3.5 Dynamic PLL Reconfiguration

3.6 Guaranteed Timing Closure of the Arria 10 GX FPGA Development Kit Reference Platform Design
   3.6.1 Supply the Kernel Clock
   3.6.2 Guarantee Kernel Clock Timing
   3.6.3 Provide a Timing-Closed Post-Fit Netlist

3.7 Quartus Prime Compilation Flow and Scripts
   3.7.1 Enabling the Quartus Prime Spectra-Q Forward-Compatibility Flow
   3.7.2 Quartus Prime Compilation Flow for Board Developers
   3.7.3 Quartus Prime Compilation Flow for Custom Platform Users
   3.7.4 Qsys Pro System Generation
   3.7.5 QDB File Generation
   3.7.6 Hash Checking

3.8 Addition of Timing Constraints

3.9 Connection of the Arria 10 GX FPGA Development Kit Reference Platform to the Intel FPGA SDK for OpenCL
   3.9.1 Describe the Arria 10 GX FPGA Development Kit Reference Platform to the Intel FPGA SDK for OpenCL
   3.9.2 Describe the Arria 10 GX FPGA Development Kit Reference Platform Hardware to the Intel FPGA SDK for OpenCL

3.10 Arria 10 FPGA Programming Flow
   3.10.1 Define the Contents of the fpga.bin File for the Arria 10 GX FPGA Development Kit Reference Platform

3.11 Host-to-Device MMD Software Implementation

3.12 Implementation of Intel FPGA SDK for OpenCL Utilities
   3.12.1 aocl install
   3.12.2 aocl uninstall
   3.12.3 aocl program
   3.12.4 aocl flash
   3.12.5 aocl diagnose

3.13 Arria 10 FPGA Development Kit Reference Platform Scripts

3.14 Considerations in Arria 10 GX FPGA Development Kit Reference Platform Implementation

4 Document Revision History
1 Intel FPGA SDK for OpenCL Arria 10 GX FPGA Development Kit Reference Platform Porting Guide

The Arria 10 GX FPGA Development Kit Reference Platform Porting Guide describes the procedures and design considerations for modifying the Arria® 10 GX FPGA Development Kit Reference Platform (a10_ref) into your own Custom Platform for use with the Intel® FPGA Software Development Kit (SDK) for OpenCL™ 1 2.

1.1 Arria 10 GX FPGA Development Kit Reference Platform: Prerequisites

The Arria 10 GX FPGA Development Kit Reference Platform Porting Guide assumes that you are an experienced FPGA designer who is familiar with Intel's FPGA design tools and concepts.

Prerequisites for the altera_a10pciedk Reference Platform:

- An Arria 10-based accelerator card with working PCI Express® (PCIe®) and memory interfaces
  Test these interfaces together in the same design using the same version of the Quartus® Prime Pro Edition software that you will use to develop your Custom Platform.

  **Attention:** The native Arria 10 GX FPGA Development Kit does not automatically work with the SDK. Before using the Arria 10 GX FPGA Development Kit with the SDK, you must first contact your field applications engineer or regional support center representative who will configure the development kit for you.
  Alternatively, contact support for assistance.

- Quartus Prime Pro Edition software
- Designing with LogicLock® Plus regions

General prerequisites:

- FPGA architecture, including clocking, global routing, and I/Os
- High-speed design
- Timing analysis
- Qsys Pro design and Avalon® interfaces

---

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

2 The Intel FPGA SDK for OpenCL is based on a published Khronos Specification, and has passed the Khronos Conformance Testing Process. Current conformance status can be found at www.khronos.org/conformance.
• Tcl scripting
• PCIe
• DDR4 external memory

This document also assumes that you are familiar with the following Intel FPGA SDK for OpenCL-specific tools and documentation:

• Custom Platform Toolkit and the Intel FPGA SDK for OpenCL Custom Platform Toolkit User Guide
• Stratix® V Network Reference Platform (s5_net) and the Stratix V Network Reference Platform Porting Guide

The memory-mapped device (MMD) and driver software stack in the a10_ref Reference Platform is derived from the s5_net Reference Platform design.

Related Links
• Intel FPGA SDK for OpenCL Custom Platform Toolkit User Guide
• Intel FPGA SDK for OpenCL Stratix V Network Reference Platform Porting Guide

1.2 Features of the Arria 10 GX FPGA Development Kit Reference Platform

Prior to designing an Intel FPGA SDK for OpenCL Custom Platform, decide on design considerations that allow you to fully utilize the available hardware on your computing card.

The Arria 10 GX FPGA Development Kit Reference Platform targets a subset of the hardware features available in the Arria 10 GX FPGA Development Kit.

Figure 1. Hardware Features of the Arria 10 GX FPGA Development Kit

Features of the a10_ref Reference Platform:
• **OpenCL Host**
The a10_ref Reference Platform uses a PCIe-based host that connects to the Arria 10 PCIe Gen3 x8 hard IP core.

• **OpenCL Global Memory**
The hardware provides one 2-gigabyte (GB) DDR4 SDRAM daughtercard that is mounted on the HiLo connector (J14 in Figure 1 on page 5).

• **FPGA Programming via one of the following methods:**
  — Partial Reconfiguration (PR) over PCIe.
  — External cable and the Arria 10 GX FPGA Development Kit’s on-board USB Download Cable II interface.
  — External USB Blaster II interface connected to a 10-pin JTAG header.

• **Guaranteed Timing**
The a10_ref Reference Platform relies on the Quartus Prime Pro Edition software's Spectra-Q™ engine compilation flow to provide guaranteed timing closure. The timing-clean a10_ref Reference Platform is preserved in the form of a precompiled post-fit netlist (that is, the base.qdb Quartus Prime Database Export File). The Intel FPGA SDK for OpenCL Offline Compiler imports this preserved post-fit netlist into each OpenCL kernel compilation.

### 1.2.1 Arria 10 GX FPGA Development Kit Reference Platform Board Variants

The Arria 10 GX FPGA Development Kit Reference Platform has one board variant (that is, a10gx) that targets the Arria 10 GX FPGA Development Kit containing the production silicon for Arria 10 FPGA (-1 speed grade) and DDR4-2400 SDRAM.

To compile your OpenCL kernel for a specific board variant, include the `--board <board_name>` option in your `aoc` command (for example, `aoc --board a10gx myKernel.cl`).

**Related Links**

- Compiling a Kernel for a Specific FPGA Board (`--board <board_name>`)

### 1.3 Contents of the Arria 10 GX FPGA Development Kit Reference Platform

Familiarize yourself with the directories and files within the Arria 10 GX FPGA Development Kit Reference Platform because they are referenced throughout this document.

<table>
<thead>
<tr>
<th>File or Folder</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>board_env.xml</td>
<td>eXtensible Markup Language (XML) file that describes the Reference Platform to the Intel FPGA SDK for OpenCL.</td>
</tr>
<tr>
<td>hardware</td>
<td>Contains the Quartus Prime project templates for the a10gx board variant.</td>
</tr>
</tbody>
</table>

Table 1. **Highlights of the Arria 10 GX FPGA Development Kit Reference Platform Directory**

---

Intel FPGA SDK for OpenCL Arria 10 GX FPGA Development Kit Reference Platform Porting Guide

6
<table>
<thead>
<tr>
<th>Windows File or Folder</th>
<th>Linux File or Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>windows64</td>
<td>linux64</td>
<td>Contains the MMD library, kernel mode driver, and executable files of the SDK utilities (that is, install, uninstall, flash, program, diagnose) for your 64-bit operating system.</td>
</tr>
<tr>
<td>source</td>
<td>source</td>
<td>For Windows, the source folder contains source codes for the MMD library and SDK utilities. The MMD library and the SDK utilities are in the windows64 folder. For Linux, the source directory contains source codes for the MMD library and SDK utilities. The MMD library and the SDK utilities are in the linux64 directory.</td>
</tr>
</tbody>
</table>

### Table 2. Contents of the a10gx Directory

The following table lists the files in the ALTERAOCLSDKROOT/board/a10_ref/hardware/a10gx directory, where ALTERAOCLSDKROOT points to the location of the SDK installation.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl_ddr4_a10.qsys</td>
<td>Qsys Pro system that, together with the .ip files in the ip/acl_ddr4_a10 subdirectory, implements the acl_ddr4_a10 component.</td>
</tr>
<tr>
<td>acl_ddr4_a10_core.qsys</td>
<td>Qsys Pro system that, together with the .ip files in the ip/acl_ddr4_a10_core subdirectory, implements the acl_ddr4_a10_core component.</td>
</tr>
<tr>
<td>base.qsf</td>
<td>Quartus Prime Settings File for the base project revision. This file includes, by reference, all the settings in the flat.qsf file. Use this revision when porting the a10_ref Reference Platform to your own Custom Platform. The Quartus Prime Pro Edition software compiles this base project revision from source code.</td>
</tr>
<tr>
<td>base.qdb</td>
<td>Quartus Prime Database Export File that contains the precompiled netlist of the static region of the design. This file is generated by the scripts/post_flow_pr.tcl file during base revision compilations and is used during import revision compilations.</td>
</tr>
<tr>
<td>base.sdc</td>
<td>Synopsys Design Constraints File that the Quartus Prime software autogenerates during a base compilation. The base.sdc file is used in the top revision compilation to import all the timing constraints from the static region. This file must be in-sync with the base.qdb file. Everytime you perform a base compilation to create a new.qdb file, copy the base.sdc file, the base.qdb file, and the pr_base_id.txt file into your Custom Platform.</td>
</tr>
<tr>
<td>board.qsys</td>
<td>Qsys Pro system that implements the board interfaces (that is, the static region) of the OpenCL hardware system.</td>
</tr>
<tr>
<td>board_spec.xml</td>
<td>XML file that provides the definition of the board hardware interfaces to the SDK.</td>
</tr>
<tr>
<td>device.tcl</td>
<td>Tcl file that is included in all revisions and contains all device-specific information (for example, device family, ordering part number (OPN), voltage settings, etc.)</td>
</tr>
<tr>
<td>flat.qsf</td>
<td>Quartus Prime Settings File for the flat project revision. This file includes all the common settings, such as pin location assignments, that are used in the other revisions of the project (that is, base, top, and top_synth). The base.qsf, top.qsf, and top_synth.qsf files include, by reference, all the settings in the flat.qsf file. The Quartus Prime software compiles the flat revision with minimal location constraints. The flat revision compilation does not generate a base.qdb file that you can use for future import compilations and does not implement the guaranteed timing flow.</td>
</tr>
<tr>
<td>File</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>import_compile.tcl</td>
<td>Tcl script for the SDK-user compilation flow (that is, import revision compilation).</td>
</tr>
<tr>
<td>max5_150.pof</td>
<td>Programming file for the MAX® V device on the Arria 10 GX FPGA Development Kit that sets the memory reference clock to 150 MHz by default at power-up. You must program the max5_150.pof file onto your a10gx board.</td>
</tr>
<tr>
<td>opencl_bsp_ip.qsf</td>
<td>Quartus Prime Settings File that collects all the required .ip files in a unique location. During flat and base revision compilations, the board.qsys Pro, acl_ddr4_a10.qsys Pro and acl_ddr4_a10_core.qsys Pro files are added to the opencl_bsp_ip.qsf file.</td>
</tr>
<tr>
<td>pr_base_id.txt</td>
<td>Text file containing a unique number for a given base compilation that the runtime uses to determine whether it is safe to use PR programming. The pr_base_id.txt file is generated each time you perform a base compilation. The unique number in this file is included in the Intel FPGA SDK for OpenCL Offline Compiler executable file (.aocx) that each import compilation generates.</td>
</tr>
<tr>
<td>quartus.ini</td>
<td>Contains any special Quartus Prime software options that you need to compile OpenCL kernels for the a10_ref Reference Platform.</td>
</tr>
<tr>
<td>top.qpf</td>
<td>Quartus Prime Project File for the OpenCL hardware system.</td>
</tr>
<tr>
<td>top.qsf</td>
<td>Quartus Prime Settings File for the SDK-user compilation flow.</td>
</tr>
<tr>
<td>top.sdc</td>
<td>Synopsys Design Constraints File that contains board-specific timing constraints.</td>
</tr>
<tr>
<td>top.v</td>
<td>Top-level Verilog Design File for the OpenCL hardware system.</td>
</tr>
<tr>
<td>top_post.sdc</td>
<td>Qsys Pro and Intel FPGA SDK for OpenCL IP-specific timing constraints.</td>
</tr>
<tr>
<td>top_synth.qsf</td>
<td>Quartus Prime Settings File for the Quartus Prime revision in which the OpenCL kernel system is synthesized.</td>
</tr>
<tr>
<td>ip/acl_ddr4_a10/&lt;file_name&gt;</td>
<td>Directory containing the .ip files that the Quartus Prime Pro Edition software needs to parameterize the acl_ddr4_a10 component. You must provide both the acl_ddr4_a10.qsys file and the corresponding .ip files in this directory to the Quartus Prime Pro Edition software.</td>
</tr>
<tr>
<td>ip/acl_ddr4_a10_core/&lt;file_name&gt;</td>
<td>Directory containing the .ip files that the Quartus Prime Pro Edition software needs to parameterize the acl_ddr4_a10_core component. You must provide both the acl_ddr4_a10_core.qsys file and the corresponding .ip files in this directory to the Quartus Prime Pro Edition software.</td>
</tr>
<tr>
<td>ip/board/&lt;file_name&gt;</td>
<td>Directory containing the .ip files that the Quartus Prime Pro Edition software needs to parameterize the board instance. You must provide both the board.qsys file and the corresponding .ip files in this directory to the Quartus Prime Pro Edition software.</td>
</tr>
<tr>
<td>ip/freeze_wrapper.v</td>
<td>Verilog Design File that implements the freeze logic placed at outputs of the Partial Reconfiguration region.</td>
</tr>
<tr>
<td>ip/irq_controller/&lt;file_name&gt;</td>
<td>IP that receives interrupts from the OpenCL kernel system and sends message signaled interrupts (MSI) to the host. Refer to the Message Signaled Interrupts section for more information.</td>
</tr>
</tbody>
</table>

continued...
<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scripts/base_write_sdc.tcl</td>
<td>Tcl script that the base revision compilation uses to generate the base.sdc file containing all the constraints collected in the base revision compilation. The Quartus Prime Pro Edition software uses the base.sdc file when compiling the import (top) revision.</td>
</tr>
<tr>
<td>scripts/create_fpga_bin_pr.tcl</td>
<td>Tcl script that generates the fpga.bin file. The fpga.bin file contains all the necessary files for configuring the FPGA. For more information on the fpga.bin file, refer to the Define the Contents of the fpga.bin File for the Arria 10 GX FPGA Development Kit Reference Platform section.</td>
</tr>
<tr>
<td>scripts/post_flow_pr.tcl</td>
<td>Tcl script that implements the guaranteed timing closure flow, as described in the Guaranteed Timing Closure of the Arria 10 GX FPGA Development Kit Reference Platform Design section.</td>
</tr>
<tr>
<td>scripts/pre_flow_pr.tcl</td>
<td>Tcl script that executes before the invocation of the Quartus Prime software compilation. Running the script generates the Qsys Pro HDL for board.qsys and kernel_system.qsys. It also creates a unique ID for the PR base revision (that is, static region). This unique ID is stored in the pr_base_id.txt file.</td>
</tr>
</tbody>
</table>

Related Links

- **Guaranteed Timing Closure of the Arria 10 GX FPGA Development Kit Reference Platform Design** on page 40
  Both the SDK and the Custom Platform contribute to the implementation of the SDK's guaranteed timing closure feature.

