_platform_toolkit/mmd/aocl_mmd.h** header file in the operating system-specific implementation of the MMD layer.

   The **aocl_mmd.h** file and the *MMD API Descriptions* reference section contain full details on the MMD application programming interface (API) descriptions, their arguments, and their return values.
4. Implement the MMD layer for your Custom Platform, and compile it into a C/C++ library.
   Example source codes of a functional MMD library are available in the **<*path_to_s5_net*>/source/host/mmd** directory of the Stratix V Network Reference Platform. In particular, the **acl_pcie.cpp** file implements the API functions defined in the **aocl_mmd.h** file.

If the AOCL users need to load a particular library at runtime, deliver the library in a directory that the operating system can locate. Instruct the AOCL users to add the library path to the *LD_LIBRARY_PATH* (for Linux) or *PATH* (for Windows) environment variable at runtime.

5. Modify the `mmdlib` and `linkflags` elements in the **board_env.xml** file by specifying the library flags necessary for linking with the MMD layer.

Related Information

MMD API Descriptions on page 2-14

Kernel Power-up State

The OpenCL kernel is an unknown state after you power-up your system or reprogram your FPGA. As a result, the MMD layer does not enable or respond to any interrupts from the kernel during these periods. The kernel is in a known state only after `aocl_mmd_set_interrupt_handler` is called. Therefore, enable interrupts from the kernel only after the handler becomes available to the MMD layer.

The general sequence of calls for a single host application is as follows:

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`

Setting Up the Altera Client Driver

The Altera SDK for OpenCL supports the Altera Client Driver (ACD) custom extension. The ACD allows the AOCL to automatically find and load the Custom Platform libraries at host runtime.

Attention: To allow AOCL users to use the ACD, you must remove the MMD library from the `linklibs` element in the **board_env.xml** file.

Enumerating the Custom Platform ACD on Windows

Specify the Custom Platform libraries in the registry key **HKEY_LOCAL_MACHINE\SOFTWARE\Altera\OpenCL\Boards**. Enter one value for each library. Each value must include the path to the library as the string value, and a **dword** setting of 0.

For example:

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

To enumerate Custom Platform ACDs on Windows, the ACD Loader scans the value in the registry key **HKEY_LOCAL_MACHINE\SOFTWARE\Altera\OpenCL\Boards**. For each value in this key, the **dword** parameter specifies the path to a DLL and a numerical data value. If the **dword** data is 0, the Installable Client Driver (ICD) Loader attempts to open the corresponding DLL. If the DLL is an MMD library, then the AOCL attempts to open any board that is associated with that library.

In this case, the ACD opens the library **c:\\board_vendor a\\my_board_mmd.dll**.

If the registry key specifies multiple libraries, the Loader loads the libraries in the order that they appear in the key. If there is an order dependency between the libraries available with your Custom Platform, ensure that you list the libraries accordingly in the registry key.

Enumerating the Custom Platform ACD on Linux

Enter the absolute paths of Custom Platform libraries in an **.acd** file. Store the **.acd** file in the **/opt/Altera/OpenCL_boards/** directory.

To enumerate Custom Platform ACDs on Linux, the ACD Loader scans the files with the extension **.acd** in the path **/opt/Altera/OpenCL_boards/**. The ACD Loader opens each **.acd** file in this path as a text file. Each **.acd** file should contain the absolute path to every library in the Custom Platform, one library per line. The ICD Loader attempts to open each library. If the library is an MMD library, then the AOCL attempts to open any board that is associated with that library.

For example, consider the file **/opt/Altera/OpenCL_boards/PlatformA.acd**. If it contains the line **/opt/PlatformA/libPlatformA_mmd.so**, the ACD Loader loads the library **/opt/PlatformA/libPlatformA_mmd.so**.

If the **.acd** file specifies multiple libraries, the Loader loads the libraries in the order that they appear in the file. If there is an order dependency between the libraries available with your Custom Platform, ensure that you list the libraries accordingly in the **.acd** file.

For more information on how AOCL users link their host applications to the ICD and ACD, refer to the *Linking Your Host Application to the Khronos ICD Loader Library* section in the *Altera SDK for OpenCL Programming Guide*.

Related Information

Linking Your Host Application to the Khronos ICD Loader Library

Providing AOCL Utilities Support

Each Custom Platform you develop for use with the Altera SDK for OpenCL must support a set of AOCL utilities. These utilities enable users to manage the accelerator board through the AOCL.

If you create a new Custom Platform, perform the following tasks to create executables of the AOCL utilities and then store them in the **utilbindir** directory of your Custom Platform:

Tip: Within the ***<path_to_s5_net>*/source/util** directory of the Stratix V Network Reference Platform, you can find source code for the `program` and `flash` utilities in the **reprogram** and **flash** subdirectories, respectively. Scripts for the `install` and `uninstall` utilities are available in the ***<path_to_s5_net>*/*<OS_platform>*/libexec** directory.

You can find the source code for the `diagnose` utility in the ***<path_to_a10_ref>*/source/util/diagnostic** directory within the Arria® 10 FPGA Development Kit Reference Platform. Contact your Altera field application engineer or technical account manager for more information.

1. Create an `install` utility executable that sets up the current host computer with the necessary drivers to communicate with the board via the MMD layer. The `install` utility takes no argument. For example, the PCIe-based MMD might need to install PCIe drivers into the host operating system.

   Executable call: `aocl install`
2. Create an `uninstall` utility executable that removes the current host computer drivers (for example, PCIe drivers) used for communicating with the board. The `uninstall` utility takes no argument.

   Executable call: `aocl uninstall`
3. Create a `diagnose` utility executable that confirms the board's integrity and the functionality of the MMD layer.

The `diagnose` utility must support the following internal calling modes:

Calling Mode	Description
`-probe`	Prints all available devices in a Custom Platform. For a given hardware configuration, the utility lists the devices in the same order, and each device is associated with the same identification string each time.
`-probe <device_name>`	Queries the specified device and prints summary statistics abou the device.
`<device_name>` where <*device name*> is the string that corresponds to the FPGA device.	Performs a full diagnostic test for the specified device. The utility generates the message `DIAGNOSTIC_PASSED` as the output. Otherwise, the utility generates the message `DIAGNOSTIC_FALIED`.

When users invoke the `diagnose` utility command without an argument, it queries the devices in the Custom Platform and supplies a list of valid <*device_name*> strings assigned to the list of devices.

Executable call without argument: `aocl diagnose`

When users invoke the `diagnose` utility command with a <*device_name*> argument, the utility runs your diagnostic test for the board. A user may give a board a different logical device name than the physical device name associated with the Custom Platform. The `aocl` utility simply converts the user-side logical device name to the Custom Platform-side physical device name. If the diagnostic test runs successfully, the utility generates the message `DIAGNOSTIC_PASSED` as the output. Otherwise, the utility generates the message `DIAGNOSTIC_FALIED`.

Executable call with argument: `aocl diagnose <device_name>`.

4. Create a `program` utility executable that receives the **fpga.bin** file and configures that design onto the FPGA. Although the main method for FPGA programming is via the host and the MMD, make this utility available to users who do not have a host system or who perform offline reprogramming.