- **Message Signaled Interrupt** on page 30
  The Arria 10 GX FPGA Development Kit Reference Platform uses one MSI line for both DMA and the kernel interface.

- **Define the Contents of the fpga.bin File for the Arria 10 GX FPGA Development Kit Reference Platform** on page 52
  Intel defines the contents of the fpga.bin file in the Arria 10 GX FPGA Development Kit Reference Platform as an Executable and Linkable Format (ELF) binary the organizes the various fields into sections.

- **Hash Checking** on page 48
  Intel assigns a unique ID to each base revision compilation to ensure a safe way of only partially reconfiguring a PR region on top of a design that has a matching static region.
2 Developing Your Arria 10 Custom Platform

Use the tools available in Arria 10 GX FPGA Development Kit Reference Platform (a10_ref) and the Intel FPGA SDK for OpenCL Custom Platform Toolkit together to create your own Custom Platform.

Developing your Custom Platform requires in-depth knowledge of the contents in the following documents and tools:

- Intel FPGA SDK for OpenCL Custom Platform User Guide
- Contents of the SDK Custom Platform Toolkit
- Stratix V Network Reference Platform Porting Guide
- Documentation for all the Intel FPGA IP in your Custom Platform
- Intel FPGA SDK for OpenCL Getting Started Guide
- Intel FPGA SDK for OpenCL Programming Guide

In addition, you must independently verify all IP on your computing card (for example, PCIe controllers and DDR4 external memory).

Related Links
- Intel FPGA SDK for OpenCL Custom Platform Toolkit User Guide
- Intel FPGA SDK for OpenCL Stratix V Network Reference Platform Porting Guide
- Intel FPGA SDK for OpenCL Getting Started Guide
- Intel FPGA SDK for OpenCL Programming Guide

2.1 Initializing Your Arria 10 Custom Platform

To initialize your Intel FPGA SDK for OpenCL Custom Platform, copy the Arria 10 GX FPGA Development Kit Reference Platform to another directory and rename it.

1. Copy the ALTERAOCLSDKROOT/board/a10_ref directory, where ALTERAOCLSDKROOT is points to the location of the SDK installation.
2. Paste the a10_ref directory into a directory that you own (that is, not a system directory) and then rename it (<your_custom_platform>).
3. Choose the a10gx board variant in the <your_custom_platform>/hardware directory to match the production silicon for the Arria 10 FPGA as the basis of your design.
4. Rename a10gx board variant to match the name of your FPGA board (<your_custom_platform>/hardware/<board_name>).
5. Modify the `<your_custom_platform>/board_env.xml` file so that the `name` and `default` fields match the changes you made in 2 on page 10 and 4 on page 10, respectively.

6. Set the environment variable `AOCL_BOARD_PACKAGE_ROOT` variable to point to the location of your Custom Platform.

7. In the SDK, invoke the command `aoc --list-boards` to confirm that the Intel FPGA SDK for OpenCL Offline Compiler displays the board name in your Custom Platform.

Related Links
- Setting the Intel FPGA SDK for OpenCL User Environment Variables for Windows
- Setting the Intel FPGA SDK for OpenCL User Environment Variables for Linux
- Describe the Arria 10 GX FPGA Development Kit Reference Platform to the Intel FPGA SDK for OpenCL on page 49

The `ALTERAOCLSDKROOT/board/a10_ref/board_env.xml` file describes the Arria 10 GX FPGA Development Kit Reference Platform to the Intel FPGA SDK for OpenCL.

2.2 Modifying the Arria 10 GX FPGA Development Kit Reference Platform Design

Modify the Quartus Prime design for the Arria 10 GX FPGA Development Kit Reference Platform to fit your design needs.

1. Instantiate your PCIe controller, as described in *Host-to-Arria 10 Communication over PCIe* section.

2. Instantiate any memory controllers and I/O channels. You can add the board interface hardware either as Qsys Pro components in the `board.qsys` Qsys Pro system or as HDL in the `top.v` file.

   The `board.qsys` file and the `top.v` file are in the `<your_custom_platform>/hardware/<board_name>` directory.

3. Modify the `device.tcl` file to match all the correct settings for the device on your board.

4. Modify the `<your_custom_platform>/hardware/<board_name>/flat.qsf` file to use only the pin-outs and settings for your system. The `base.qsf`, `top.qsf`, and `top_synth.qsf` files will include all the settings from the `flat.qsf` file.

   The `top.qsf` file and `top_synth.qsf` file are in the `<your_custom_platform>/hardware/<board_name>` directory.

Related Links
- *Host-to-Arria 10 FPGA Communication over PCIe* on page 22
  The Arria 10 GX FPGA Development Kit Reference Platform instantiates the Arria 10 PCIe hard IP to implement a host-to-device connection over PCIe.
2.3 Integrating Your Arria 10 Custom Platform with the Intel FPGA SDK for OpenCL

After you modify your Quartus Prime design files, integrate your Custom Platform with the Intel FPGA SDK for OpenCL.

1. Update the `<your_custom_platform>/hardware/<board_name>/board_spec.xml` file. Ensure that there is at least one global memory interface, and all the global memory interfaces correspond to the exported interfaces from the `board.qsys` Qsys Pro System File.

2. Set the environment variable `ACL_QSH_COMPILE_CMD` to `quartus_sh --flow compile top -c base`.
   Setting this environment variable instructs the SDK to compile the base revision corresponding to the `<your_custom_platform>/hardware/<board_name>/base.qsf` file.

3. Perform the steps outlined in the `ALTERAOCLSDKROOT/board/custom_platform_toolkit/tests/README.txt` file to compile the `ALTERAOCLSDKROOT/board/custom_platform_toolkit/tests/boardtest/boardtest.cl` OpenCL kernel source file.
   The environment variable `ALTERAOCLSDKROOT` points to the location of the SDK installation.

4. If compilation fails because of timing failures, fix the errors, or compile `ALTERAOCLSDKROOT/board/custom_platform_toolkit/tests/boardtest.cl` with different seeds. To compile the kernel with a different seed, include the `--seed <N>` option in the `aoc` command (for example, `aoc --seed 2 boardtest.cl`).
   You might be able to fix minor timing issues by simply compiling your kernel with a different seed.

Related Links

- Describe the Arria 10 GX FPGA Development Kit Reference Platform Hardware to the Intel FPGA SDK for OpenCL on page 50
  The Arria 10 GX FPGA Development Kit Reference Platform includes an `ALTERAOCLSDKROOT/board/a10_ref/hardware/a10gx/board_spec.xml` file that describes the hardware to the Intel FPGA SDK for OpenCL.

2.4 Setting up the Arria 10 Custom Platform Software Development Environment

Prior to building the software layer for your Intel FPGA SDK for OpenCL Custom Platform, set up the software development environment.
To compile the MMD layer for Windows, perform the following tasks:

a. Install the GNU make utility on your development machine.

b. Install a version of Microsoft Visual Studio that has the ability to compile 64-bit software (for example, Microsoft Visual Studio version 2010 Professional).

c. Set the development environment so that SDK users can invoke commands and utilities at the command prompt.

d. Modify the `<your_custom_platform_name>/source/Makefile.common` file so that `TOP_DEST_DIR` points to the top-level directory of your Custom Platform.

e. In the `Makefile.common` file or the development environment, set the `JUNGO_LICENSE` variable to your Jungo WinDriver license.

f. To check that you have set up the software development environment properly, invoke the `gmake` or `gmake clean` command.

To compile the MMD layer for Linux, perform the following tasks:

a. Ensure that you use a Linux distribution that Intel supports (for example, GNU Compiler Collection (GCC) version 4.47).

b. Modify the `<your_custom_platform>/source/Makefile.common` file so that `TOP_DEST_DIR` points to the top-level directory of your Custom Platform.

to check that you have set up the software environment properly, invoke the `make` or `make clean` command.

**Related Links**

Jungo Connectivity Ltd. website

### 2.5 Establishing Arria 10 Custom Platform Host Communication

After modifying and rebranding the Arria 10 GX FPGA Development Kit Reference Platform to your own Custom Platform, use the tools and utilities in your Custom Platform to establish communication between your FPGA accelerator board and your host application.

1. Program your FPGA device with the `<your_custom_platform>/hardware/<board_name>/base.sof` file and then reboot your system.

You should have created the `base.sof` file when integrating your Custom Platform with the Intel FPGA SDK for OpenCL. Refer to the `Integrating Your Arria 10 Custom Platform with the Intel FPGA SDK for OpenCL` section for more information.

2. Confirm that your operating system recognizes a PCIe device with your vendor and device IDs.
   
   — For Windows, open the **Device Manager** and verify that the correct device and IDs appear in the listed information.
For Linux, invoke the `lspci` command and verify that the correct device and IDs appear in the listed information.

3. In the SDK, verify that the environment variable `AOCL_BOARD_PACKAGE_ROOT` points to the location of the current Custom Platform. Then run the `aocli install` utility command to install the kernel driver on your machine.

4. For Windows, set the `PATH` environment variable. For Linux, set the `LD_LIBRARY_PATH` environment variable.

   For more information on the settings for `PATH` and `LD_LIBRARY_PATH`, refer to Setting the Intel FPGA SDK for OpenCL User Environment Variables in the Intel FPGA SDK for OpenCL Getting Started Guide.

5. Modify the `version_id_test` function in your `<your_custom_platform>/source/host/mmd/acl_pcie_device.cpp` MMD source code file to exit after reading from the version ID register.

6. Run the `aocl diagnose` utility command and confirm that the version ID register reads back the ID successfully. You may set the environment variables `ACL_HAL_DEBUG` and `ACL_PCIE_DEBUG` to a value of 1 to visualize the result of the diagnostic test on your terminal.

**Related Links**

- Integrating Your Arria 10 Custom Platform with the Intel FPGA SDK for OpenCL on page 12
  
  After you modify your Quartus Prime design files, integrate your Custom Platform with the Intel FPGA SDK for OpenCL.

- Setting the Intel FPGA SDK for OpenCL Environment Variables for Linux

- Setting the Intel FPGA SDK for OpenCL User Environment Variables for Windows

### 2.6 Branding Your Arria 10 Custom Platform

Modify the library, driver and source files in the Arria 10 GX FPGA Development Kit Reference Platform to reference your Intel FPGA SDK for OpenCL Custom Platform.

1. In the software development environment available with the `a10_ref` Reference Platform, replace all references of "a10_ref" with the name of your Custom Platform.

2. Modify the `PACKAGE_NAME` and `MMD_LIB_NAME` fields in the `<your_custom_platform>/source/Makefile.common` file.

3. Modify the `name`, `linklib`, and `mmlibs` elements in `<your_custom_platform>/board_env.xml` file to your custom MMD library name.

4. In your Custom Platform, modify the following lines of code in the `hw_pcie_constants.h` file to include information of your Custom Platform:

```c
#define ACL_BOARD_PKG_NAME "a10_ref"
#define ACL_VENDOR_NAME "Intel Corporation"
#define ACL_BOARD_NAME "Arria 10 Reference Platform"
```
For Windows, the hw_pcie_constants.h file is in the <your_custom_platform>\source_windows64\include folder. For Linux, the hw_pcie_constants.h file is in the <your_custom_platform>/linux64/driver directory.

Note: The ACL_BOARD_PKG_NAME variable setting must match the name attribute of the board_env element that you specified in the board_env.xml file.

5. Define the Device ID, Subsystem Vendor ID, Subsystem Device ID, and Revision ID, as defined in the Device Identification Registers for Arria 10 PCIe Hard IP section.

Note: The PCIe IDs in the hw_pcie_constants.h file must match the parameters in the PCIe controller hardware.

6. Update your Custom Platform's board.qsys Qsys Pro system and the hw_pcie_constants.h file with the IDs defined in 5 on page 15.

7. For Windows, update DeviceList fields in the <your_custom_platform>\windows64\driver\acl_boards_a10_ref.inf file to match your PCIe ID values and then rename the file to acl_board_<your_custom_platform>.inf.

Note: The <your_custom_platform> string in acl_board_<your_custom_platform>.inf must match the string you specify for the name field in the board_env.xml file.

8. Run make in the <your_custom_platform>/source directory to generate the driver.

Related Links
Device Identification Registers for Arria 10 PCIe Hard IP on page 23
To build PCIe hardware, you must set PCIe IDs related to the device hardware.

2.7 Changing the Device Part Number

When porting the Arria 10 GX FPGA Development Kit Reference Platform to your own board, change the device part number, where applicable, to the part number of the device on your board.

Update the device part number in the following files within the <your_custom_platform>/hardware/<board_name> directory:

- In the device.tcl file, change the device part number in the set global assignment -name DEVICE 10AX115S2F45I1SG QSF assignment. The updated device number will appear in the base.qsf, top.qsf, and top_synth.qsf files.
- In the board.qsys, acl_ddr4_a10.qsys, and acl_ddr4_a10_core.qsys files, change all occurrences of 10AX115S2F45I1SG.

2.8 Connecting the Memory in the Arria 10 Custom Platform

Calibrate the external memory IP and controllers in your Custom Platform, and connect them to the host.
1. In your Custom Platform, instantiate your external memory IP based on the information in the DDR4 as Global Memory for OpenCL Applications section. Update the information pertaining to the global_mem element in the <your_custom_platform>/hardware/<board_name>/board_spec.xml file.

2. Remove the boardtest hardware configuration file that you created during the integration of your Custom Platform with the Intel FPGA SDK for OpenCL.

   The environment variable ALTERAOCLSDKROOT points to the location of the SDK installation.

4. Reprogram the FPGA with the new boardtest hardware configuration file and then reboot your machine.

5. Modify the wait_for_uniphy function in the acl_pcie_device.cpp MMD source code file to exit after checking the UniPHY status register. Rebuild the MMD software.
   For Windows, the acl_pcie_device.cpp file is in the <your_custom_platform>/source/host/mmd folder. For Linux, the acl_pcie_device.cpp file is in the <your_custom_platform>/source/host/mmd directory.

6. Run the aocl diagnose SDK utility and confirm that the host reads back both the version ID and the value 0 from the uniphy_status component. The utility should return the message Uniphy are calibrated.

7. Consider analyzing your design in the SignalTap™ II Logic Analyzer to confirm the successful calibration of all memory controllers.

Related Links
- DDR4 as Global Memory for OpenCL Applications on page 32
  The Arria 10 GX FPGA Development Kit has one bank of 2GB x72 DDR4-2400 SDRAM.
- Integrating Your Arria 10 Custom Platform with the Intel FPGA SDK for OpenCL on page 12
  After you modify your Quartus Prime design files, integrate your Custom Platform with the Intel FPGA SDK for OpenCL.

2.9 Modifying the Kernel PLL Reference Clock

The Arria 10 GX FPGA Reference Platform uses an external 125 MHz clock as a reference for the I/O PLL. The I/O PLL relies on this reference clock to generate the internal kernel_clk clock, and the kernel_clk2x clock that runs at twice the frequency of kernel_clk. When porting the a10_ref Reference Platform to your own board using a different reference clock, update the board.qsys and top.sdc files with the new reference clock speed.
1. In the `<your_custom_platform>/hardware/<board_name>/board.qsys` file, update the `REF_CLK_RATE` parameter value on the `kernel_clk_gen` IP module.

2. In the `<your_custom_platform>/hardware/<board_name>/top.sdc` file, update the `create_clock` assignment for `kernel_pll_refclk`.

3. [Optional] In the `<your_custom_platform>/hardware/<board_name>/top.v` file, update the comment for the `kernel_pll_refclk` input port.

After you update the `board.qsys` and the `top.sdc` files, the `post_flow_pr.tcl` script will automatically determine the I/O PLL reference frequency and compute the correct PLL settings.

### 2.10 Integrating an OpenCL Kernel in Your Arria 10 Custom Platform

After you establish host communication and connect the external memory, test the FPGA programming process from kernel creation to program execution.

1. Perform the steps outlined in `ALTERAOCLSDKROOT/board/custom_platform_toolkit/tests/README.txt` file to build the hardware configuration file from the `ALTERAOCLSDKROOT/board/custom_platform_toolkit/tests/boardtest/boardtest.cl` kernel source file.

   The environment variable `ALTERAOCLSDKROOT` points to the location of the Intel FPGA SDK for OpenCL installation.

2. Program your FPGA device with the hardware configuration file you created in 1 on page 17 and then reboot your machine.

3. Remove the early-exit modification in the `version_id_test` function in the `acl_pcie_device.cpp` file that you implemented when you established communication between the board and the host interface.

   For Windows, the `acl_pcie_device.cpp` file is in the `<your_custom_platform>/source/host/mmd` folder. For Linux, the `acl_pcie_device.cpp` file is in the `<your_custom_platform>/source/host/mmd` directory.

4. Invoke the `aocl diagnose <device_name>` command, where `<device_name>` is the string you define in your Custom Platform to identify each board.

   By default, `<device_name>` is the acl number (for example, acl0 to acl31) that corresponds to your FPGA device. In this case, invoke the `aocl diagnose acl0` command.

5. Build the `boardtest` host application using the `.sln` file (Windows) or `Makefile` (Linux) in the SDK's Custom Platform Toolkit.

   For Windows, the `.sln` file for Windows is in the `ALTERAOCLSDKROOT\board\custom_platform_toolkit\tests\boardtest\host` folder. For Linux, the `Makefile` is in the `ALTERAOCLSDKROOT/board/custom_platform_toolkit/tests/boardtest` directory.

6. Set the environment variable `CL_CONTEXT_COMPILER_MODE_ALTERA` to a value of 3 and run the `boardtest` host application.
For more information on CL_CONTEXT_COMPILER_MODE_ALTERA, refer to Troubleshooting Arria 10 GX FPGA Development Kit Reference Platform Porting Issues.

Related Links
• Establishing Arria 10 Custom Platform Host Communication on page 13
  After modifying and rebranding the Arria 10 GX FPGA Development Kit Reference Platform to your own Custom Platform, use the tools and utilities in your Custom Platform to establish communication between your FPGA accelerator board and your host application.
• Troubleshooting Arria 10 GX FPGA Development Kit Reference Platform Porting Issues on page 19
  Set Intel FPGA SDK for OpenCL-specific environment variables to help diagnose Custom Platform design problems.

2.11 Guaranteeing Timing Closure in the Arria 10 Custom Platform

When modifying the Arria 10 GX FPGA Development Kit Reference Platform into your own Custom Platform, ensure that guaranteed timing closure holds true for your Custom Platform.

1. Establish the floorplan of your design.
   Important: Consider all design criteria outlined in the FPGA System Design section of the Intel FPGA SDK for OpenCL Custom Platform Toolkit User Guide.

2. Compile several seeds of the ALTERAOCLSDKROOT/board/custom_platform_toolkit/tests/boardtest/boardtest.cl file until you generate a design that closes timing cleanly.
   To specify the seed number, include the --seed <N> option in your aoc command.

3. Copy the base.qdb, base.sdc, and pr_base_id.txt files from the ALTERAOCLSDKROOT/board/a10_ref/hardware/a10gx directory into your Custom Platform.

4. Use the flat.qsf file in the a10_ref Reference Platform as references to determine the type of information you must include in the flat.qsf file for your Custom Platform.
   The base.qsf, top.qsf, and top_synth.qsf files automatically inherit all the settings in the flat.qsf file. However, if you need to modify LogicLock Plus region or PR assignments, only make these changes in the base.qsf file.

5. Remove the ACL_QSH_COMPILE_CMD environment variable that you added when integrating your Custom Platform with the Intel FPGA SDK for OpenCL.

6. Confirm that you can use the .aocx file to reprogram the FPGA by invoking the aocl program acl0 boardtest.aocx command.

7. Ensure that the environment variable CL_CONTEXT_COMPILER_MODE_ALTERA is not set.

8. Run the boardtest_host executable.

Related Links
• Arria 10 FPGA System Design on page 34
To integrate all components, close timing, and deliver a post-fit netlist that functions in the hardware, you must first address several additional FPGA design complexities.

- FPGA System Design
- Integrating Your Arria 10 Custom Platform with the Intel FPGA SDK for OpenCL on page 12

After you modify your Quartus Prime design files, integrate your Custom Platform with the Intel FPGA SDK for OpenCL.

### 2.11.1 Generating the base.qdb Post-Fit Netlist for Your Arria 10 Custom Platform

To implement the Spectra-Q engine compilation flow, you must generate a base.qdb Quartus Prime Archive File for your Arria 10 Custom Platform.

The steps below represent a general procedure for regenerating the base.qdb file:

1. Port the system design and the flat.qsf file to your computing card.
2. Compile the `ALTERAOCLSDKROOT/board/custom_platform_toolkit/tests/boardtest/boardtest.cl` kernel source file using the base revision. Fix any timing failures and recompile the kernel until timing is clean.

`ALTERAOCLSDKROOT` points to the location of the Intel FPGA SDK for OpenCL installation.
3. Copy the generated `base.qdb`, `base.sdc`, and `pr_base_id.txt` files into your Custom Platform.
4. Using the default compilation flow, test the `base.qdb` file across several OpenCL design examples and confirm that the following criteria are satisfied:
   - All compilations close timing.
   - The OpenCL design examples achieve satisfactory Fmax.
   - The OpenCL design examples function on the accelerator board.

**Related Links**
- Integrating Your Arria 10 Custom Platform with the Intel FPGA SDK for OpenCL on page 12
- Provide a Timing-Closed Post-Fit Netlist on page 42

Each Intel FPGA SDK for OpenCL-compatible Reference and Custom Platform, such as the Arria 10 GX FPGA Development Kit Reference Platform, provides a timing-closed post-fit netlist that imports placement and routing information for all nodes clocked by non-kernel clocks.

### 2.12 Troubleshooting Arria 10 GX FPGA Development Kit Reference Platform Porting Issues

Set Intel FPGA SDK for OpenCL-specific environment variables to help diagnose Custom Platform design problems.
Table 3. Intel FPGA SDK for OpenCL-Specific Environment Variables for Identifying Custom Platform Design Problems

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL_HAL_DEBUG</td>
<td>Set this variable to a value of 1 to 5 to enable increasing debug output from the Hardware Abstraction Layer (HAL), which interfaces directly with the MMD layer.</td>
</tr>
<tr>
<td>ACL_PCIE_DEBUG</td>
<td>Set this variable to a value of 1 to 10000 to enable increasing debug output from the MMD. This variable setting is useful for confirming that the version ID register was read correctly and the UniPHY IP cores are calibrated.</td>
</tr>
<tr>
<td>ACL_PCIE_JTAG_CABLE</td>
<td>Set this variable to override the default <code>quartus_pgm</code> argument that specifies the cable number. The default is cable 1. If there are multiple USB-Blaster™ cables, you can specify a particular one here.</td>
</tr>
<tr>
<td>ACL_PCIE_JTAG_DEVICE_INDEX</td>
<td>Set this variable to override the default <code>quartus_pgm</code> argument that specifies the FPGA device index. By default, this variable has a value of 1. If the FPGA is not the first device in the JTAG chain, you can customize the value.</td>
</tr>
<tr>
<td>ACL_PCIE_USE_JTAG_PROGRAMMING</td>
<td>Set this variable to force the MMD to reprogram the FPGA using the JTAG cable instead of Partial Reconfiguration.</td>
</tr>
<tr>
<td>ACL_PCIE_DMA_USE_MSI</td>
<td>Set this variable if you want to use MSI for DMA transfers on Windows.</td>
</tr>
<tr>
<td>CL_CONTEXT_COMPILER_MODE_ALTERA</td>
<td>Unset this variable or set it to a value of 3. The OpenCL host runtime reprograms the FPGA as needed, which it does at least once during initialization. To prevent the host application from programming the FPGA, set this variable to a value of 3.</td>
</tr>
</tbody>
</table>
3 Arria 10 GX FPGA Development Kit Reference Platform
Design Architecture

Intel created the Arria 10 GX FPGA Development Kit Reference Platform (a10_ref) based on various design considerations. Familiarize yourself with these design considerations. Having a thorough understanding of the design decision-making process might help in the design of your own Intel FPGA SDK for OpenCL Custom Platform.

Host-to-Arria 10 FPGA Communication over PCIe on page 22
The Arria 10 GX FPGA Development Kit Reference Platform instantiates the Arria 10 PCIe hard IP to implement a host-to-device connection over PCIe.

DDR4 as Global Memory for OpenCL Applications on page 32
The Arria 10 GX FPGA Development Kit has one bank of 2GB x72 DDR4-2400 SDRAM.

Host Connection to OpenCL Kernels on page 34
The PCIe host needs to pass commands and arguments to the OpenCL kernels via the control register access (CRA) Avalon slave port that each OpenCL kernel generates.

Arria 10 FPGA System Design on page 34
To integrate all components, close timing, and deliver a post-fit netlist that functions in the hardware, you must first address several additional FPGA design complexities.

Dynamic PLL Reconfiguration on page 40
The Intel FPGA SDK for OpenCL relies on the post_flow_pr.tcl Tcl script and the instantiation of the acl_kernel_clk_a10 Qsys Pro component to modify kernel PLL.

Guaranteed Timing Closure of the Arria 10 GX FPGA Development Kit Reference Platform Design on page 40
Both the SDK and the Custom Platform contribute to the implementation of the SDK’s guaranteed timing closure feature.

Quartus Prime Compilation Flow and Scripts on page 44
The import_compile.tcl Tcl Script File in the Arria 10 GX FPGA Development Kit Reference Platform controls the Quartus Prime compilation flow.

Addition of Timing Constraints on page 49
In the Arria 10 FPGA Development Kit Reference Platform, the top.sdc file contains all timing constraints applicable before IP instantiation in Qsys Pro. The top_post.sdc file contains timing constraints applicable after Qsys Pro.

Connection of the Arria 10 GX FPGA Development Kit Reference Platform to the Intel FPGA SDK for OpenCL on page 49
A Custom Platform must include a board_env.xml file to describe its general contents to the Intel FPGA SDK for OpenCL Offline Compiler. For each hardware
design, your Custom Platform also requires a `board_spec.xml` file for each hardware design that describes the hardware.

**Arria 10 FPGA Programming Flow** on page 51

There are three ways to program the Arria 10 FPGA for the Arria 10 GX FPGA Development Kit Reference Platform: Flash, `quartus_pgm`, and Partial Reconfiguration.

**Host-to-Device MMD Software Implementation** on page 52

The Arria 10 GX FPGA Development Kit Reference Platform's MMD layer is a thin software layer that is essential for communication between the host and the board.

**Implementation of Intel FPGA SDK for OpenCL Utilities** on page 53

The Arria 10 GX FPGA Development Kit Reference Platform includes a set of Intel FPGA SDK for OpenCL utilities for managing the FPGA board.

**Arria 10 FPGA Development Kit Reference Platform Scripts** on page 55

The Arria 10 FPGA Development Kit Reference Platform includes a number of Tcl scripts in its `hardware/<board_name>/scripts` directory.

**Considerations in Arria 10 GX FPGA Development Kit Reference Platform Implementation** on page 55

The implementation of the Arria 10 GX FPGA Development Kit Reference Platform includes some workarounds that address certain Quartus Prime Pro Edition software known issues.

### 3.1 Host-to-Arria 10 FPGA Communication over PCIe

The Arria 10 GX FPGA Development Kit Reference Platform instantiates the Arria 10 PCIe hard IP to implement a host-to-device connection over PCIe.

#### 3.1.1 Instantiation of Arria 10 PCIe Hard IP with Direct Memory Access

The Arria 10 GX FPGA Development Kit Reference Platform instantiates the Arria 10 PCIe hard IP with direct memory access (DMA) to implement a host-to-device connection over PCIe.

**Dependencies**

- Arria 10 PCIe hard IP core
- *Parameter Settings* section of the *Arria 10 Avalon-MM DMA Interface for PCIe Solutions User Guide*

**Table 4. Highlights of Arria 10 PCIe Hard IP Parameter Settings**

Set the parameters for the Arria 10 PCIe hard IP in the parameter editor within the Quartus Prime Pro Edition software.

<table>
<thead>
<tr>
<th>Parameter(s)</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System Settings</strong></td>
<td></td>
</tr>
</tbody>
</table>
| **Application interface type** | Avalon-MM with DMA  
This Avalon Memory-Mapped (Avalon-MM) interface instantiates the embedded DMA of the PCIe hard IP core. |
| **Hard IP mode**            | Gen3x8, Interface: 256-bit, 250 MHz  
Number of Lanes: x8  
Lane Rate: Gen3 (8.0 Gbps) |

*continued...*
### Parameter(s) and Setting

**Note:** This setting is the fastest configuration that the Avalon-MM DMA slave interface currently supports.

<table>
<thead>
<tr>
<th>Parameter(s)</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rx Buffer credit allocation</td>
<td>Low</td>
</tr>
<tr>
<td>Arria 10 Avalon-MM Settings</td>
<td></td>
</tr>
<tr>
<td>Export MSI/MSI-X conduit interfaces</td>
<td>Enabled</td>
</tr>
<tr>
<td>Instantiate Internal Descriptor Controller</td>
<td>Enabled</td>
</tr>
<tr>
<td>Address width of accessible PCIe memory space</td>
<td>64 bits</td>
</tr>
<tr>
<td>Base Address Register (BAR) Settings</td>
<td>This design uses two BARs.</td>
</tr>
<tr>
<td>Base Address Registers (BARs)</td>
<td>For BAR 0, set <strong>Type</strong> to <strong>64-bit prefetchable memory</strong>. The <strong>Size</strong> parameter setting is disabled because the <strong>Instantiate Internal Descriptor Controller</strong> parameter is enabled in the Avalon-MM system settings. For BAR 4, set <strong>Type</strong> to <strong>64-bit prefetchable memory</strong>, and set <strong>Size</strong> to <strong>256 KBytes - 18 bits</strong>. BAR 4 is used to connect PCIe to the OpenCL kernel systems and other board modules.</td>
</tr>
</tbody>
</table>

### Related Links
- Parameter Settings for Arria 10 Avalon-MM DMA Interface for PCIe Solutions
- Arria 10 Avalon-MM DMA for PCI Express

### 3.1.2 Device Identification Registers for Arria 10 PCIe Hard IP

To build PCIe hardware, you must set PCIe IDs related to the device hardware.

**Table 5. Device Hardware-Related PCIe ID Registers**

<table>
<thead>
<tr>
<th>ID Register Name</th>
<th>ID Provider</th>
<th>Description</th>
<th>Parameter Name in PCIe IP Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vendor ID</td>
<td>PCI-SIG®</td>
<td>Identifies the FPGA manufacturer. Always set this register to <strong>0x1172</strong>, which is Intel vendor ID.</td>
<td>vendor_id_hwtcl</td>
</tr>
<tr>
<td>Device ID</td>
<td>Intel</td>
<td>Describes the PCIe configuration on the FPGA according to Intel's internal guideline. Set the device ID to the device code of the FPGA on your accelerator board.</td>
<td>device_id_hwtcl</td>
</tr>
</tbody>
</table>

**continued...**
For the Arria 10 GX FPGA Development Kit Reference Platform, set the Device ID register to 0x2494, which signifies Gen 3 speed, 8 lanes, Arria 10 device family, and Avalon-MM interface, respectively. Refer to Table 6 on page 25 for more information.