   The `program` utility command takes <*device_name*> and an Altera Offline Compiler Executable file name (**<*kernel_filename*>.aocx**) as arguments. When users invoke the command, the AOCL extracts the **fpga.bin** file and passes it to the `program` utility.

   Important: Altera highly recommends that the `program` utility links with and calls the `aocl_mmd_reprogram` function implemented in the MMD layer. Refer to the *aocl_mmd_reprogram* and *Reprogram Support* reference sections for more information.

   Executable call: `aocl program <device_name> <kernel_filename>.aocx`, where <*device name*> is the acl number that corresponds to the FPGA device.

5. Create a `flash` utility executable that receives the **fpga.bin** file and programs that design into the flash memory on the board. The `flash` utility command takes <*device_name*> and a **.aocx** file name as arguments. When users invoke the command, the AOCL extracts the **fpga.bin** file and passes it to the `flash` utility.

   Executable call: `aocl flash <device_name> <kernel_filename>.aocx`, where <*device name*> is the acl number that corresponds to the FPGA device.

When users invoke a utility command, the utility probes the current Custom Platform's **board_env.xml** file and executes the **<*utility_executable*>** file within the **utilbindir** directory.

Related Information

- **Creating the board_env.xml File** on page 1-10
- **Reprogram Support** on page 2-25
- **aocl_mmd_reprogram** on page 2-25

Testing the Hardware Design

After you create the software utilities and the MMD layer, and your hardware design achieves timing closure, test the design.

To test the hardware design, perform the following tasks:

1. Navigate to the **boardtest.cl** OpenCL kernel within the ***ALTERAOCLSDKROOT*/board/custom_platform_toolkit/tests/boardtest** directory.

   ALTERAOCLSDKROOT points to the location of the Altera SDK for OpenCL installation.
2. Compile your kernel to generate an Altera Offline Compiler Executable file (**.aocx**) by invoking the `aoc --no-interleaving default boardtest.cl` command.
3. Program the accelerator board by invoking the `aocl program acl0 boardtest.aocx` command.
4. Invoke the commands `aocl compile-config` and `aocl link-config`. Confirm they include flags necessary for your MMD layer to compile and link successfully.
5. Build the **boardtest** host application.

   - For Windows systems, you may invoke the `make` command or use Microsoft Visual Studio.

     If you invoke the `make` command, the **Makefile** is located in the ***ALTERAOCLSDKROOT*/board/custom_platform_toolkit/tests/boardtest** directory.

     If you build your host application in Microsoft Visual Studio, the **boardtest.sln** and **main.cpp** files are located in the ***ALTERAOCLSDKROOT*/board/custom_platform_toolkit/tests/boardtest/host** directory.
   - For Linux systems, invoke the `make -f Makefile.linux` command.

     The **Makefile.linux** file is located in the ***ALTERAOCLSDKROOT*/board/custom_platform_toolkit/tests/boardtest** directory.
6. Run the **boardtest** executable.

Attention: To ensure that your hardware design produces consistent performance, you might have to test it using multiple OpenCL kernels in addition to **boardtest.cl**.

To qualify as an Altera preferred board, rigorous testing across multiple boards is necessary. Specifically, you should perform overnight testing of all Custom Platform tests and executes the AOCL example designs on multiple boards . All board variants within a Custom Platform must go through the testing process.

Applying for the Altera Preferred Board Status

Registering your Custom Platform and the supported FPGA boards in the Altera Preferred Board Partner Program allows them to benefit from ongoing internal testing across versions of the Quartus Prime Design Suite. Altera-tested Custom Platforms and boards are more likely to be forward compatible with future QPDS versions.

For your Custom Platform and the supported FPGA boards to achieve the Altera Preferred Board status, you must generate the following data and submit it to Altera:

1. The output from invoking the `aocl board-xml-test` command.
2. The output from invoking the `aoc --list-boards` command.
3. The outputs from the host compilation, host execution, and all Quartus Prime report files (**.rpt**). Also, for each board in your Custom Platform, the **acl_quartus_report.txt** file from the following tests:
   a. All tests included in the ***ALTERAOCLSDKROOT*/board/custom_platform_toolkit/tests** directory, where *ALTERAOCLSDKROOT* points to location of the Altera SDK for OpenCL installation.
   b. Compilations of the following examples on the OpenCL Design Examples page of the Altera website:
      1. Vector Addition
      2. Matrix Multiplication
      3. FFT (1D)
      4. FFT (2D)
      5. Sobel Filter
      6. Finite Difference Computation (3D)
4. For each board in the Custom Platform, a summary of the following:
   a. `HOST-TO-MEMORY BANDWIDTH` as reported by the boardtest test in the Custom Platform Toolkit (**/tests/boardtest**).
   b. `KERNEL-TO-MEMORY BANDWIDTH` as reported by the boardtest test.
   c. `Throughput` in swap-and-execute(s) reported by the swapper test in the Custom Platform Toolkit (**/tests/swapper**).
   d. `Actual clock freq` as reported in the **acl_quartus_report.txt** file from the blank test in the Custom Platform Toolkit (***ALTERAOCLSDKROOT*/board/custom_platform_toolkit/tests/blank**).

   Important: Use global routing to reduce consumption of local routing resources. Using global routing is necessary because it helps meet timing and improve kernel performance (Fmax). Use global or regional routing for any net with fan-out greater than 5000, and for kernel clock, 2x clock and reset. Check the Non-Global High Fan-Out Signals report in the Resource subsection, which is under the Fitter section of the Compilation Report.
5. Submit the necessary number of boards to Altera for in-house regression testing. Regression testing tests the out-of-the-box experience for each board variant on each operating system that your Custom Platform supports. Ensure that you test the procedure outlined below before you submit your boards:
   a. Install the board into the physical machine.
   b. Boot the machine and invoke the `aocl install` utility command.
   c. Invoke the `aocl diagnose` command.
   d. Run the AOCL test programs. The tester can also invoke the `aocl program <device_name> <kernel_filename>.cl` command to verify the functionality of the `program` utility.

Related Information

OpenCL Design Examples page on the Altera website

Shipping Recommendations

Before shipping your Altera-verified board to Altera SDK for OpenCL users, program the flash memory of the board with the hello_world OpenCL design example. Programming the flash memory of the board with the **hello_world.aocx** hardware configuration file allows the AOCL user to power on the board and observe a working kernel.

Download the hello_world OpenCL design example from the OpenCL Design Examples page on the Altera website.

For more information, refer to the **README.txt** file available with the hello_world OpenCL design example and the *Programming the Flash Memory of an FPGA* sections in the *Altera SDK for OpenCL Getting Started Guide*.

Related Information

- **OpenCL Design Examples on the Altera website**
- **Programming the Flash Memory of an FPGA on Windows**
- **Programming the Flash Memory of an FPGA on Linux**

Document Revision History

Table 1-4: Document Revision History of the AOCL Custom Platform Design Chapter of the Altera SDK for OpenCL Custom Platform Toolkit User Guide

Date	Version	Changes
November 2015	2015.11.02	• Changed instances of *Quartus II* to *Quartus Prime*. • Changed instances of *Altera Complete Design Suite* to *Quartus Prime Design Suite*. • Updated the support requirement for the `diagnose` utility in the *Providing AOCL Utilities Support* section. • In the *Creating the board_env.xml File* section, added the `mmdlib` XML element to the list of elements included in the **board_env.xml** file.
May 2015	15.0.0	• Added the *Setting Up the Altera Client Driver* section.