### Revision ID

When setting this ID, ensure that it matches the following revision IDs:
- For Windows, the revision ID specified for the DeviceList field in the `<your_custom_platform>`\windows64\driver\acl_boards_<your_custom_platform>.inf file.
- For Linux, the revision ID specified for the ACL_PCI_REVISION variable in the `<your_custom_platform>/linux64/driver/hw_pcie_constants.h` file.

### Class Code

Intel

The Intel FPGA SDK for OpenCL utility checks the base class value to verify whether the board is an OpenCL device.
Do not modify the class code settings.
- Base class: 0x12
- Sub class: 0x00
- Programming interface: 0x01

### Subsystem Vendor ID

Board vendor

Identifies the manufacturer of the accelerator board.
Set this register to the vendor ID of manufacturer of your accelerator board. For the a10_ref Reference Platform, the subsystem vendor ID is 0x1172.
If you are a board vendor, set this register to your vendor ID.

### Subsystem Device ID

Board vendor

Identifies the accelerator board.
The SDK uses this ID to identify the board because the software might perform differently on different boards. If you create a Custom Platform that supports multiple boards, use this ID to distinguish between the boards. Alternatively, if you have multiple Custom Platforms, each supporting a single board, you can use this ID to distinguish between the Custom Platforms.

**Important:** Make this ID unique to your Custom Platform. For example, for the a10_ref Reference Platform, the ID is 0xA151.

---

You can find these PCIe ID definitions in the PCIe controller instantiated in the ALTERAOCLSDKROOTboard/a10_ref/hardware/a10gx/board.qsys Qsys Pro System File. These IDs are necessary in the driver and the SDK's programming flow. The kernel driver uses the Vendor ID, Subsystem Vendor ID and the Subsystem Device ID to identify the boards it supports. The SDK's programming flow checks the Device ID to ensure that it programs a device with a .aocx Intel FPGA SDK for OpenCL Offline Compiler executable file targeting that specific device.
Table 6. Intel FPGA SDK for OpenCL’s Numbering Convention for PCIe Hard IP Device ID

<table>
<thead>
<tr>
<th>Location in ID</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:14</td>
<td>RESERVED</td>
</tr>
<tr>
<td>13:12</td>
<td>Speed</td>
</tr>
<tr>
<td>11</td>
<td>RESERVED</td>
</tr>
<tr>
<td>10:8</td>
<td>Number of lanes</td>
</tr>
<tr>
<td>7:4</td>
<td>Device family</td>
</tr>
<tr>
<td>3</td>
<td>1 — Soft IP (SIP)</td>
</tr>
<tr>
<td>2:0</td>
<td>Qsys Pro PCIe interface type</td>
</tr>
</tbody>
</table>

3.1.3 Instantiation of the version_id Component

Intel specifies an additional version ID and uses it to verify the address map of the system. The host verifies the version ID of the Arria 10 GX FPGA Development Kit Reference Platform when instantiating the version_id component that connects to the PCIe Avalon master.

The version ID for the a10_ref Reference Platform is A0C7C1E2.

Before communicating with any part of the FPGA system, the host first reads from this version_id register to confirm the following:

- The PCIe can access the FPGA fabric successfully.
- The address map matches the map in the MMD software.
Update the VERSION_ID parameter in the version_id component to a new value with every slave addition or removal from the PCIe BAR 4 bus, or whenever the address map changes.

3.1.4 Definitions of Arria 10 FPGA Development Kit Reference Platform Hardware Constraints in Software Headers Files

After you build the PCIe component in your hardware design, you need a software layer to communicate with the board via PCIe. To enable communication between the board and the host interface, define the hardware constants for the software in header files.

The two header files that describe the hardware design to the software are in the following locations:

- For Windows systems, the header files are in the `ALTERAOCLSDKROOT\board \a10_ref\source\include` folder, where `ALTERAOCLSDKROOT` is the path to the SDK installation.
- For Linux systems, the header files are in the `ALTERAOCLSDKROOT/board/ a10_ref/linux64/driver` directory.

<table>
<thead>
<tr>
<th>Table 7. Arria 10 GX FPGA Development Kit Reference Platform Header Files</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Header File Name</strong></td>
</tr>
<tr>
<td>hw_pcie_constants.h</td>
</tr>
<tr>
<td>hw_pcie_dma.h</td>
</tr>
<tr>
<td>- <code>ACL_PCIE_DMA_ONCHIP_RD_FIFO_BASE</code> refers to the Qsys Pro address of <code>rd_dts_slave</code> on the PCIe IP's <code>dma_rd_master</code>.</td>
</tr>
<tr>
<td>- <code>ACL_PCIE_DMA_ONCHIP_WR_FIFO_BASE</code> refers to the Qsys Pro address of <code>wr_dts_slave</code> on the PCIe IP's <code>dma_rd_master</code>.</td>
</tr>
<tr>
<td>Update these addresses whenever you change the board design. Refer to the Direct Memory Access section for more information.</td>
</tr>
<tr>
<td>- <code>ACL_PCIE_DMA_TABLE_SIZE</code> refers to the DMA descriptor FIFO depth connected to the DMA. When using the internal descriptor controller, refer to the DMA Descriptor Controller Registers section in the Arria 10 Avalon-MM DMA Interface for PCIe Solutions User Guide for the required size.</td>
</tr>
<tr>
<td>- <code>ACL_PCIE_DMA_PAGES_LOCKED</code> specifies the maximum pages you can lock. You may modify this constant to improve performance.</td>
</tr>
<tr>
<td>- <code>ACL_PCIE_DMA_NONAligned_TRANS_LOG</code> specifies the starting and ending power-of-two values that non-aligned DMA transfers should have. You may modify this constant to improve performance.</td>
</tr>
</tbody>
</table>

Related Links

- [Direct Memory Access](#) on page 28
  The Arria 10 GX FPGA Development Kit Reference Platform relies on the PCIe hard IP core's soft DMA engine to transfer data.
- [Device Identification Registers for Arria 10 PCIe Hard IP](#) on page 23
  To build PCIe hardware, you must set PCIe IDs related to the device hardware.
- [DMA Descriptor Controller Registers](#)
A PCIe kernel driver is necessary for the OpenCL runtime library to access your board design via a PCIe bus.

Use the Intel FPGA SDK for OpenCL install utility to install the kernel driver.

The a10_ref Reference Platform

- For Windows systems, the driver is in the `<path_to_al0pciedk>\windows64\driver` folder.

  The kernel driver, the WinDriver application programming interface (API), is a third-party driver from Jungo Connectivity Ltd. For more information about the WinDriver, refer to the Jungo Connectivity Ltd. website or contact a Jungo Connectivity representative.

- For Linux, an open-source MMD-compatible kernel driver is in the `<path_to_al0pciedk>/linux64/driver` directory. The table below highlights some of the files that are available in this directory.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pcie_linux_driver_exports.h</td>
<td>Header file that defines the special commands that the kernel driver supports. The installed kernel driver works as a character device. The basic operations to the driver are <code>open()</code>, <code>close()</code>, <code>read()</code>, and <code>write()</code>. To execute a complicated command, create a variable as an <code>acl_cmd struct</code> type, specify the command with the proper parameters, and then send the command through a <code>read()</code> or <code>write()</code> operation. This header file defines the interface of the kernel driver, which the MMD layer uses to communicate with the device.</td>
</tr>
<tr>
<td>aclpci.c</td>
<td>File that implements the Linux kernel driver's basic structures and functions, such as the <code>init</code>, <code>remove</code>, and <code>probe</code> functions, as well as hardware design-specific functions that handle interrupts. For more information on the interrupt handler, refer to the Message Signaled Interrupts section.</td>
</tr>
<tr>
<td>aclpci fileio.c</td>
<td>File that implements the kernel driver's file I/O operations. The kernel driver that is available with the a10_ref Reference Platform supports four file I/O operations: <code>open()</code>, <code>close()</code>, <code>read()</code>, and <code>write()</code>. Implementing these file I/O operations allows the AOCL user program to access the kernel driver through the file I/O system calls (that is, <code>open</code>, <code>read</code>, <code>write</code>, or <code>close</code>).</td>
</tr>
<tr>
<td>aclpci cmd.c</td>
<td>File that implements the specific commands defined in the <code>pcie_linux_driver_exports.h</code> file. These special commands include <code>SAVE_PCI_CONTROL_REGS</code>, <code>LOAD_PCI.Control_REGS</code>, <code>DO_PR</code>, <code>GET_PCI_SLOT_INFO</code>, etc.</td>
</tr>
<tr>
<td>aclpci dma.c</td>
<td>File that implements DMA-related routines in the kernel driver. Refer to the Direct Memory Access section for more information.</td>
</tr>
<tr>
<td>aclpci pr.c</td>
<td>File that implements PR-related routines in the kernel driver. Refer to the Partial Reconfiguration section for more information.</td>
</tr>
<tr>
<td>aclpci queue.c</td>
<td>File that implements a queue structure for use in the kernel driver to simplify programming.</td>
</tr>
</tbody>
</table>
Related Links

- **Partial Reconfiguration** on page 31
  The Arria 10 GX FPGA Development Kit Reference Platform uses PR as a default mechanism to reconfigure the OpenCL kernel-related partition of the design without altering the static board interface that is in a running state.

- **aocl install** on page 53
  The install utility in the Arria 10 GX FPGA Development Kit Reference Platform installs the kernel driver on the host computer.

- **Message Signaled Interrupt** on page 30
  The Arria 10 GX FPGA Development Kit Reference Platform uses one MSI line for both DMA and the kernel interface.

- **Direct Memory Access** on page 28
  The Arria 10 GX FPGA Development Kit Reference Platform relies on the PCIe hard IP core's soft DMA engine to transfer data.

- **Jungo Connectivity Ltd. website**

### 3.1.6 Direct Memory Access

The Arria 10 GX FPGA Development Kit Reference Platform relies on the PCIe hard IP core's soft DMA engine to transfer data. The Arria 10 PCIe hard IP core's DMA interface is instantiated as a soft IP inside the PCIe hardware when the **Avalon-MM with DMA** application interface type is selected in the IP parameter editor.

**Note:** The DMA interface is capable of full duplex data transfers. However, the driver handles one read or write transfer at a time.

#### Hardware Considerations

The instantiation process exports the DMA controller slave ports (that is, `rd_dts_slave` and `wr_dts_slave`) and master ports (that is, `rd_dcm_master` and `wr_dcm_master`) into the PCIe module. Two additional master ports, `dma_rd_master` and `dma_wr_master`, are exported for DMA read and write operations, respectively. For the DMA interface to function properly, all these ports must be connected correctly in the `board.qsys` Qsys Pro system, where the PCIe hard IP is instantiated.

At the start of DMA transfer, the DMA Descriptor Controller reads from the DMA descriptor table in user memory, and stores the status and the descriptor table into a FIFO address. There are two FIFO addresses: Read Descriptor FIFO address and Write Descriptor FIFO address. After storing the descriptor table into a FIFO address, DMA transfer into the FIFO address can occur. The `dma_rd_master` port, which moves data from user memory to the device, must connect to the `rd_dts_slave` and `wr_dts_slave` ports. Because the `dma_rd_master` port connects to DDR4 memory also, the locations of the `rd_dts_slave` and `wr_dts_slave` ports in the address space must be defined in the `hw_pcie_dma.h` file.

The `rd_dcm_master` and `wr_dcm_master` ports must connect to the `txs` port. At the end of the DMA transfer, the DMA controller writes the MSI data and the **done** status into the user memory via the `txs` slave. The `txs` slave is part of the PCIe hard IP in `board.qsys`. 
All modules that use DMA must connect to the `dma_rd_master` and `dma_wr_master` ports. For DDR4 memory connection, Intel recommends implementing an additional pipeline to connect the two 256-bit PCIe DMA ports to the 512-bit memory slave. For more information, refer to the *DDR4 Connection to PCIe Host* section.

**Software Considerations**

The MMD layer uses DMA to transfer data if it receives a data transfer request that satisfies both of the following conditions:

- A transfer size that is greater than 1024 bytes.
- The starting addresses for both the host buffer and the device offset are aligned to 64 bytes.

**Related Links**

- Definitions of Arria 10 FPGA Development Kit Reference Platform Hardware Constraints in Software Headers Files on page 26
  
  To enable communication between the board and the host interface, define the hardware constants for the software in header files.

- Arria 10 DMA Avalon-MM DMA Interface to the Application Layer

- DMA Descriptor Controller Registers

- Implementing a DMA Transfer on page 29
  
  Implement a DMA transfer in the MMD on Windows (`ALTERAOCLSDKROOT\board\a10_ref\source\host\mmd\acl_pcie_dma_windows.cpp`) or in the kernel driver on Linux (`ALTERAOCLSDKROOT/board/a10_ref/linux64/driver/aclpci_dma`).

- DDR4 Connection to PCIe Host on page 33
  
  Connect all global memory systems in the Arria 10 GX FPGA Development Kit Reference Platform to the host via the OpenCL Memory Bank Divider component.

**3.1.6.1 Implementing a DMA Transfer**

Implement a DMA transfer in the MMD on Windows (`ALTERAOCLSDKROOT\board\a10_ref\source\host\mmd\acl_pcie_dma_windows.cpp`) or in the kernel driver on Linux (`ALTERAOCLSDKROOT/board/a10_ref/linux64/driver/aclpci_dma`).

**Note:**

For Windows, the Jungo WinDriver imposes a 5000 to 10000 limit on the number of interrupts received per second in user mode. This limit translates to a 2.5 gigabytes per second (GBps) to 5 GBps DMA bandwidth when a full 128-entry table of 4 KB page is transferred per interrupt.

On Windows, polling is the default method for maximizing PCIe DMA bandwidth at the expense of CPU run time. To use interrupts instead of polling, assign a non-NULL value to the `ACL_PCIE_DMA_USE_MSI` environment variable.

The steps below describe the general procedure for implementing a DMA transfer:

1. Verify that the previous DMA transfer sent all the requested bytes of data.
2. Map the virtual memories that are requested for DMA transfer to physical addresses.
Note: The amount of virtual memory that can be mapped at a time is system dependent. Large DMA transfers will require multiple mapping or unmapping operations. For a higher bandwidth, map the virtual memory ahead in a separate thread that is in parallel to the transfer.

3. Set up the DMA descriptor table on local memory.

4. Write the location of the DMA descriptor table, which is in user memory, to the DMA control registers (that is, RC Read Status and Descriptor Base and RC Write Status and Descriptor Base).

5. Write the Qsys Pro address of descriptor FIFOs to the DMA control registers (that is EP Read Descriptor FIFO Base and EP Write Status and Descriptor FIFO Base).

6. Write the start signal to the RD_DMA_LAST_PTR and WR_DMA_LAST_PTR DMA control registers.

7. After the current DMA transfer finishes, repeat the procedure to implement the next DMA transfer.

Related Links

Direct Memory Access on page 28
The Arria 10 GX FPGA Development Kit Reference Platform relies on the PCIe hard IP core's soft DMA engine to transfer data.

3.1.7 Message Signaled Interrupt

The Arria 10 GX FPGA Development Kit Reference Platform uses one MSI line for both DMA and the kernel interface.

Two different modules generate the signal for the MSI line. The DMA controller in the PCIe hard IP core generates the DMA's MSI. The PCI Express interrupt request (IRQ) module (that is, the ALTERAOCLSDKROOT/board/a10_ref/hardware/a10gx/ip/irq_controller directory) generates the kernel interface's MSI.

For more information on the PCI Express IRQ module, refer to Handling PCIe Interrupts webpage.

Hardware Considerations

In ALTERAOCLSDKROOT/board/a10_ref/hardware/a10gx/board.qsys, the DMA MSI is connected internally; however, you must connect the kernel interface interrupt manually. For the kernel interface interrupt, the PCI Express IRQ module is instantiated as pcie_irq_0 in board.qsys. The kernel interface interrupts connections are as follows:
• The `kernel_irq_to_host` port from the OpenCL Kernel Interface (kernel_interface) connects to the interrupt receiver, which allows the OpenCL kernels to signal the PCI Express IRQ module to send an MSI.

• The PCIe hard IP's `msi_intfc` port connects to the `MSI_Interface` port in the PCI Express IRQ module. The kernel interface interrupt receives the MSI address and the data necessary to generate the interrupt via `msi_intfc`.

• The `IRQ_Gen_Master` port on the PCI Express IRQ module, which is used to write the MSI, connects to the `txs` port on the PCIe hard IP.

• The `IRQ_Read_Slave` and `IRQ_Mask_Slave` ports connect to the `pipe_stage_host_ctrl` module on Bar 4. After receiving an MSI, the user driver can read the `IRQ_Read_Slave` port to check the status of the kernel interface interrupt, and read the `IRQ_Mask_Slave` port to mask the interrupt.

**Software Considerations**

The interrupt service routine in the Linux driver checks which module generates the interrupt. For the DMA's MSI, the driver reads the DMA descriptor table's status bit in local memory, as specified in the Read DMA Example section of the *Arria 10 Avalon-MM DMA Interface for PCIe Solutions User Guide*. For kernel interface's MSI, the driver reads the interrupt line sent by the kernel interface.

The interrupt service routine involves the following tasks:

1. Check DMA status on the DMA descriptor table.
2. Read the kernel status from the `IRQ_READ_SLAVE` port on the PCI Express IRQ module.
3. If a kernel interrupt was triggered, mask the interrupt by writing to the `IRQ_MASK_SLAVE` port on the PCI Express IRQ module. Then, execute the kernel interrupt service routine.
4. If a DMA interrupt was triggered, reset the DMA descriptor table and execute the DMA interrupt service routine.
5. If applicable, unmask a masked kernel interrupt.

**Related Links**

- Handling PCIe Interrupts
- Read DMA Example

### 3.1.8 Partial Reconfiguration

The Arria 10 GX FPGA Development Kit Reference Platform uses PR as a default mechanism to reconfigure the OpenCL kernel-related partition of the design without altering the static board interface that is in a running state.

You can only use PR when the static board interface, generated during base compilations, matches the static region of the design that is used to compile the OpenCL kernel's PR region.

For Windows MMD implementation, the `ALTERAOCLSDKROOT\board\a10_ref\source\host\mmd\acl_pcie_config.cpp` file contains the MMD code that communicates with the PR configuration controller within the static region of the
design. The `program_core_with_PR_file` function within the `acl_pciie_config.cpp` file requires a handle to the PR bitstream and the length of the PR bitstream in order to perform the PR operation.

For Linux driver implementation, the `ALTERAOCLSDKROOT/board/a10_ref/linux64/driver/aclpci_pr.c` file includes the main host driver routine that communicates with the PR configuration controller within the static region of the design. The `aclpci_pr` function within the `acl_pci_pr.c` file requires the following information in order to perform the PR operation:

- A handle to the board
- A handle to the PR bitstream
- The length of the PR bitstream

After verifying that the device is opened, the bitstream is of adequate length, and the PCIe endpoint of the device is reachable, the `aclpci_pr` function writes 0x1 to the PR IP status register. Then, the `aclpci_pr` function writes the complete bitstream, 32 bits at a time, to the PR IP. After the bitstream transfer is complete, the `aclpci_pr` function performs a read operation to the PR IP status register to verify whether PR is successful. A return value of 0x14 indicates a successful PR operation; any other return value indicates an error.

To override the default reconfiguration mechanism, set the `ACL_PCIE_USE_JTAG_PROGRAMMING` environment variable, as shown below:

- For Windows, type `set ACL_PCIE_USE_JTAG_PROGRAMMING=1` at the command prompt.
- For Linux, type `export ACL_PCIE_USE_JTAG_PROGRAMMING=1` at the command prompt.

Setting `ACL_PCIE_USE_JTAG_PROGRAMMING` specifies that JTAG full-chip configuration is the default mechanism for reconfiguring the device.

### 3.2 DDR4 as Global Memory for OpenCL Applications

The Arria 10 GX FPGA Development Kit has one bank of 2GB x72 DDR4-2400 SDRAM. The DDR4 SDRAM is a daughtercard that is mounted to the development kit's HiLo connector.

In the current version of the a10_ref Reference Platform, all Qsys Pro components related to the DDR4 global memory are now part of the `ALTERAOCLSDKROOT/board/a10_ref/hardware/a10gx/acl_ddr4_a10.qsys` Qsys Pro subsystem within `board.qsys`. In addition, the location of the clock domain crossings has changed to increase the number of blocks operating in the slower PCIe domain. With this modified structure, you can add multiple memories with different clock domains to the system.

If you have a Custom Platform that is ported from a previous version of the a10_ref Reference Platform, you have the option to modify your Custom Platform as described above. This modification is not mandatory.

**Dependencies**

- DDR4 external memory interfaces
For more information on the DDR4 external memory interface IP, refer to the DDR2, DDR3, and DDR4 SDRAM Board Design Guidelines section in External Memory Interface Handbook Volume 2: Design Guidelines.

To use the DDR4 SDRAM as global memory for Intel FPGA SDK for OpenCL designs, you must instantiate the memory controller IP, connect the memory IP to the host, and connect the memory IP to the kernel.

Related Links
DDR2, DDR3, and DDR4 SDRAM Board Design Guidelines

3.2.1 DDR4 IP Instantiation

The Arria 10 GX FPGA Development Kit Reference Platform uses one DDR4 Controller IP to communicate with the physical memory.

Table 9. DDR4 SDRAM Controller IP Configuration Settings

<table>
<thead>
<tr>
<th>IP Parameter</th>
<th>Configuration Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timing Parameters</td>
<td>As per the computing card’s data specifications.</td>
</tr>
<tr>
<td>Avalon Width Power of 2</td>
<td>Currently, OpenCL does not support non-power-of-2 bus widths. As a result, the a10_ref Reference Platform uses the option that forces the DDR4 controller to power of 2. Use the additional pins of this x72 core for error checking between the memory controller and the physical module.</td>
</tr>
<tr>
<td>Byte Enable Support</td>
<td>Enabled</td>
</tr>
<tr>
<td>Performance</td>
<td>Enabling the reordering of DDR4 memory accesses and a deeper command queue look-ahead depth might provide increased bandwidth for some OpenCL kernels. For a target application, adjust these and other parameters as necessary. Note: Increasing the command queue look-ahead depth allows the DDR4 memory controller to reorder more memory accesses to increase efficiency, which improves overall memory throughput.</td>
</tr>
<tr>
<td>Debug</td>
<td>Disabled for production.</td>
</tr>
</tbody>
</table>

3.2.2 DDR4 Connection to PCIe Host

Connect all global memory systems in the Arria 10 GX FPGA Development Kit Reference Platform to the host via the OpenCL Memory Bank Divider component.

The DDR4 IP core has one bank where its width and address configurations match those of the DDR4 SDRAM. Intel tunes the other parameters such as burst size, pending reads, and pipelining. These parameters are customizable for an end application or board design.

The Avalon master interfaces from the OpenCL Memory Bank Divider component connect to their respective memory controllers. The Avalon slave connects to the PCIe and DMA IP core. Implementations of appropriate clock crossing and pipelining are based on the design floorplan and the clock domains specific to the computing card. The OpenCL Memory Bank Divider section in the Intel FPGA SDK for OpenCL Custom Platform Toolkit User Guide specifies the connection details of the snoop and memorg ports.

Important: Instruct the host to verify the successful calibration of the memory controller.
The ALTERAOCKROOT/board/a10_ref/hardware/a10gx/board.qsys Qsys Pro system uses a custom UniPHY Status to AVS IP component to aggregate different UniPHY status conduits into a single Avalon slave port named s. This slave port connects to the pipe_stage_host_ctrl component so that the PCIe host can access it.

**Related Links**
OpenCL Memory Bank Divider

### 3.2.3 DDR4 Connection to the OpenCL Kernel

The OpenCL kernel needs to connect directly to the memory controller in the Arria 10 GX FPGA Development Kit Reference Platform via a FIFO-based clock crosser.

A clock crosser is necessary because the kernel interface for the compiler must be clocked in the kernel clock domain. In addition, the width, address width, and burst size characteristics of the kernel interface must match those specified in the OpenCL Memory Bank Divider connecting to the host. Appropriate pipelining also exists between the clock crosser and the memory controller.

### 3.3 Host Connection to OpenCL Kernels

The PCIe host needs to pass commands and arguments to the OpenCL kernels via the control register access (CRA) Avalon slave port that each OpenCL kernel generates. The OpenCL Kernel Interface component exports an Avalon master interface (kernel_cra) that connects to this slave port. The OpenCL Kernel Interface component also generates the kernel reset (kernel_reset) that resets all logic in the kernel clock domain.

The Arria 10 FPGA Development Kit Reference Platform has one DDR4 memory bank. As a result, the Reference Platform instantiates the OpenCL Kernel Interface component and sets the Number of global memory systems parameter to 1.

### 3.4 Arria 10 FPGA System Design

To integrate all components, close timing, and deliver a post-fit netlist that functions in the hardware, you must first address several additional FPGA design complexities.

Examples of design complexities:

- Designing a robust reset sequence
- Establishing a design floorplan
- Managing global routing
- Pipelining

Optimizations of these design complexities occur in tandem with one another in order to meet timing and board hardware optimization requirements.

#### 3.4.1 Clocks

Several clock domains affect the Qsys Pro hardware system of the Arria 10 GX FPGA Development Kit Reference Platform.
These clock domains include:
- 250 MHz PCIe clock
- 300 MHz DDR4 clock
- 50 MHz general clock (\texttt{config\_clk})
- 125 MHz kernel reference clock
- Kernel clock that can have any clock frequency

With the exception of the kernel clock, the a10\_ref Reference Platform is responsible for the timing closure of these clocks. However, because the board design must clock cross all interfaces in the kernel clock domain, the board design also has logic in the kernel clock domain. It is crucial that this logic is minimal and achieves an \textit{Fmax} higher than typical kernel performance.

\textbf{Related Links}

Guaranteed Timing Closure of the Arria 10 GX FPGA Development Kit Reference Platform Design on page 40
Both the SDK and the Custom Platform contribute to the implementation of the SDK's guaranteed timing closure feature.

\section*{3.4.2 Resets}

The Arria 10 GX FPGA Development Kit Reference Platform design includes the implementation of reset drivers.

These reset drivers include:
- The \texttt{por\_reset\_counter} in the \texttt{ALTERAOCLSDKROOT/board/a10\_ref/hardware/a10gx/board.qsys} Qsys Pro system implements the power-on-reset. The power-on-reset resets all the hardware on the device by issuing a reset for a number of cycles after the FPGA completes configuration.
- The PCIe bus issues a \texttt{perst} reset that resets all hardware on the device.
- The OpenCL Kernel Interface component issues the \texttt{kernel\_reset} that resets all logic in the kernel clock domain.

The power-on-reset and the \texttt{perst} reset are combined into a single \texttt{global\_reset}; therefore, there are only two reset sources in the system (that is, \texttt{global\_reset} and \texttt{kernel\_reset}). However, these resets are explicitly synchronized across the various clock domains, resulting in several reset interfaces.
Important Considerations Regarding Resets

- Synchronizing resets to different clock domains might cause several high fan-out resets.
  Qsys Pro automatically synchronizes resets to the clock domain of each connected component. In doing so, Qsys Pro instantiates new reset controllers with derived names that might change when the design changes. This name change makes it difficult to make and maintain global clock assignments to some of the resets. As a result, for each clock domain, there are explicit reset controllers. For example, `global_reset` drives `reset_controller_pcie` and `reset_controller_ddr4`; however, they are synchronized to the PCIe and DDR4 clock domains, respectively.

- Resets and clocks must work together to propagate reset to all logic.
  Resetting a circuit in a given clock domain involves asserting the reset over a number of clock cycles. However, your design may apply resets to the PLLs that generate the clocks for a given clock domain. This means a clock domain can hold in reset without receiving the clock edge that is necessary for synchronous resets. In addition, a clock holding in reset might prevent the propagation of a reset signal because it is synchronized to and from that clock domain. Avoid such situations by ensuring that your design satisfies the following criteria:
    - Generate the `global_reset` signal off the free-running `config_clk`.
    - The `ddr4_calibrate` IP resets the External Memory Interface controller separately.

- Apply resets to both reset interfaces of a clock-crossing bridge or FIFO component.
  FIFO content corruption might occur if only part of a clock-crossing bridge or a dual-clock FIFO component is reset. These components typically provide a reset input for each clock domain; therefore, reset both interfaces or none at all. For example, in the a10_ref Reference Platform, `kernel_reset` resets all the kernel clock-crossing bridges between DDR on both the `m0_reset` and `s0_reset` interfaces.

3.4.3 Floorplan

Intel establishes the floorplan of the Arria 10 GX FPGA Development Kit Reference Platform by iterating on the design and IP placements.

Dependencies

- Partial Reconfiguration
- Chip Planner
- LogicLock Plus regions

Intel performed the following tasks iteratively to derive the floorplan of the a10_ref Reference Platform:

1. Compile a design without any region or floorplanning constraints.
   Intel recommends that you compile the design with several seeds.

2. Examine the placement of the IP cores (for example, PCIe, DDR4, Avalon interconnect pipeline stages and adapters) for candidate locations, as determined by the Quartus Prime Pro Edition software's Fitter. In particular, Intel recommends examining the seeds that meet or almost meet the timing constraints.
For the a10_ref Reference Platform, the PCIe I/O is located in the lower left corner of the Arria 10 FPGA. The DDR4 I/O is located on the top part of the left I/O column of the device. Because the placements of the PCIe and DDR4 IP components tend to be close to the locations of their respective I/Os, you can apply LogicLock Plus regions to constrain the IP components to those candidate regions.

Figure 2. Floorplan of the Arria 10 FPGA Development Kit Reference Platform

As shown in this Chip Planner view of the floorplan, the two LogicLock Plus regions spread out between the PCIe I/O and the top region of the left I/O column (that is, the DDR4 I/O area).
• The largest LogicLock Plus region (Region 1) covers the PCIe I/O and contains most of the static board interface logic.

• Regions 2 contains an Avalon interconnect pipeline stage that bridges the PCIe I/O and DDR4 I/O regions. The Avalon interconnect pipeline stages also help improve the timing closure rate of the static board interface part of the design.