Date	Version	Changes
December 2014	14.1.0	• Specified that the Custom Platform Toolkit is available in the ***ALTERAOCLSDKROOT*/board** directory. • Added the `uninstall` utility executable in the *Providing AOCL Utilities Support* section. • Indicated that the version attributes in the **board_env.xml** and **board_spec.xml** files have to match the Altera Complete Design Suite and Altera SDK for OpenCL version you use to develop the Custom Platform. • Added instruction for including the `compile` eXtensible Markup Language element and its associated attributes in the **board_spec.xml** file in the section *Creating the board_spec.xml File*. • Added information on the automigration of Custom Platform in sections *Custom Platform Automigration* and *Customizing Automigration*. • Removed the *Generating the Rapid Prototyping Library* section.
October 2014	14.0.1	• Reorganized existing document into two chapters.
June 2014	14.0.0	• Initial release.

Altera SDK for OpenCL Custom Platform Toolkit Reference Material

2

2015.11.02

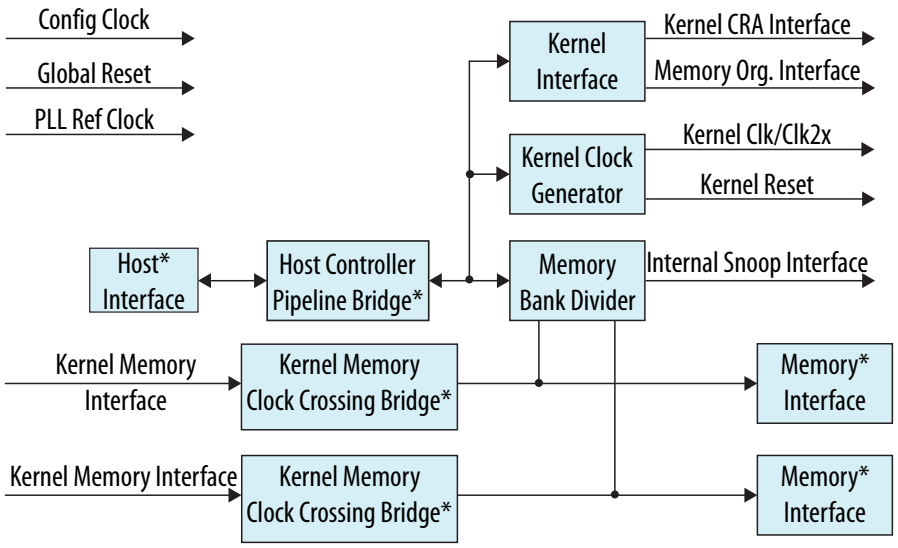
UG-OCL007 Subscribe Send Feedback

The *Altera SDK for OpenCL Custom Platform Toolkit Reference Material* chapter provides supplementary information that can assist you in the implementation of your Custom Platform.

The Board Qsys Subsystem

When designing your board hardware, you have the option to create a Qsys subsystem within the top-level Qsys system (**system.qsys**) that contains all nonkernel logic.
The board Qsys subsystem is the main design entry point for a new accelerator board. It is the location where the instantiations of the OpenCL host and global memory interfaces occur. Your board design must have a minimum of 128 kilobytes (KB) of external memory. Any Avalon Memory-Mapped (Avalon-MM) slave interface (for example, a block RAM) can potentially be a memory interface.

The diagram below represents a board system implementation in more details:



Note: Blocks denoted with an asterisk (*) are blocks that you have to add to the board Qsys subsystem.

The OpenCL host communication interface and global memory interface are the main components of the board system. The memory-mapped device (MMD) layer communicates, over some medium, with the intellectual property (IP) core instantiated in this Qsys system.

For example, an MMD layer executes on a PCI Express (PCIe)-based host interface, and the host interface generates Avalon interface requests from an Altera PCIe endpoint on the FPGA.

Within the board Qsys subsystem, you can also define the global memory system available to the OpenCL kernel. The global memory system may consist of different types of memory interfaces. Each memory type may consist of one, two, four, or eight banks of physical memory. All the banks of a given memory type must be the same size in bytes and have equivalent interfaces. If you have streaming I/O, you must also include the corresponding IP in the board Qsys system. In addition, you must update the **board_spec.xml** file to describe the channel interfaces.

Altera SDK for OpenCL-Specific Qsys System Components

The Qsys system for your board logic includes components specific to the Altera SDK for OpenCL (AOCL) that are necessary for implementing features that instantiate host communication and global memory interfaces.

The board Qsys system must export an Avalon-MM master for controlling OpenCL kernels. It must also export one or more Avalon-MM slave ports that the kernels use as global memory interfaces. The ***ALTERAOCLSDKROOT*/ip/board** directory of the AOCL includes a library that contains AOCL-specific Qsys system components, where *ALTERAOCLSDKROOT* points to the location of the AOCL installation. These components are necessary for implementing features such as Avalon-MM interfaces, organizing programmable banks, cache snooping, and supporting Altera's guaranteed timing closures.

1. **OpenCL Kernel Clock Generator** on page 2-2
   The OpenCL Kernel Clock Generator is a Qsys component that generates a clock output and a clock 2x output for use by the OpenCL kernels.
2. **OpenCL Kernel Interface** on page 2-3
   The OpenCL Kernel Interface is a Qsys component that allows the host interface to access and control the OpenCL kernel.
3. **OpenCL Memory Bank Divider** on page 2-4
   The OpenCL Memory Bank Divider is a Qsys component that takes an incoming request from the host interface on the Avalon-MM slave port and routes it to the appropriate bank master port.

OpenCL Kernel Clock Generator

The OpenCL Kernel Clock Generator is a Qsys component that generates a clock output and a clock 2x output for use by the OpenCL kernels. An Avalon-MM slave interface allows reprogramming of the phase-locked loops (PLLs) and kernel clock status information.

Table 2-1: Parameter Settings for the OpenCL Kernel Clock Generator Component

Parameter	Description
REF_CLK_RATE	Frequency of the reference clock that drives the kernel PLL (that is, `pll_refclk`).
KERNEL_TARGET_CLOCK_RATE	Frequency that the Quartus Prime software attempts to achieve during compilation. Keep this parameter at its default setting.