You must create a dedicated LogicLock Plus region for the OpenCL kernel system. Furthermore, do not place kernel logic in the board’s LogicLock Plus regions (that is, static region). The static region and the OpenCL kernel system region (that is, PR region) do not overlap each other. As shown in Figure 2 on page 37, the logic for the boardtest.cl OpenCL kernel, that is, the scatter area, can be placed anywhere except within the seven LogicLock Plus regions.

Intel recommends the following strategies to maximize the available FPGA resources for the OpenCL kernel system to improve kernel routability:

• The OpenCL kernel system PR region should cover the entire device except the LogicLock Plus regions of the board.

• The size of a LogicLock Plus region should be just large enough to contain the board logic and to meet timing constraints of the board clocks. Oversized LogicLock Plus regions consume FPGA resources unnecessarily.

• Avoid creating tightly-packed LogicLock Plus regions that cause very high logic utilization and high routing congestion.

High routing congestion within the LogicLock Plus regions might decrease the Fitter's ability to route OpenCL kernel signals through the regions.

In the case where the board clocks are not meeting timing and the critical path is between the LogicLock Plus regions (that is, across region-to-region gap), insert back-to-back pipeline stages on paths that cross the gap. For example, if the critical path is between Region 1 and Region 2, lock down the first pipeline stage (an Avalon-MM Pipeline Bridge component) to Region 1, lock down the second pipeline stage to Region 2, and connect the two pipeline stages directly. This technique ensures that pipeline registers are on both sides of the region-to-region gap, thereby minimizing the delay of paths crossing the gap.

Refer to the Pipelining section for more information.

Related Links
Pipelining on page 39

You must manually insert pipelines throughout the FPGA system.

3.4.4 Global Routing

FPGAs have dedicated clock trees that distribute high fan-out signals to various sections of the devices. In the FPGA system that the Arria 10 FPGA Development Kit Reference Platform targets, global routing can distribute high fan-out signals regionally or globally.

• Regional distribution applies across any quadrant of the device.

• Global distribution applies across the entire device.

There is no restriction on the placement location of the OpenCL kernel on the device. As a result, the kernel clocks and kernel reset must distribute high fan-out signals globally.
To support PR, global routing for the Kernel Reset signal that drives logic inside a PR region requires special handling. Refer to the *Partial Reconfiguration* section for more information.

**Related Links**

*Partial Reconfiguration* on page 31

The Arria 10 GX FPGA Development Kit Reference Platform uses PR as a default mechanism to reconfigure the OpenCL kernel-related partition of the design without altering the static board interface that is in a running state.

### 3.4.5 Pipelining

You must manually insert pipelines throughout the FPGA system.

In Qsys Pro, you can implement pipelines via an Avalon-MM Pipeline Bridge component by setting the following pipelining parameters within the *Avalon-MM Pipeline Bridge* dialog box:

- Select **Pipeline command signals**
- Select **Pipeline response signals**
- Select both **Pipeline command signals** and **Pipeline response signals**

**Examples of Pipeline Implementation**

- Signals that traverse long distances because of the floorplan's shape or the region-to-region gaps require additional pipelines.

  The DMA at the bottom of the FPGA must connect to the DDR4 memory at the top of the FPGA. To achieve timing closure of the board interface logic at a DDR4 clock speed of 300 MHz, additional pipeline stages between the OpenCL Memory Bank Divider component and the DDR4 controller IP are necessary. In the Arria 10 GX FPGA Development Kit Reference Platform's *board.qsys* Qsys Pro system, the pipeline stages are named `pipe_stage_ddr4a_dimm_*`. The middle pipeline stage, `pipe_stage_ddr4a_dimm`, combines both the direct kernel DDR4 accesses and the accesses through the OpenCL Memory Bank Divider. The multistage pipeline approach ensures that the kernel entry point to the pipeline is geared towards neither the OpenCL Memory Bank Divider, which is close to the PCIe IP core, nor the DDR4 IP core, which is at the very top of the FPGA.

### 3.4.6 DDR4 Calibration

The Arria 10 GX FPGA Development Kit Reference Platform includes special mechanisms to ensure the functional stability of the Arria 10 silicon. For example, the DDR4 memory might not calibrate successfully after FPGA reconfiguration. The driver within the a10_ref Reference Platform can detect a failed calibration via the Uniphy Status to AVS IP, and retrigger calibration through the `ddr4_calibrate` IP block.
3.4.7 Kernel Reprogramming via Partial Reconfiguration

The Arria 10 GX FPGA Development Kit Reference Platform provides the ability to modify the OpenCL kernel and reprograms it onto the FPGA. The a10_ref Reference Platform places the OpenCL kernel in a PR region of the device. Doing so allows you to reprogram the kernel-specific portion of the FPGA across the PCIe bus without affecting the board interface region (that is, static region) of the device.

**Dependencies**

Quartus Prime Pro Edition software's Partial Reconfiguration feature

To ensure that the device functions properly during and after PR reprogramming, following these rules:

- Place a freeze wrapper around the PR region. The freeze wrapper holds the critical control outputs from the PR region in a known, inactive state during the reprogramming of the logic inside the PR region.

  The \texttt{ALTERAOCLSDKROOT}/board/a10_ref/hardware/a10gx/ip/freezewrapper.v file implements the freeze wrapper, where \texttt{ALTERAOCLSDKROOT} is the path to the SDK installation.

- Hold the \texttt{kernel_reset_n} signal, which is routed using Global Clock resources, in a logic 1 (deasserted) state during reprogramming of the PR region. When programming completes, assert the \texttt{kernel_reset_n} signal (that is, set it to the low state) before disabling the freeze wrapper. Asserting the \texttt{kernel_reset_n} signal resets all logic in the PR region to a known state. This assertion step is necessary because the state of all flipflops in the PR region is undefined after PR programming. The logic in the \texttt{freezewrapper.v} file implements the required behavior for the reset and freeze signals.

3.5 Dynamic PLL Reconfiguration

PLL that is used to generate the OpenCL kernel clocks resides in the static region of the design's floorplan. As a result, reprogramming of the kernel partition via PR does not modify the PLL settings. The Intel FPGA SDK for OpenCL relies on the \texttt{post_flow_pr.tcl} Tcl script and the instantiation of the \texttt{acl_kernel_clk_a10} Qsys Pro component to modify kernel PLL.

In both PR reprogramming and full-chip JTAG programming, the PLL is dynamically reconfigured by default after FPGA configuration completes. This default dynamic PLL reconfiguration step is unnecessary after full-chip programming because the correct PLL settings are already part of the .sof file programmed onto the FPGA over JTAG.

3.6 Guaranteed Timing Closure of the Arria 10 GX FPGA Development Kit Reference Platform Design

One of the key features of the Intel FPGA SDK for OpenCL is that it abstracts away hardware details, such as timing closure, for software developers. Both the SDK and the Custom Platform contribute to the implementation of the SDK's guaranteed timing closure feature.
The SDK provides the IP to generate the kernel clock, and a post-flow script that ensures this clock is configured with a safe operating frequency confirmed by timing analysis. The Custom Platform developer imports a post-fit netlist that has already achieved timing closure on all non-kernel clocks.

### 3.6.1 Supply the Kernel Clock

In the Arria 10 GX FPGA Development Kit Reference Platform, the OpenCL Kernel Clock Generator component provides the kernel clock and its 2x variant.

The `REF_CLK_RATE` parameter specifies the frequency of the reference clock that connects to the kernel PLL (`pll_refclk`). For the a10_ref Reference Platform, the `REF_CLK_RATE` frequency is 125 MHz.

The `KERNEL_TARGET_CLOCK_RATE` parameter specifies the frequency that the Quartus Prime Pro Edition software attempts to achieve during compilation. The board hardware contains some logic that the kernel clock clocks. At a minimum, the board hardware includes the clock crossing hardware. To prevent this logic from limiting the Fmax achievable by a kernel, the `KERNEL_TARGET_CLOCK_RATE` must be higher than the frequency that a simple kernel can achieve on your device. For the Arria 10 GX FPGA Development Kit that the a10_ref Reference Platform targets, the `KERNEL_TARGET_CLOCK_RATE` is 400 MHz.

**Caution:** When developing a Custom Platform, setting a high target Fmax might cause difficulty in achieving timing closure.

When developing your Custom Platform and attempting to close timing, add an overriding SDC definition to relax the timing of the kernel. The following code example from the `ALTERAOCLSDKROOT/board/a10_ref/hardware/a10gx/top_post.sdc` file applies a 5 ns (200 MHz) maximum delay constraint on the OpenCL kernel during base revision compilations:

```plaintext
if {! [string equal $::TimeQuestInfo(nameofexecutable) "quartus_map"]}
{
  if { [get_current_revision] eq "base" }
  {
    post_message -type critical_warning "Compiling with slowed OpenCL Kernel clock."
    This is to help achieve timing closure for board bringup."
    if {! [string equal $::TimeQuestInfo(nameofexecutable) "quartus_sta"]}
    {
      set kernel_keepers [get_keepers system_inst\kernel_system\*]
      set_max_delay 5 -from $kernel_keepers -to $kernel_keepers
    }
  }
}
```

### 3.6.2 Guarantee Kernel Clock Timing

The Quartus Prime database interface executable (`quartus_cdb`) runs a script after every Quartus Prime Pro Edition software compilation as a post-flow script. In the Arria 10 GX FPGA Development Kit Reference Platform, the OpenCL Kernel Clock Generator component works together with the post-flow script to guarantee kernel clock timing.
In the import revision compilation, the compilation script `import_compile.tcl` invokes the `ALTERAOCLSDKROOT/board/a10_ref/hardware/a10gx/scripts/post_flow.tcl` Tcl script in the `a10_ref` Reference Platform after every Quartus Prime Pro Edition software compilation using `quartus_cdb`.

The `post_flow.tcl` script also determines the kernel clock and configures it to a functional frequency.

*Important:* Execute this post flow script for every Quartus Prime compilation.

### 3.6.3 Provide a Timing-Closed Post-Fit Netlist

Each Intel FPGA SDK for OpenCL-compatible Reference and Custom Platform, such as the Arria 10 GX FPGA Development Kit Reference Platform, provides a timing-closed post-fit netlist that imports placement and routing information for all nodes clocked by non-kernel clocks.

**Dependencies**

Quartus Prime Pro Edition software's Spectra-Q engine

Intel provides several mechanisms for preserving the placement and routing of some previously compiled logic and importing this logic into a new compilation. For Arria 10 devices, the previously compiled logic is imported into the Spectra-Q engine compilation flow.
**Figure 3. Custom Platform Development Flow and Hand-Off between Board Developer and End User**

This figure illustrates the hand-off between the board vendor and the SDK end user. The board developer is responsible for porting the a10_ref Reference Platform to their own board, closing timing, and locking down the static part of the board.

![Custom Platform Development Flow Diagram](image)

**Figure 4. Structure of the Hierarchy for the OpenCL Hardware System on the Arria 10 Device**

This figure illustrates that the placement and routing for everything outside the kernel_system partition are preserved and are imported in the top revision compilations. The kernel_system partition itself is not preserved and is compiled from source.

```
root_partition (top.v)
```

Board Interface (board.qsys)

Freeze Wrapper (freeze_wrapper.v)

OpenCL Kernel (kernel_system.qsys)

The Spectra-Q engine compilation flow for Arria 10 preserves the placement and routing of the board interface partition via the exported Quartus Prime Archive File. The base.qdb file contains all the database files for the base compilation of root_partition. The a10_ref Reference Platform is configured with the project
revisions and partitioning that are necessary to implement the Spectra-Q engine compilation flow. By default, the SDK invokes the Quartus Prime Pro Edition software on the top revision. This revision is configured to import and restore the base.qdb file, which has been precompiled and exported from a base revision compilation.

When developing your Custom Platform from the a10_ref Reference Platform, it is essential to maintain the flat.qsf, base.qsf, top.qsf, and top_synth.qsf Quartus Prime Settings Files.

The a10_ref Reference Platform includes two additional partitions: the Top partition and the kernel_system partition. The Top partition contains all logic, and the kernel_system partition contains the logic in the PR region. The PR region is specified by the following assignments:

```
set_instance_assignment -name PARTIAL_RECONFIGURATION_PARTITION ON -to freeze_wrapper_inst|kernel_system_inst
```

### Related Links

- Generating the base.qdb Post-Fit Netlist for Your Arria 10 Custom Platform on page 19

To implement the Spectra-Q engine compilation flow, you must generate a base.qdb Quartus Prime Archive File for your Arria 10 Custom Platform.

### 3.7 Quartus Prime Compilation Flow and Scripts

The `import_compile.tcl` Tcl Script File in the Arria 10 GX FPGA Development Kit Reference Platform controls the Quartus Prime compilation flow.

Invoke the Quartus Prime compilation flow by calling the following `quartus_sh` executables:

- The board developer runs the `quartus_sh --flow compile top -c base` command to execute the base revision compilation. This compilation closes timing, locks down the static region, and generates the base.qdb file.

- The user of the Arria 10 FPGA Development Kit Reference Platform or a Custom Platform runs the `quartus_sh -t import_compile.tcl` command to execute the import revision compilation. This compilation generates programming files that are guaranteed to be timing closed and PR-compatible with each other.

### 3.7.1 Enabling the Quartus Prime Spectra-Q Forward-Compatibility Flow

The Quartus Prime Spectra-Q forward-compatibility flow allows you to use base.qdb files that are forward compatible with future versions of the Quartus Prime Pro Edition software.

Enabling the forward-compatibility flow allows you to use board vendor-generated precompiled post-fit netlists, in the form of the base.qdb file, in a future Quartus Prime Pro Edition software version. The forward-compatibility flow eliminates the need to match the Quartus Prime Pro Edition software version used to develop the Custom Platform and the version used to run the Custom Platform.
Warning: Intel does not guarantee that the compilation of your board design in a future version of the Quartus Prime Pro Edition software will be successful. It is possible that your base.qdb file implements a configuration that will become illegal in future Quartus Prime Pro Edition software versions.

If you are migrating a previous version of the Arria 10 GX FPGA Development Kit Reference Platform to the current version and you want to incorporate the forward-compatibility flow, perform the following tasks:

1. Add the following command in the ALTERAOCLSDKROOT/board/a10_ref/hardware/a10gx/scripts/post_flow_pr.tcl script to generate a forward-compatible base.qdb file:

   `quartus_cdb top -c base --export_design --snapshot final --file base.qdb`

   For information on the function of the post_flow_pr.tcl script, refer to Quartus Prime Compilation Flow for Board Developers.

2. In the ALTERAOCLSDKROOT/board/a10_ref/hardware/a10gx/import_compile.tcl script, add the quartus_fit and then the quartus_asm commands after importing the base.qdb file.

   Running these commands verifies that the imported base.qdb file is usable in the Quartus Prime Pro Edition software version that Custom Platform users work with.

   For more information on the function of the import_compile.tcl script, refer to the Quartus Prime Compilation Flow for Custom Platform Users.

Related Links
- Quartus Prime Compilation Flow for Board Developers on page 45
  The quartus_sh --flow compile top -c base command executes the Quartus Prime compilation flow that generates a base.sof full-chip JTAG programming file within the .aocx file.
- Quartus Prime Compilation Flow for Custom Platform Users on page 46
  The import_compile.tcl script executes the Quartus Prime compilation flow that generates a top.sof full-chip JTAG programming file and a top.rbf PR bitstream file within the .aocx file.

3.7.2 Quartus Prime Compilation Flow for Board Developers

The quartus_sh --flow compile top -c base command executes the Quartus Prime compilation flow that generates a base.sof full-chip JTAG programming file within the .aocx file. The script performs the necessary tasks to ensure that the import revision compilations using the timing-closed and locked-down static region are PR-compatible with each other.