Table 2-2: Signals and Ports for the OpenCL Kernel Clock Generator Component

Signal or Port	Description
`pll_refclk`	The reference clock for the kernel PLL. The frequency of this clock must match the frequency you specify for the **REF_CLK_RATE** component parameter.
`clk`	The clock used for the host control interface. The clock rate of `clk` can be slow.
`reset`	The reset signal that resets the PLL and the control logic. Resetting the PLL disables the kernel clocks temporarily. Connect this reset signal to the power-on reset signal in your system.
`ctrl`	The slave port used to connect to the OpenCL host interface and to adjust the frequency based on the OpenCL kernel.
`kernel_clk` `kernel_clk2x`	The kernel clock and its 2x variant that runs on twice the speed. The `kernel_clk2x` signal is directly exported from this interface. Because `kernel_clk` has internal Qsys connections, export it using a clock source component. You can also use the clock source to export the kernel reset. In addition, clock all logic at the board Qsys system interface with `kernel_clk`, except for any I/O that you add.
`kernel_pll_locked`	(Optional) If the PLL is locked onto the reference clock, the value of this signal is 1. The host interface manages this signal normally; however, this signal is made available in the board Qsys system.

OpenCL Kernel Interface

The OpenCL Kernel Interface is a Qsys component that allows the host interface to access and control the OpenCL kernel.

Table 2-3: Parameter Settings for the OpenCL Kernel Interface Component

Parameter	Description
Number of global memory systems	Number of global memory types in your board design.

Table 2-4: Signals and Ports for the OpenCL Kernel Interface Component

Signal or Port	Description
`clk`	The clock input used for the host control interface. The clock rate of `clk` can be slow.
`reset`	This reset input resets the control interface. It also triggers the `kernel_reset` signal, which resets all kernel logic.
`kernel_ctrl`	Use this slave port to connect to the OpenCL host interface. This interface is a low-speed interface with which you set kernel arguments and start the kernel's execution.
`kernel_clk`	The `kernel_clk` output from the OpenCL Kernel Clock Generator drives this clock input.

Signal or Port	Description
kernel_cra	This Avalon-MM master interface communicates directly with the kernels generated by the Altera Offline Compiler (AOC). Export the Avalon-MM interface to the OpenCL Kernel Interface and name it in the **board_spec.xml** file.
sw_reset_in	When necessary, the OpenCL host interface resets the kernel via the kernel_ctrl interface. If the board design requires a kernel reset, it can do so via this reset input. Otherwise, connect the interface to a global power-on reset.
kernel_reset	Use this reset output to reset the kernel and any other hardware that communicates with the kernel. **Warning:** This reset occurs between the MMD open and close calls. Therefore, it must not reset anything necessary for the operation of your MMD.
sw_reset_export	This reset output is the same as kernel_reset, but it is synchronized to the clk interface. Use this output to reset logic that is not in the kernel_clk clock domain but should be reset whenever the kernel resets.
acl_bsp_memorg_host	The memory interfaces use these signals. Based on the number of global memory systems you specify in the OpenCL Kernel Interface component parameter editor, the Quartus Prime software creates the corresponding number of copies of this signal, each with a different hexadecimal suffix. Connect each signal to the OpenCL Memory Bank Divider component associated with each global memory system (for example, DDR). Then, list the hexadecimal suffix in the config_addr attribute of the global_mem element in the **board_spec.xml** file.
kernel_irq_from_kernel	An interrupt input from the kernel. This signal will be exported and named in the **board_spec.xml** file.
kernel_irq_to_host	An interrupt output from the kernel. This signal will connect to the host interface.

OpenCL Memory Bank Divider

The OpenCL Memory Bank Divider is a Qsys component that takes an incoming request from the host interface on the Avalon-MM slave port and routes it to the appropriate bank master port. This component must reside on the path between the host and the global memory interfaces. In addition, it must reside outside of the path between the kernel and the global memory interfaces.

Table 2-5: Parameter Settings for the OpenCL Memory Bank Divider Component

Parameter	Description
Number of banks	Number of memory banks for each of the global memory types included in your board system.
Separate read/write ports	Enable this parameter so that each bank has one port for read operation and one for write operation.

Parameter	Description
Add pipeline stage to output	Enable this parameter to allow for potential timing improvements.
Data Width	Width of the data bus to the memory in bits.
Address Width (total addressable)	Total number of address bits necessary to address all global memory.
Burst size (maximum)	The `maxburst` value defined in the `interface` attribute of the `global_mem` element in the **board_spec.xml** file.
Maximum Pending Reads	Maximum number of pending read transfers the component can process without asserting a `waitrequest` signal. **Caution:** A high **Maximum Pending Reads** value causes Qsys to insert a deep response FIFO buffer, between the component's master and slave, that consumes a lot of device resources. It also increases the achievable bandwidth between host and memory interfaces.

Table 2-6: Signals and Ports for the OpenCL Memory Bank Divider Component

Signal or Port	Description
`clk`	The bank divider logic uses this clock input. If the IP of your host and memory interfaces have different clocks, ensure that `clk` clock rate is not slower than the slowest of the two IP clocks.
`reset`	The reset input that connects to the board power-on reset.
`s`	The slave port that connects to the host interface controller.
`kernel_clk`	The `kernel_clk` output from the OpenCL Kernel Clock Generator drives this clock input.
`kernel_reset`	The `kernel_reset` output from the OpenCL Kernel Interface drives this reset input.

Signal or Port	Description
acl_bsp_snoop	Export this Avalon Streaming (Avalon-ST) source. In the **board_spec.xml** file, under interfaces, describe only the snoop interface for the default memory (acl_internal_snoop). If you have a heterogeneous memory design, perform these tasks only for the OpenCL Memory Bank Divider component associated with the default memory. **Important:** The memory system you build in Qsys alters the width of acl_bsp_snoop. You must update the width of the streamsource interface within the channels element in the **board_spec.xml** file to match the width of acl_bsp_snoop. **Important:** In the **board_spec.xml** file, update the width of the snoop interface (acl_internal_snoop) specified with the streamsource kernel interface within the interfaces element. Updating the width ensures that the global_mem interface entries in **board_spec.xml** match the characteristics of the bank<*N*> Avalon-MM masters from corresponding OpenCL Memory Bank Divider component for the default memory.
acl_bsp_memorg_host	This conduit connects to the acl_bsp_memorg_host interface of the OpenCL Kernel Interface.
bank1, bank2, ..., bank8	The number of memory masters available in the OpenCL Memory Bank Divider depends on the number of memory banks that were included when the unit was instantiated. Connect each bank with each memory interface in the same order as the starting address for the corresponding kernel memory interface specified in the **board_spec.xml** file. For example, global_mem interface that begins at address 0 must correspond to the memory master in bank1 from the OpenCL Memory Bank Divider.

Related Information

- **channels** on page 2-10
- **interfaces** on page 2-11
- **global_mem** on page 2-8
- **OpenCL Kernel Interface** on page 2-3

XML Elements, Attributes, and Parameters in the board_spec.xml File

This section describes the metadata you must include in the **board_spec.xml** file.

board on page 2-7
The board element of the **board_spec.xml** file provides the version and the name of the accelerator board.

board

The `board` element of the **board_spec.xml** file provides the version and the name of the accelerator board.

Example eXtensible Markup Language (XML) code:

```
<board version="15.0" name="a10_ref">
...
</board>
```

Table 2-7: Attributes for the board Element

Attribute	Description
`version`	The version of the board. The board version must match the version of the Quartus Prime software you use to develop the Custom Platform.
`name`	The name of the accelerator board, which must match the name of the directory in which the **board_spec.xml** file resides. The name must contain a combination of only letters, numbers, underscores (_), hyphens (-), or periods (.) (for example, s5_net).

device

The `device` element of the **board_spec.xml** file provides the device model and the resources that the board design uses.