Running the quartus_sh --flow compile top -c base command executes the following tasks:
• Runs `quartus_syn` to execute the Analysis and Synthesis stage of the Quartus Prime compilation flow.
• Runs `quartus_fit` to execute the Place and Route stage of the Quartus Prime compilation flow.
• Runs `quartus_sta` to execute the Static Timing Analysis stage of the Quartus Prime compilation flow.
• Runs the `ALTERAOCLSDKROOT/board/a10_ref/hardware/a10gx/scripts/post_flow_pr.tcl` file.

The `post_flow_pr.tcl` script determines the maximum frequency at which the OpenCL kernel can run and generates the corresponding PLL settings. The script then reruns static timing analysis. The script also exports the Spectra-Q engine database of the base revision compilation to the forward-compatible `.qdb` Quartus Prime Database Export File. Refer to the QDB File Generation section for more information.

• Runs `quartus_asm` to generate the `.sof` file with updated embedded PLL settings. Updating the `.sof` file allows it to run safely on the board with the maximum kernel frequency.
• Generates the `.fpga.bin` file, which contains the full-chip programming file. The full-chip programming file (`.base.sof`) is in the `.acl.sof` section of the `.fpga.bin` file.

The `.aocx` file that the base revision compilation flow generates only contains the `.sof` full-chip programming file. It does not contain a programming file that can be used with PR because this `.aocx` file is only intended to be written to Flash memory as the default FPGA image. The Intel FPGA SDK for OpenCL program utility automatically uses JTAG programming when it programs with a `.aocx` file from the base revision compilation. Only the import revision compilation flow, executed by the SDK user, generates a `.aocx` file that can be used with PR.

**Related Links**

• Hash Checking on page 48
  Intel assigns a unique ID to each base revision compilation to ensure a safe way of only partially reconfiguring a PR region on top of a design that has a matching static region.

• Qsys Pro System Generation on page 48
  The Intel FPGA SDK for OpenCL Offline Compiler generates the `board.qsys` and `kernel_system.qsys` Qsys Pro systems in the `ALTERAOCLSDKROOT/board/<custom_platform>/hardware/<board_name>` directory after successfully completing a first-stage compilation.

• QDB File Generation on page 48
  The `base.qdb` Quartus Prime Archive File contains all the necessary Spectra-Q engine database files for importing a timing-closed and placed-and-routed netlist of the static region.

### 3.7.3 Quartus Prime Compilation Flow for Custom Platform Users

The `import_compile.tcl` script executes the Quartus Prime compilation flow that generates a `.top.sof` full-chip JTAG programming file and a `.top.rbf` PR bitstream file within the `.aocx` file.
The `import_compile.tcl` script executes the following tasks:

- Runs the `ALTERAOCLSDKROOT/board/a10_ref/hardware/a10gx/scripts/pre_flow_pr.tcl` file. The `pre_flow_pr.tcl` script generates the `board.qsys` and the `kernel_system.qsys` Qsys Pro System Files. Refer to the Qsys Pro System Generation section for more information.
- Imports the Spectra-Q engine database of the base revision compilation to the `.qdb` file. Refer to the QDB File Generation section for more information.
- Runs `quartus_fit` and `quartus_asm` to verify that the `.qdb` file is forward compatible.
- Runs `quartus_syn` to execute the Analysis and Synthesis stage of the Quartus Prime compilation flow for the kernel partition only.
- Runs `quartus_fit` to execute the Place and Route stage of the Quartus Prime compilation flow for the entire design.
- Runs `quartus_sta` to execute the Static Timing Analysis stage of the Quartus Prime compilation flow.
- Runs the `ALTERAOCLSDKROOT/board/a10_ref/hardware/a10gx/scripts/post_flow_pr.tcl` file. The `post_flow_pr.tcl` script determines the maximum frequency at which the OpenCL kernel can run and generates the corresponding PLL settings. The script then reruns the static timing analysis.
- Runs `quartus_asm` to generate the full-chip programming files for the base revision.
- Runs `quartus_asm` to generate the full-chip programming files for the import revision.
- Runs `quartus_cpf` to generate the PR programming files.
- Generates the `fpga.bin` file, which contains the following files and IDs:
  - The `top.sof` full-chip programming file.
  - The `top.rbf` PR programming file.
  - The `pr_base_id.txt` unique ID for PR base revision.

Before `quartus_asm` generates the `.sof` file in an import revision compilation, the static region of the import revision compilation is compared to the static region of the base revision compilation to check for errors. To prevent a mismatch error in the I/O configuration shift register (IOCSR) bits, the PLL settings in the `base.sof` and `top.sof` files must be identical. When designing the Arria 10 FPGA Development Kit Reference Platform, Intel ensured in the `import_compile.tcl` Tcl script that the PLL settings in both the `base.sof` file and the `top.sof` file are identical, resulting in an additional `quartus_asm` execution step to regenerate the `base.sof` file.

Related Links
- Qsys Pro System Generation on page 48
  The Intel FPGA SDK for OpenCL Offline Compiler generates the `board.qsys` and `kernel_system.qsys` Qsys Pro systems in the `ALTERAOCLSDKROOT/board/<custom_platform>/hardware/<board_name>` directory after successfully completing a first-stage compilation.
3.7.4 Qsys Pro System Generation

The Intel FPGA SDK for OpenCL Offline Compiler generates the board.qsys and kernel_system.qsys Qsys Pro systems in the ALTERAOCLSDKROOT/board/<custom_platform>/hardware/<board_name> directory after successfully completing a first-stage compilation. The ALTERAOCLSDKROOT environment variable points to the location of the Intel FPGA SDK for OpenCL installation directory.

The board.qsys Qsys Pro system represents the bulk of the static region. The kernel_system.qsys Qsys Pro system is the top-level of the PR region. The pre_flow_pr.tcl script generates both Qsys Pro systems on the fly before the beginning of the Quartus Prime compilation flow in both the base and import revision compilations.

3.7.5 QDB File Generation

The base.qdb Quartus Prime Archive File contains all the necessary Spectra-Q engine database files for importing a timing-closed and placed-and-routed netlist of the static region.

The ALTERAOCLSDKROOT/board/a10_ref/hardware/a10gx/scripts/post_flow_pr.tcl script creates the base.qdb file. The .tcl file invokes the export_design command to export the entire base revision compilation database to the base.qdb file. The base.sdc and pr_base_id.txt files are not part of the base.qdb file. For your Custom Platform, you must add the base.sdc and pr_base_id.txt files to the board directory (that is, ALTERAOCLSDKROOT/board/<custom_platform>/hardware/<board_name>) separately.

3.7.6 Hash Checking

Intel assigns a unique ID to each base revision compilation to ensure a safe way of only partially reconfiguring a PR region on top of a design that has a matching static region.

The unique ID is generated at the beginning of a base revision compilation using the MD5 message-digest algorithm. The MD5 algorithm generates a hash of a text file that contains the current working directory and a high-resolution timer value. The MD5 algorithm then truncates the hash to a 32-bit value. The ALTERAOCLSDKROOT/board/a10_ref/hardware/a10gx/scripts/pre_flow_pr.tcl script stores this 32-bit value in the pr_base_id register IP within the board.qsys Qsys Pro system by overwriting the default value of 0xdeadbeef.

The unique ID for the base revision compilation is added to the pr_base_id.txt file. The ID becomes part of the import revision compilation directory after the pr_base_id.txt file is copied from the ALTERAOCLSDKROOT/board/a10_ref/hardware/a10gx directory. During the fpga.bin generation step of the import revision compilation, the unique ID is added as the .acl.hash section of the fpga.bin file.
When the Intel FPGA SDK for OpenCL user invokes the aocl program utility to reconfigure the FPGA, the software first checks that the pr_base_id value in the currently programmed static region matches the hash value in the fpga.bin section within the .aocx file. If the two 32-bit values match, it is safe to execute partial reconfiguration. If the 32-bit values do not match, the aocl program utility performs full-chip JTAG programming via USB-Blaster.

### 3.8 Addition of Timing Constraints

A Custom Platform must apply the correct timing constraints to the Quartus Prime project. In the Arria 10 FPGA Development Kit Reference Platform, the top.sdc file contains all timing constraints applicable before IP instantiation in Qsys Pro. The top_post.sdc file contains timing constraints applicable after Qsys Pro.

The order of the application of time constraints is based on the order of appearance of the top.sdc and top_post.sdc in the top.qsf file.

One noteworthy constraint in the a10_ref Reference Platform is the multicycle constraint for the kernel reset in the top_post.sdc file. Using global routing saves routing resources and provides more balanced skew. However, the delay across the global route might cause recovery timing issues that limit kernel clock speed. Therefore, it is necessary to include a multicycle path on the global reset signal.

### 3.9 Connection of the Arria 10 GX FPGA Development Kit Reference Platform to the Intel FPGA SDK for OpenCL

A Custom Platform must include a board_env.xml file to describe its general contents to the Intel FPGA SDK for OpenCL Offline Compiler. For each hardware design, your Custom Platform also requires a board_spec.xml file for each hardware design that describes the hardware.

The following sections describe the implementation of these files for the Arria 10 GX FPGA Development Kit Reference Platform.

#### 3.9.1 Describe the Arria 10 GX FPGA Development Kit Reference Platform to the Intel FPGA SDK for OpenCL

The ALTERAOCLSDKROOT/board/a10_ref/board_env.xml file describes the Arria 10 GX FPGA Development Kit Reference Platform to the Intel FPGA SDK for OpenCL. Details of each field in the board_env.xml file are available in the Creating the board_env.xml File section of the Intel FPGA SDK for OpenCL Custom Platform Toolkit User Guide.

In the a10_ref Reference Platform, Intel uses the bin folder for Windows dynamic link libraries (DLLs), the lib directory for delivering libraries, and the libexec directory for delivering the SDK utility executables. This directory structure allows the PATH environment variable to point to the location of the DLLs (that is, bin) in isolation of the SDK utility executables.

**Related Links**

Creating the board_env.xml File
3.9.2 Describe the Arria 10 GX FPGA Development Kit Reference Platform Hardware to the Intel FPGA SDK for OpenCL

The Arria 10 GX FPGA Development Kit Reference Platform includes an
ALTERAOCLSDKROOT/board/a10_ref/hardware/a10gx/board_spec.xml file
that describes the hardware to the Intel FPGA SDK for OpenCL.

Device

The device section contains the name of the device model file available in the
ALTERAOCLSDKROOT/share/models/dm directory of the SDK and in the board
spec.xml file. The used_resources element accounts for all logic outside of
the kernel partition. The value of used_resources for alms equals the difference
between the total number of adaptive logic modules (ALMs) used in final placement
and the total number of ALMs available to the kernel partition. You can derive this
value from the Partition Statistic section of the Fitter report after a compilation.
Consider the following ALM categories within an example Fitter report:

```
+---------------------------------------------------------------------------------
| ; Fitter Partition Statistics |
+-------------------------------+-------------------------------+
| ; Statistic ; l ; freeze_wrapper_inst| kernel_system_inst |
+-------------------------------+-------------------------------+
| ; ALMs needed [-A-B+C] ; 0 / 427200 (0%) ; 0 / 385220 (0%) |
|-------------------------------+-------------------------------+
```

The value of used_resources equals the total number of ALMs in l minus the total
number of ALMs in freeze wrapper inst | kernel_system_inst. In the example
above, used_resources = 427200 - 385220 = 41980 ALMs.

You can derive used_resources for rams and dsps in the same way using M20Ks
and DSP blocks, respectively. The used_resources value for ffs is four times the
used_resources value for alms because there are two primary and two secondary
logic registers per ALM.

Global Memory

In the board_spec.xml file, there is one global_mem section for DDR memory.
Assign the string DDR to the name attribute of the global_mem element. The board
instance in Qsys Pro provides all of these interfaces. Therefore, the string board is
specified in the name attribute of all the interface elements within global_mem.

- DDR

Because DDR memory serves as the default memory for the board that the
a10_ref Reference Platform targets, its address attribute begins at zero. Its
cfg_addr is 0x018 to match the memorg conduit used to connect to the
corresponding OpenCL Memory Bank Divider for DDR.

**Attention:** The width and burst sizes must match the parameters in the OpenCL
Memory Bank Divider for DDR (memory_bank_divider).
Interfaces

The interfaces section describes kernel clocks, reset, CRA, and snoop interfaces. The OpenCL Memory Bank Divider for the default memory (in this case, memory_bank_divider) exports the snoop interface described in the interfaces section. The width of the snoop interface should match the width of the corresponding streaming interface.

3.10 Arria 10 FPGA Programming Flow

There are three ways to program the Arria 10 FPGA for the Arria 10 GX FPGA Development Kit Reference Platform: Flash, quartus_pgm, and Partial Reconfiguration.

In the order from the longest to the shortest configuration time, the three FPGA programming methods are as follows:

- To replace both the FPGA periphery and the core while maintaining the programmed state after power cycling, use Flash programming.
- To replace both the FPGA periphery and the core, use the Quartus Prime Programmer command-line executable (quartus_pgm) to program the device via cables such as USB-Blaster.
- To replace only the kernel portion of the device, use PR.

The default FPGA programming flow is to use PR over PCIe. The Partial Reconfiguration Controller IP instantiates PR over PCIe using the following IP parameter settings:

Table 10. Parameter Settings for the Partial Reconfiguration Controller IP

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use as PR Internal Host</td>
<td>Enabled</td>
</tr>
<tr>
<td>Enable Avalon-MM slave interface</td>
<td>Enabled</td>
</tr>
<tr>
<td>Input data width</td>
<td>32 bits</td>
</tr>
<tr>
<td>Clock-to-Data ratio</td>
<td>1</td>
</tr>
<tr>
<td>Divide error detection frequency by</td>
<td>1</td>
</tr>
<tr>
<td>Auto-instantiate PR block</td>
<td>Enabled</td>
</tr>
<tr>
<td>Auto-instantiate CRC block</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

The 50 MHz config_clk clocks the Partial Reconfiguration Controller IP. The Avalon-MM interface connects to the host control bus on PCIe BAR4. Using PCIe Gen3x8 under these configuration settings, the duration of partial reconfiguration of the PR region is about 1.6 seconds.

You cannot use PR if there is a mismatch between the hash within the .aocx file and the hash in the static region of the current image on the FPGA. In this case, program the FPGA via USB-Blaster by invoking quartus_pgm instead. If the .aocx file is not PR compatible with the current image on the FPGA, the Quartus Prime Programmer displays the following message:
3 Arria 10 GX FPGA Development Kit Reference Platform Design Architecture

aocl program acl0 boardtest.aocx
aocl program: Running program from <path_to_a10_ref>/linux64/libexec
Reprogramming device with handle 1
MMD INFO : [acla10_ref0] PR base and import compile IDs do not match
MMD INFO : [acla10_ref0] PR base ID currently configured is 0x7d056bf2
MMD INFO : [acla10_ref0] PR import compile expects ID to be 0x30242eb9
mmd program_device: Board reprogram failed

Only use `quartus_pgm` via USB-Blaster if you use a cable to connect the board and the host computer. Cabling is a point of potential failure, and it does not scale well to large deployments. If possible, reserve the `quartus_pgm` programming approach for development and testing purposes only.