Example XML code:

```
<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>
```

Table 2-8: Attributes for the device Element

Attribute	Description
`device_model`	The file name of the device model file that describes the available FPGA resources on the accelerator board.
`used_resources`	Reports the number of adaptive logic modules (ALMs), flip-flops, digital signal processor (DSP) blocks and RAM blocks that the board design consumes, in the absence of any kernel, to the Altera SDK for OpenCL. If you create a defined partition around all the board logic, you can obtain the used resources data from the Partition Statistics section of the Fitter report. Extract the information from the following parameters: • `alms num`—The number of logic ALMs used, excluding the number of ALMs with only their registers used. The value should correspond to [a]+[b]+[d] from part [A] of the Fitter Partition Statistics. • `ffs num`—The number of flip flops. • `dsps num`—The number of DSP blocks. • `rams num`—The number of RAM blocks.

global_mem

The `global_mem` and `interface` elements of the **board_spec.xml** file provides information on the memory interfaces that connect to the kernel.

Example XML code:

```
<!-- 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>
```

Note: For each global memory that the kernel accesses, you must include one `interface` element that describes its characteristics.

Table 2-9: Attributes for the global_mem Element

Attribute	Description
`name`	The name the Altera SDK for OpenCL user uses to identify the memory type. Each name must be unique and must comprise of less than 32 characters.
`max_bandwidth`	The maximum bandwidth of all global memory interfaces combined in their current configuration. The Altera Offline Compiler uses `max_bandwidth` to choose an architecture suitable for the application and the board. Compute this bandwidth value from datasheets of your memories.
`interleaved_bytes`	The AOC currently can interleave data across banks no finer than the size of one full burst. This attribute specifies this size in bytes, which is generally computed by `burst_size` x `width_bytes`. The `interleaved_bytes` value must be the same for the host interface and the kernels. Therefore, the configuration of the AOCL Memory Bank Divider must match the exported kernel slave interfaces in this respect.
`config_addr`	The address of the ACL Mem Organization Control Qsys component (mem_org_mode) that the host software uses to configure memory. You may omit this attribute if your board has homogeneous memory; the software will use the default address (0x18) for this component. If your board has heterogeneous memory, there is a mem_org_mode component in the board system for each memory type. Enter the `config_addr` attribute and set it to the value of the base address of the mem_org_mode component(s).

Attribute	Description
`default`	Include this optional attribute and assign a value of 1 to set the global memory as the default memory interface. If you do not implement this attribute, the first memory type defined in the **board_spec.xml** file becomes the default memory interface.
`interface`	See the *interface* section for the parameters you must specify for each interface.

Related Information

interface on page 2-12

host

The `host` element of the **board_spec.xml** file provides information on the interface from the host to the kernel.

Example XML code:

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

Table 2-10: Attributes for the host Element

Attribute	Description
`kernel_config`	This attribute informs the Altera Offline Compiler at what offset the kernel resides, from the perspective of the `kernel_cra` master on the `kernel_interface` module. `start`: the starting address of the kernel. Normally, this attribute has a value of 0 because the `kernel_cra` master should not master anything except kernels. `size`: keep this parameter at the default value of `0x0100000`.

channels

The Altera SDK for OpenCL supports data streaming directly between kernels and I/O via explicitly named *channels*. Include the `channels` element in the **board_spec.xml** file if your accelerator board provides channels for direct kernel-to-I/O accesses. For the `channels` element, you must identify all the channel interfaces, which are implemented using the Avalon-ST specification. Specify each channel interface via the `interface` attribute. Refer to the *interface* section for the parameters you must specify for each interface. The channel interface only supports data, and valid and ready Avalon-ST signals. The I/O channel defaults to 8-bit symbols and big-endian ordering at the interface level.

Example XML code:

```
<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>
```

Related Information

interface on page 2-12

interfaces

The `interfaces` element of the **board_spec.xml** file describes the kernel interfaces which will connect to OpenCL kernels and control their behaviors. For this element, include one of each interface of types `master`, `irq` and `streamsource`. Refer to the *interface* section for the parameters you must specify for each interface.

Example XML code:

```
<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>
```

In addition to the `master`, `irq`, and `streamsource` interfaces, if your design includes a separate Qsys subsystem containing the board logic, the kernel clock and reset interfaces exported from it are also part of the `interfaces` element. Specify these interfaces with the `kernel_clk_reset` attribute and its corresponding parameters.

Table 2-11: Parameters for the kernel_clk_reset Attribute

Important: Name the kernel clock and reset interfaces in the Qsys connection format (that is, *<instance_name>.<interface_name>*).

For example: `board.kernel_clk`

Attribute	Description
`clk`	The Qsys name for the kernel clock interface. The `kernel_clk` output from the OpenCL Kernel Clock Generator component drives this interface.
`clk2x`	The Qsys name for the kernel clock interface. The `kernel_clk2x` output from the OpenCL Kernel Clock Generator component drives this interface.
`reset`	The Qsys connection for the kernel reset. The `kernel_reset` output from the OpenCL Kernel Interface component drives this interface.

Related Information

interface on page 2-12

interface

In the **board_spec.xml** file, each global memory, channel or kernel interface is comprised of individual interfaces. For the `global_mem`, `channels`, and `interfaces` XML elements, include an `interface` attribute for each interface and specify the corresponding parameters.

Table 2-12: Parameters for the interface XML Attribute

Parameter	Applicable Interface	Description
`name`	All	For `global_mem`: name of the Qsys component for the slave interface of the memory. For `channels`: instance name of the Qsys component that has the channel interface. For `interfaces`: name of the entity in which the kernel interface resides (for example, `board`).
`port`		For `global_mem`: name of the Qsys component for the slave interface of the memory. For `channels`: name of the streaming interface in the Qsys component. For `interfaces`: name of the interface to the OpenCL Kernel Interface Qsys component. For example, `kernel_cra` is the Avalon-MM interface, and `kernel_irq` is an interrupt.
`type`		For `global_mem`: set to `slave`. For `channels`: • Set to `streamsource` for a stream source that provides data to the kernel. • Set to `streamsink` for a stream sink interface that consumes data from the kernel. For `interfaces`: set to either `master`, `irq`, or `streamsource`.
`width`		For `global_mem`: width of the memory interface in bits. For `channels`: number of bits in the channel interface. Valid data widths are 8, 16, 32, 64, 128, 256 and 512 bits. For `interfaces`: width of the kernel interface in bits.

Parameter	Applicable Interface	Description
maxburst	global_mem	Maximum burst size for the slave interface. **Attention:** The value of width ÷ 8 x maxburst must equal to the value of interleaved_bytes.
address		Starting address of the memory interface that corresponds to the host interface-side address. For example, address 0 should correspond to the bank1 memory master from the OpenCL Memory Bank Divider. In addition, any non-zero starting address must abut the end address of the previous memory.
size		Size of the memory interface in bytes. The sizes of all memory interfaces should be equal.
latency		An integer specifying the time in nanoseconds (ns) for the memory interface to respond to a request. The latency is the round-trip time from the kernel issuing the board system a memory read request to the memory data returning to the kernel. For example, the Altera DDR3 memory controller running at 200 MHz with clock-crossing bridges has a latency of approximately 240 ns.
latency_type		If the memory interface has variable latency, set this parameter to average to signify that the specified latency is considered the average case. If the complete kernel-to-memory path has a guaranteed fixed latency, set this parameter to fixed.
chan_id	channels	A string used to identify the channel interface. The string may have up to 128 characters.
clock	interfaces	For the streamsource kernel interface type, the parameter specifies the name of the clock that the snoop stream uses. Usually, this clock is the kernel clock.

compile

The compile element of the **board_spec.xml** file and its associated attributes and parameters describe the general control of Quartus Prime compilation, registration, and automigration.

Example XML code:

```
<compile project="top" revision="top" qsys_file="system.qsys" generic_kernel="1">
  <generate cmd="qsys-generate --synthesis=VERILOG system.qsys"/>
  <synthesize cmd="quartus_sh --flow compile top -c top"/>
  <auto_migrate platform_type="a10_ref" >
    <include fixes=""/>
    <exclude fixes=""/>
  </auto_migrate>
</compile>
```

Attribute	Description
`project`	Name of the Quartus Prime project file (**.qpf**) that the Quartus Prime software intends to compile.
`revision`	Name of the revision within the Quartus Prime project that the Quartus Prime software compiles to generate the Altera Offline Executable file (**.aocx**) .
`qsys_file`	Name of the Qsys file into which the OpenCL kernel is embedded.
`generic_kernel`	Set this value to 1 if you want the Altera Offline Compiler to generate a common Verilog interface for all OpenCL compilations. This setting is necessary in situations where you must set up Quartus Prime design partitions around the kernel, such as in the Configuration via Protocol (CvP) flow.
`generate_cmd`	Command required to prepare for full compilation, such as to generate the Verilog files for the Qsys system into which the OpenCL kernel is embedded.
`synthesize_cmd`	Command required to generate the **fpga.bin** file from the Custom Platform. Usually, this command instructs the Quartus Prime software to perform a full compilation.
`auto_migrate`	• `platform_type`—Choose this value based on the value referenced in the Altera Reference Platform from which you derive your Custom Platform. Valid values are `none`, `s5_net`, `c5soc`, and `a10_ref`. • `include fixes`—Comma-separated list of named fixes that you want to apply to the Custom Platform. • `exclude fixes`—Comma-separated list of named fixes that you do not want to apply to the Custom Platform.

MMD API Descriptions

The MMD interface is a cumulation of all the MMD application programming interface (API) functions.

Important: Full details about these functions, their arguments, and their return values are available in the **aocl_mmd.h** file. The **aocl_mmd.h** file is part of the Altera SDK for OpenCL Custom Platform Toolkit. Include the file in the operating system-specific implementations of the MMD layer.

aocl_mmd_get_offline_info on page 2-15
The `aocl_mmd_get_offline_info` function obtains offline information about the board specified in the `requested_info_id` argument.

aocl_mmd_get_info on page 2-18
The `aocl_mmd_get_info` function obtains information about the board specified in the `requested_info_id` argument.

aocl_mmd_open on page 2-19
The `aocl_mmd_open` function opens and initializes the specified device.

aocl_mmd_close on page 2-19
The `aocl_mmd_close` function closes an opened device via its handle.

aocl_mmd_get_offline_info

The `aocl_mmd_get_offline_info` function obtains offline information about the board specified in the `requested_info_id` argument. This function is offline because it is device-independent and does not require a handle from the `aocl_mmd_open()` call.

Syntax

```
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 )
```

Function Arguments

1. `requested_info_id`—An enum value of type `aocl_mmd_offline_info_t` that indicates the offline device information returning to the caller.

Table 2-13: Possible Enum Values for the requested_info_id Argument

Name	Description	Type
`AOCL_MMD_VERSION`	Version of MMD layer	`char*`
`AOCL_MMD_NUM_BOARDS`	Number of candidate boards	`int`
`AOCL_MMD_BOARD_NAMES`	Names of available boards **Attention:** Separate each board name by a semicolon (;) delimiter.	`char*`

Name	Description	Type
AOCL_MMD_VENDOR_NAME	Name of board vendor	char*
AOCL_MMD_VENDOR_ID	An integer board vendor ID	int
AOCL_MMD_USES_YIELD	A value of 0 instructs the Altera SDK for OpenCL host runtime to suspend user's processes. The host runtime resumes these processes after it receives an event update (for example, an interrupt) from the MMD layer . A value of 1 instructs the AOCL host runtime to continuously call the aocl_mmd_yield function while it waits for events to complete. **Caution:** Setting AOCL_MMD_USES_YIELD to 1 might cause high CPU utilization if the aocl_mmd_yield function does not suspend the current thread.	int

Name	Description	Type
AOCL_MMD_MEM_TYPES_SUPPORTED	A bit field listing all memory types that the Custom Platform supports. You may combine the following enum values in this bit field: AOCL_MMD_PHYSICAL_MEMORY—Custom Platform includes IP to communicate directly with physical memory (for example, DDR and QDR). AOCL_MMD_SVM_COARSE_GRAIN_BUFFER—Custom Platform supports both the caching of shared virtual memory (SVM) pointer data for OpenCL cl_mem objects and the requirement of explicit user function calls to synchronize the cache between the host processor and the FPGA. **Note:** Currently, Altera does not support this level of SVM except for a subset of AOCL_MMD_SVM_FINE_GRAIN_SYSTEM support. AOCL_MMD_SVM_FINE_GRAIN_BUFFER—Custom Platform supports caching of SVM pointer data for individual bytes. To synchronize the cache between the host processor and the FPGA, the Custom Platform requires information from the host runtime collected during pointer allocation. After an SVM pointer receives this additional data, the board interface synchronizes the cache between the host processor and the FPGA automatically. **Note:** Currently, Altera does not support this level of SVM except for a subset of AOCL_MMD_SVM_FINE_GRAIN_SYSTEM support. AOCL_MMD_SVM_FINE_GRAIN_SYSTEM—Custom Platform supports caching of SVM pointer data for individual bytes and it does not require additional information from the host runtime to synchronize the cache between the host processor and the FPGA. The board interface synchronizes the cache between the host processor and the FPGA automatically for all SVM pointers. **Attention:** Altera's support for this level of SVM is preliminary. Some features might not be fully supported.	int (bit field)

2. param_value_size—Size of the param_value field in bytes. This size_t value should match the size of the expected return type that the enum definition indicates.

   For example, if AOCL_MMD_NUM_BOARDS returns a value of type int, set the param_value_size to sizeof (int). You should see the same number of bytes returned in the param_size_ret argument.
3. param_value—A void* pointer to the variable that receives the returned information.

4. `param_size_ret`—A pointer argument of type `size_t*` that receives the number of bytes of returned data.