### 3.10.1 Define the Contents of the `fpga.bin` File for the Arria 10 GX FPGA Development Kit Reference Platform

You may arbitrarily define the contents of the `fpga.bin` file in a Custom Platform because it passes from the Intel FPGA SDK for OpenCL to the Custom Platform as a black box. Intel defines the contents of the `fpga.bin` file in the Arria 10 GX FPGA Development Kit Reference Platform as an Executable and Linkable Format (ELF) binary that organizes the various fields into sections.

Table 11. Contents of the Arria 10 GX FPGA Development Reference Platform’s `fpga.bin` File

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.acl.sof</td>
<td>The full programming bitstream for the compiled design. This section appears in the <code>fpga.bin</code> files generated from both the base revision and the import revision compilations.</td>
</tr>
<tr>
<td>.acl.core.rbf</td>
<td>The PR programming bitstream for the kernel region. This section only appears in the <code>fpga.bin</code> file generated from import revision compilation.</td>
</tr>
<tr>
<td>.acl.hash</td>
<td>The unique ID for the base revision compilation. This section only appears in the <code>fpga.bin</code> file generated from import revision compilation.</td>
</tr>
</tbody>
</table>

### 3.11 Host-to-Device MMD Software Implementation

The Arria 10 GX FPGA Development Kit Reference Platform’s MMD layer is a thin software layer that is essential for communication between the host and the board. A full implementation of the MMD library is necessary for every Custom Platform for the proper functioning of the OpenCL host applications and board utilities. Details of the API functions, their arguments, and return values for MMD layer are specified in the `<your_custom_platform>/source/include/aocl_mmd.h` file, where `<your_custom_platform>` points to the top-level directory of your Custom Platform.

The source codes of an MMD library that demonstrates good performance are available in the `ALTERAOCLSDKROOT/board/a10_ref/source/host/mmd` directory. Refer to the Host-to-Device MMD Software Implementation section in the Stratix V Network Reference Platform Porting Guide for more information.

For more information on the MMD API functions, refer to the MMD API Descriptions section of the Intel FPGA SDK for OpenCL Custom Platform Toolkit User Guide.

**Related Links**
- Host-to-Device MMD Software Implementation
3.12 Implementation of Intel FPGA SDK for OpenCL Utilities

The Arria 10 GX FPGA Development Kit Reference Platform includes a set of Intel FPGA SDK for OpenCL utilities for managing the FPGA board.

For more information on the implementation requirements of the AOCL utilities, refer to the Providing Intel FPGA SDK for OpenCL Utilities Support section of the Intel FPGA SDK for OpenCL Custom Platform Toolkit User Guide.

Related Links
Providing Intel FPGA SDK for OpenCL Utilities Support

3.12.1 aocl install

The install utility in the Arria 10 GX FPGA Development Kit Reference Platform installs the kernel driver on the host computer. Users of the Intel FPGA SDK for OpenCL only need to install the driver once, after which the driver should be automatically loaded each time the machine reboots.

Windows

The install.bat script is located in the \<your_custom_platform>\windows64\libexec directory, where <your_custom_platform> points to the top-level directory of your Custom Platform. This install.bat script triggers the install executable from Jungo Connectivity Ltd. to install the WinDriver on the host machine.

Linux

The install script is located in the <your_custom_platform>/linux64/libexec directory. This install script first compiles the kernel module in a temporary location and then performs the necessary setup to enable automatic driver loading after reboot.

3.12.2 aocl uninstall

The uninstall utility in the Arria 10 GX FPGA Development Kit Reference Platform removes the current host computer drivers used for communicating with the board.

Windows

The uninstall.bat script is located in the \<your_custom_platform>\windows64\libexec directory, where <your_custom_platform> points to the top-level directory of your Custom Platform. This uninstall.bat script triggers the uninstall executable from Jungo Connectivity Ltd. to uninstall the WinDriver on the host machine.

Linux

The uninstall script is located in the <your_custom_platform>/linux64/libexec directory. This uninstall script removes the driver module from the kernel.
3.12.3 aocl program

The program utility in the Arria 10 GX FPGA Development Kit Reference Platform programs the board with the specified .aocx file. Calling the aoc1_mmd_reprogram() MMD API function implements the program utility.

3.12.4 aocl flash

The flash utility in the Arria 10 GX FPGA Development Kit Reference Platform configures the power-on image for the FPGA using the specified .aocx file. Calling into the MMD library implements the flash utility.

Figure 5. JTAG Chain with Arria 10 FPGA, MAX V CPLD, and CFI Flash Memory

This figure illustrates the JTAG chain and the location of the common flash interface (CFI) relative to the MAX V CPLD on the Arria 10 GX FPGA Development Kit.

3.12.5 aocl diagnose

The diagnose utility in the Arria 10 GX FPGA Development Kit Reference Platform reports device information and identifies issues. The diagnose utility first verifies the installation of the kernel driver. Depending on whether an additional argument is specified in the command, the utility then performs different tasks.

Without an argument, the utility returns the overall information of all the devices installed in a host machine. If a specific device name is provided as an argument (that is, aocl diagnose <device_name>), the diagnose utility runs a memory transfer test and then reports the host-device transfer performance.
3.13 Arria 10 FPGA Development Kit Reference Platform Scripts

The Arria 10 FPGA Development Kit Reference Platform includes a number of Tcl scripts in its `hardware/<board_name>/scripts` directory.

Table 12. Tcl Scripts within the ALTERAOCLSDKROOT/board/a10_ref/hardware/a10gx/scripts Directory

<table>
<thead>
<tr>
<th>Script</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_write_sdc.tcl</td>
<td>The <code>post_flow_pr.tcl</code> script runs this script during the base revision compilation. The <code>base_write_sdc.tcl</code> script then exports all the SDC constraints to the <code>base.sdc</code> file, which is part of the board directory.</td>
</tr>
<tr>
<td>create_fpga_bin_pr.tcl</td>
<td>Creates the ELF binary file, <code>fpga.bin</code>, from the <code>.sof</code> file, the <code>.rbf</code> file, and the <code>pr_base_id.txt</code> file.</td>
</tr>
<tr>
<td>post_flow_pr.tcl</td>
<td>This script runs after every Quartus Prime Pro Edition software compilation. It facilitates the guaranteed timing flow by setting the kernel clock PLL, generating a small report in the <code>acl_quartus_report.txt</code> file, and rerunning STA with the modified kernel clock settings.</td>
</tr>
<tr>
<td>pre_flow_pr.tcl</td>
<td>This script generates the RTL of the top-level <code>board.qsys</code> Qsys Pro system for the static region and the <code>kernel_system.qsys</code> Qsys Pro system for the kernel PR region.</td>
</tr>
</tbody>
</table>

3.14 Considerations in Arria 10 GX FPGA Development Kit Reference Platform Implementation

The implementation of the Arria 10 GX FPGA Development Kit Reference Platform includes some workarounds that address certain Quartus Prime Pro Edition software known issues.

- The `quartus_syn` executable reads the SDC files. However, it does not support the Tcl command `get_current_revision`. Therefore, in the `top_post.sdc` file, a check is in place to determine whether `quartus_syn` has read the file before checking the current version.

In addition to these workarounds, take into account the following considerations:

- Quartus Prime compilation is only ever performed after the Intel FPGA SDK for OpenCL Offline Compiler embeds an OpenCL kernel inside the system.
- Perform Quartus Prime compilation after you install the Intel FPGA SDK for OpenCL and set the `ALTERAOCLSDKROOT` environment variable to point to the SDK installation.
- The name of the directory where the Quartus Prime project resides must match the `name` field in the `board_spec.xml` file within the Custom Platform. The name must be case sensitive.
- The `PATH` or `LD_LIBRARY_PATH` environment variable must point to the MMD library in the Custom Platform.
### Table 13. Document Revision History of the Arria 10 GX FPGA Development Kit Reference Platform Porting Guide

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>October 2016</td>
<td>2016.10.31</td>
<td>• Rebranded Altera SDK for OpenCL to Intel FPGA SDK for OpenCL.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Rebranded Altera Offline Compiler to Intel FPGA SDK for OpenCL Offline Compiler.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Changed the short-form name of the Reference Platform from altera_a10pciedk to a10_ref, to match the directory name in the SDK.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added notice that you must contact your field applications engineer or regional support center representative to configure the Arria 10 GX FPGA Development Kit before using it with the SDK.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Removed the a10gx_es2 and the a10gx_es3 board variants from the Reference Platform. The a10_ref Reference Platform only supports the a10gx board variant.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• In Contents of the Arria 10 GX FPGA Development Kit Reference Platform:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>— For Windows, changed the source_windows64 directory to source.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>— Updated the list of files available in the a10gx subdirectory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>— Removed information for the max5_133.pof file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Removed statement regarding PR being an early-access feature.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated the location of the acl_ddr4_a10.qsys and acl_ddr4_a10_core.qsys files from the a10gx/ip directory to the top-level a10gx directory. The board.qsys, acl_ddr4_a10.qsys, and acl_ddr4_a10_core.qsys systems were migrated to Qsys Pro.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• In the ip subdirectory, added .ip files that contain parameters of instantiated external OpenCL IP. Refer to Contents of the Arria 10 GX FPGA Development Kit Reference Platform for more information.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added an openc1_bsp_ip.qsf file so that qsys_archive in Qsys Pro can insert .qsys and .ip files into this revision. All Verilog and Qsys source files from top.sdc and top_post.sdc are now in openc1_bsp_ip.qsf.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• In Modifying the Arria 10 GX FPGA Development Kit Reference Platform Design, added a step to update the device.tcl file with the correct settings.</td>
</tr>
</tbody>
</table>

*continued...*
### Changes

In **Changing the Device Part Number**:
- Noted that the QSF setting for the device part number is now in `device.tcl` instead of `flat.qsf`. The following device-specific assignments are now in `device.tcl`:
  - `FAMILY`, `MIN_CORE_JUNCTION_TEMP`, `MAX_CORE_JUNCTION_TEMP`, `DEVICE_FILTER_PACKAGE`, `DEVICE_FILTER_PIN_COUNT`, `ERROR_CHECK_FREQUENCY_DIVISOR`, `STRATIX_DEVICE_IO_STANDARD`, `RESERVE_ALL_UNUSED_PINS_WEAK_PULLUP`, `RESERVE_DATA0_AFTER_CONFIGURATION`
- Noted that the device part number must be updated in `acl_ddr4_a10.qsys` and `acl_ddr4_a10_core.qsys`, in addition to `board.qsys`.

In **Guaranteeing Timing Closure in the Arria 10 Custom Platform and Generating the base.qdb Post-Fit Netlist for Your Arria 10 Custom Platform**,
- noted that `base.sdc` must be copied along with `base.qdb` and `pr_base_id.txt` into the Custom Platform.

In **Floorplan**, updated the floorplan of the `a10_ref` Reference Platform.

In **Provide a Timing-Closed Post-Fit Netlist**, removed the QSF assignments that enabled the Spectra-Q engine compilation flow for base and top revision compilations. The `base.qsf` file no longer needs to be updated in order to enable the flow.

In **Enabling the Quartus Prime Spectra-Q Forward Compatibility Flow**:
- Modified the Quartus Prime software command to be added to the `post_flow_pr.tcl` script to generate the forward-compatible `base.qdb` file.
- Removed the step of modifying the `quartus.ini` file because it is no longer needed.

In **Quartus Prime Compilation Flow for Board Developers**, modified the list of tasks that are performed when the `quartus_sh --flow compile top -c base` command was invoked because the process would no longer run the `pre_flow_pr.tcl` script.
- In the `top.qpf` file, reorganized the order of the revisions to `opencl_bsp_ip`, `flat`, `base`, `top_synth`, and then `top`. In addition, removed old references to Quartus Prime software version 15.1.

---

**continued...**
• Modified top_post.sdc file to reflect Qsys Pro RTL hierarchy changes

• To facilitate Partial Reconfiguration:
  — Added set_global_assignment -name REVISION_TYPE PR_BASE to the base.qsf file
  — Added set_global_assignment -name REVISION_TYPE PR_BASE to the top_synth.qsf file
  — Added set_global_assignment -name REVISION_TYPE PR_IMPL to the top.qsf file
  — In the quartus.ini file, removed the following lines:
    • qhd_enable_pr_bak_export=on
    • pr_allow_lims_on_globals_user Guarantee Frozen High=on
    • apl_use_advanced_pcl=off
    • qhd_force_bak_export=on
    • hd_force_bak_import=on

• In the flat.qsf file:
  — Removed the wildcarded LREGION assignments for pipe_stage_dma* and pipe_stage_pcie_* and the commented GLOBAL_SIGNAL assignments.
  — Added the line PR_ALLOW_GLOBAL_LIMS ON -to freeze_wrapper_inst
  — Changed the GLOBAL_SIGNAL assignment to kernel clocks

July 2016 2016.07.29

• Maintenance release.
• In Arria 10 GX FPGA Development Kit Reference Platform Board Variants and Initializing Your Arria 10 Custom Platform, added reminder to match the board variant with the status of the Arria 10 device on your board.

May 2016 2016.05.09

• Modified content to reflect the creation of the base.qdb file in lieu of the base_qhd.qar file.
• Modified content to reflect the implementation of the flat.qsf file, which contains all the common QSF assignments shared among the base.qsf, top.qsf, and top_synth.qsf files. Use the flat revision for compilation flows that cannot use PR and do not require guaranteed timing. Because the flat revision is included in both the base and top revisions, use the flat revision to expand your design (for example, to attach extra DDR memory banks on your board).
• Modified content to reflect the updated functionality of the pre_flow_pr.tcl and post_flow_pr.tcl scripts.
• Updated the command you run to execute the base revision compilation
  from quartus_sh -t base_compile.tcl
  to quartus_sh --flow compile top -c base.
  This update enables you to compile the design from the Quartus Prime Pro Edition software GUI.
• Removed the ip/acl_kernel_clk_a10/acl_kernel_clk_a10.qsys and ip/acl_temperature_sensor_a10/<file_name> files from the Reference Platform because the acl_kernel_clk_a10 and acl_temperature_sensor_a10 IP are now part of the Altera SDK for OpenCL.
  Use the IPs from AOCL instead of duplicating them in your Custom Platform. A check is in place to verify that these IPs are not duplicated in your Custom Platform.

continued...
<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| December 2015 | 2015.12.21 | • The guaranteed timing flow is now part of AOCL. To avoid duplication, removed the following files from the Reference Platform:  
  — adjust_pll.tcl, which creates the PLL configuration file and modifies the PLL atoms  
  — pr_checks.tcl, which checks for initialized MLABs  
• Removed information on the following legacy files; they are no longer part of the Reference Platform:  
  — hardware/<board_name>/base_compile.tcl  
  — hardware/<board_name>/base_qhd.qar  
  — hardware/<board_name>/system.qsys  
  — scripts/call_script_as_function.tcl  
  — scripts/create_pr_base_id.tcl  
• Added memory hierarchy in board.qsys:  
  — The DDR4 subsystem is now in a separate IP located in the ip/ acl_ddr4_a10 directory  
  — The DDR4 core and pipeline stages are not in separate Qsys systems  
• In *Describe the Arria 10 GX FPGA Development Kit Reference Platform Hardware to the AOCL*, updated the example Fitter Partition Statistics report and the explanation on how to calculate used_resources for alms.  
• Under *Quartus Prime Compilation Flow and Scripts*, added the section *Enabling the Quartus Prime Spectra-Q Forward-Compatibility Flow*. Modified the import_compile.tcl file and added INI settings to quartus.ini and base.qsf to enable the Forward Compatibility flow. Support for the Forward Compatibility flow is preliminary. Refer to the Altera SDK for OpenCL version 16.0 Release Notes for more details.  

Initial release.