Return Value

A negative return value indicates an error.

aocl_mmd_get_info

The `aocl_mmd_get_info` function obtains information about the board specified in the `requested_info_id` argument.

Syntax

```
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 );
```

Function Arguments

1. `handle`—A positive `int` value representing the handle to the board obtained from the `aocl_mmd_open()` call.
2. `requested_info_id`—An enum value of type `aocl_mmd_offline_info_t` that indicates the device information returning to the caller.

Table 2-14: Possible Enum Values for the requested_info_id Argument

Name	Description	Type
`AOCL_MMD_NUM_KERNEL_INTERFACES`	Number of kernel interfaces	`int`
`AOCL_MMD_KERNEL_INTERFACES`	Kernel interfaces	`int*`
`AOCL_MMD_PLL_INTERFACES`	Kernel clock handles	`int*`
`AOCL_MMD_MEMORY_INTERFACE`	Global memory handle	`int`
`AOCL_MMD_TERMPERATURE`	Temperature measurement	`float`
`AOCL_MMD_PCIE_INFO`	PCIe information	`char*`
`AOCL_MMD_BOARD_NAME`	Board name	`char*`
`AOCL_MMD_BOARD_UNIQUE_ID`	Unique board ID	`char*`

3. `param_value_size`—Size of the `param_value` field in bytes. This `size_t` value should match the size of the expected return type that the enum definition indicates.

   For example, if `AOCL_MMD_TEMPERATURE` returns a value of type `float`, set the `param_value_size` to `sizeof (float)`. You should see the same number of bytes returned in the `param_size_ret` argument.
4. `param_value`—A `void*` pointer to the variable that receives the returned information.
5. `param_size_ret`—A pointer argument of type `size_t*` that receives the number of bytes of returned data.

Return Value

A negative return value indicates an error.

aocl_mmd_open

The `aocl_mmd_open` function opens and initializes the specified device.

Syntax

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

Function Arguments

`name`—The function opens the board with a name that matches this `const char*` string. The name typically matches the one specified by the `AOCL_MMD_BOARD_NAMES` offline information.

The OpenCL runtime first queries the `AOCL_MMD_BOARD_NAMES` offline information to identify the boards that it might be able to open. Then it attempts to open all possible devices by calling `aocl_mmd_open` and using each of the board names as argument.

Important: The name must be a C-style NULL-terminated ASCII string.

Return Value

If `aocl_mmd_open()` executes successfully, the return value is a positive integer that acts as a handle to the board.

If `aocl_mmd_open()` fails to execute, a negative return value indicates an error. In the event of an error, the OpenCL runtime proceeds to open other known devices. Therefore, it is imperative that the MMD layer does not exit the application if an open call fails.

aocl_mmd_close

The `aocl_mmd_close` function closes an opened device via its handle.

Syntax

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

Function Arguments

`handle`—A positive `int` value representing the handle to the board obtained from the `aocl_mmd_open()` call.

Return Value

If the `aocl_mmd_close()` executes successfully, the return value is 0.

If `aocl_mmd_close()` fails to execute, a negative return value indicates an error.

aocl_mmd_read

The `aocl_mmd_read` function is the read operation on a single interface.

Syntax

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

Function Arguments

1. `handle`—A positive `int` value representing the handle to the board obtained from the `aocl_mmd_open()` call.
2. `op`—The operation object of type `aocl_mmd_op_t` used to track the progress of the operation. If `op` is NULL, the call must block, and return only after the operation completes.

   Note: `aocl_mmd_op_t` is defined as follows:

   ```
   typedef void* aocl_mmd_op_t;
   ```

3. `len`—The size of the data, in bytes, that the function transfers. Declare `len` with type `size_t`.
4. `dst`—The host buffer, of type `void*`, to which data is written.
5. `interface`—The handle to the interface that `aocl_mmd_read` is accessing. For example, to access global memory, this handle is the enum value `aocl_mmd_get_info()` returns when its `requested_info_id` argument is `AOCL_MMD_MEMORY_INTERFACE`. The interface argument is of type `aocl_mmd_interface_t`, and can take one of the following values:

Name	Description
`AOCL_MMD_KERNEL`	Control interface into the kernel interface
`AOCL_MMD_MEMORY`	Data interface to device memory
`AOCL_MMD_PLL`	Interface for reconfigurable PLL

6. `offset`—The `size_t` byte offset within the interface at which the data transfer begins.

Return Value

If the read operation is successful, the return value is 0.

If the read operation fails, a negative return value indicates an error.

aocl_mmd_write

The `aocl_mmd_write` function is the write operation on a single interface.

Syntax

```
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 );
```

Function Arguments

1. `handle`—A positive `int` value representing the handle to the board obtained from the `aocl_mmd_open()` call.
2. `op`—The operation object of type `aocl_mmd_op_t` used to track the progress of the operation. If `op` is NULL, the call must block, and return only after the operation completes.

   Note: `aocl_mmd_op_t` is defined as follows:

   ```
   typedef void* aocl_mmd_op_t;
   ```

3. `len`—The size of the data, in bytes, that the function transfers. Declare `len` with type `size_t`.
4. `src`—The host buffer, of type `const void*`, from which data is read.
5. `interface`—The handle to the interface that `aocl_mmd_write` is accessing. For example, to access global memory, this handle is the enum value `aocl_mmd_get_info()` returns when its `requested_info_id` argument is `AOCL_MMD_MEMORY_INTERFACE`. The interface argument is of type `aocl_mmd_interface_t`, and can take one of the following values:

Name	Description
`AOCL_MMD_KERNEL`	Control interface into the kernel interface
`AOCL_MMD_MEMORY`	Data interface to device memory
`AOCL_MMD_PLL`	Interface for reconfigurable PLL

6. `offset`—The `size_t` byte offset within the interface at which the data transfer begins.

Return Value

If the read operation is successful, the return value is 0.

If the read operation fails, a negative return value indicates an error.

aocl_mmd_copy

The `aocl_mmd_copy` function is the copy operation on a single interface.

Syntax

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

Function Arguments

1. `handle`—A positive `int` value representing the handle to the board obtained from the `aocl_mmd_open()` call.
2. `op`—The operation object of type `aocl_mmd_op_t` used to track the progress of the operation. If `op` is NULL, the call must block, and return only after the operation completes.

Note: `aocl_mmd_op_t` is defined as follows:

```
typedef void* aocl_mmd_op_t;
```

3. `len`—The size of the data, in bytes, that the function transfers. Declare `len` with type `size_t`.
4. `intf`—The handle to the interface that `aocl_mmd_read` is accessing. For example, to access global memory, this handle is the enum value `aocl_mmd_get_info()` returns when its `requested_info_id` argument is `AOCL_MMD_MEMORY_INTERFACE`. The interface argument is of type `aocl_mmd_interface_t`, and can take one of the following values:

Name	Description
`AOCL_MMD_KERNEL`	Control interface into the kernel interface
`AOCL_MMD_MEMORY`	Data interface to device memory
`AOCL_MMD_PLL`	Interface for reconfigurable PLL

5. `src_offset`—The `size_t` byte offset within the source interface at which the data transfer begins.
6. `dst_offset`—The `size_t` byte offset within the destination interface at which the data transfer begins.

Return Value

If the copy operation is successful, the return value is 0.

If the copy operation fails, a negative return value indicates an error.

aocl_mmd_set_interrupt_handler

The `aocl_mmd_set_interrupt_handler` function sets the interrupt handler for the opened device. When the device internals identify an asynchronous kernel event (for example, a kernel completion), the interrupt handler is called to notify the OpenCL runtime of the event.

Attention: Ignore the interrupts from the kernel until this handler is set.

Syntax

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

Function Arguments

1. `handle`—A positive `int` value representing the handle to the board obtained from the `aocl_mmd_open()` call.
2. `fn`—The callback function to invoke when a kernel interrupt occurs. The `fn` argument is of type `aocl_mmd_interrupt_handler_fn`, which is defined as follows:

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

3. `user_data`—The `void*` type user-provided data that passes to `fn` when it is called.

Return Value

If the function executes successfully, the return value is 0.

If the function fails to execute, a negative return value indicates an error.

aocl_mmd_set_status_handler

The `aocl_mmd_set_status_handler` function sets the operation status handler for the opened device. The operation status handler is called under the following circumstances:

- When the operation completes successfully and status is 0.
- When the operation completes with errors and status is a negative value.

Syntax

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

Function Arguments

1. `handle`—A positive `int` value representing the handle to the board obtained from the `aocl_mmd_open()` call.
2. `fn`—The callback function to invoke when a status update occurs. The `fn` argument is of type `aocl_mmd_status_handler_fn`, which is defined as follows:

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

3. `user_data`—The `void*` type user-provided data that passes to `fn` when it is called.

Return Value

If the function executes successfully, the return value is 0.

If the function fails to execute, a negative return value indicates an error.

aocl_mmd_yield

The `aocl_mmd_yield` function is called when the host interface is idle. The host interface might be idle because it is waiting for the device to process certain events.

Syntax

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

Function Arguments

1. `handle`—A positive `int` value representing the handle to the board obtained from the `aocl_mmd_open()` call.

Return Value

A nonzero return value indicates that the yield function performed work necessary for proper device functioning such as processing direct memory access (DMA) transactions.

A return value of 0 indicates that the yield function did not perform work necessary for proper device functioning.

Note: The yield function might be called continuously as long as it reports that it has necessary work to perform.

aocl_mmd_shared_mem_alloc

The `aocl_mmd_shared_mem_alloc` function allocates shared memory between the host and the FPGA. The host accesses the shared memory via the pointer returned by `aocl_mmd_shared_mem_alloc`. The FPGA accesses the shared memory via `device_ptr_out`. If shared memory is not available, `aocl_mmd_shared_mem_alloc` returns `NULL`. If you do not reboot the CPU after you reprogram the FPGA, the shared memory will persist.

Syntax

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

Function Arguments

1. `handle`—A positive `int` value representing the handle to the board obtained from the `aocl_mmd_open()` call.
2. `size`—The size of the shared memory that the function allocates. Declare `size` with the type `size_t`.
3. `device_ptr_out`—The argument that receives the pointer value the device uses to access shared memory. The `device_ptr_out` is of type `unsigned long long` to handle cases where the host has a smaller pointer size than the device. The `device_ptr_out` argument cannot have a NULL value.

Return Value

If `aocl_mmd_shared_mem_alloc` executes successfully, the return value is the pointer value that the host uses to access the shared memory. Otherwise, the return value is NULL.

aocl_mmd_shared_mem_free

The `aocl_mmd_shared_mem_free` function frees allocated shared memory. This function does nothing if shared memory is not available.

Syntax

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

Function Arguments

1. `handle`—A positive `int` value representing the handle to the board obtained from the `aocl_mmd_open()` call.
2. `host_ptr`—The host pointer that points to the shared memory, as returned by the `aocl_mmd_shared_mem_alloc()` function.
3. `size`—The size of the allocated shared memory that the function frees. Declare `size` with the type `size_t`.

Return Value

The `aocl_mmd_shared_mem_free` function has no return value.

aocl_mmd_reprogram

The `aocl_mmd_reprogram` function is the reprogram operation for the specified device.
The host must guarantee that no other OpenCL operations are executing on the device during the reprogram operation. During `aocl_mmd_reprogram` execution, the kernels are idle and no read, write, or copy operation can occur.

Disable interrupts and reprogram the FPGA with the data from `user_data`, which has a size specified by the `size` argument. The host then calls `aocl_mmd_set_status_handler` and `aocl_mmd_set_interrupt_handler` again, which enable the interrupts. If events such as interrupts occur during `aocl_mmd_reprogram` execution, race conditions or data corruption might occur.

Syntax

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

Function Arguments

1. `handle`—A positive `int` value representing the handle to the board obtained from the `aocl_mmd_open()` call.
2. `user_data`—The `void*` type binary contents of the **fpga.bin** file that are created during compilation.
3. `size`—The size of `user_data` in bytes. The size argument is of `size_t`.

Return Value

If `aocl_mmd_reprogram` executes successfully, the return value is the pointer value that the host uses to access shared memory.

Reprogram Support

For Altera SDK for OpenCL users who program their FPGAs with the `clCreateProgramWithBinary` flow (that is, reprogram-on-the-fly), the `aocl_mmd_reprogram` subroutine is used to configure the FPGA from within the host applications. The host ensures that this call executes only when the FPGA is idle, meaning that no kernels are running and no transfers are outstanding. The MMD layer must then reconfigure the device with the data in the `user_data` argument of `aocl_mmd_reprogram`.

The data in the `user_data` argument is the same **fpga.bin** data created during Quartus Prime compilation. The Altera Offline Compiler packages the exact contents of **fpga.bin** into the **.aocx** file during compilation. The contents of the **fpga.bin** is irrelevant to the AOC. It simply passes the file contents through the host and to the `aocl_mmd_reprogram` call via the `user_data` argument.

For more information on the `clCreateProgramWithBinary` function, refer to the *OpenCL Specification version 1.0* and the *Programming an FPGA via the Host* section of the *Altera SDK for OpenCL Programming Guide*.

Related Information

- **OpenCL Specification version 1.0**
- **Programming an FPGA via the Host**

Document Revision History

Table 2-15: Document Revision History of the AOCL Custom Platform Toolkit Reference Material Chapter of the Altera SDK for OpenCL Custom Platform Toolkit User Guide

Date	Version	Changes
November 2015	2015.11.02	• Maintenance release, and changed instances of *Quartus II* to *Quartus Prime*.
May 2015	15.0.0	• Maintenance release.
December 2014	14.1.0	• Under *XML Elements, Attributes, and Parameters in the board_spec.xml File*, added information on the `compile` eXtensible Markup Language element and its associated attributes and parameters. • Under *MMD API Descriptions*, added information on the `AOCL_MMD_USES_YIELD` and the `AOCL_MMD_MEM_TYPES_SUPPORTED` enum values for the `requested_info_id` variable in the `aocl_mmd_get_offline_info` function.
October 2014	14.0.1	• Reorganized existing document into two chapters.
June 2014	14.0.0	• Initial release.