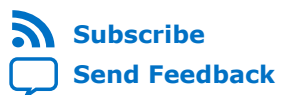# Intel® FPGA SDK for OpenCL™

Stratix® V Network Reference Platform Porting Guide

Updated for Intel® Quartus® Prime Design Suite: **17.1**

Contents

(intel®)

1 Intel® FPGA SDK for OpenCL™ Stratix® V Network Reference Platform Porting Guide

The *Intel® FPGA SDK for OpenCL™ Stratix® V Network Reference Platform Porting Guide* describes the procedures and design considerations you can implement to modify the Stratix V Network Reference Platform (s5_net) into your own Custom Platform for use with the Intel FPGA Software Development Kit (SDK) for OpenCL[1][2]. This document also contains reference information on the design decisions for s5_net, which makes use of features such as heterogeneous memory buffers and I/O channels to maximize hardware usage on a computing card designed for networking.

1.1 Stratix V Network Reference Platform: Prerequisites

The *Stratix V Network Reference Platform Porting Guide* assumes that you are an experienced FPGA designer who is familiar with Intel's FPGA design tools and concepts.

These design tools and concepts include:

- FPGA architecture, including clocking, global routing and I/Os
- High-speed design
- Timing analysis
- Intel Quartus® Prime software
- Platform Designer (Standard) design and Avalon® interfaces
- Tcl scripting
- Designing with Logic Lock regions
- PCI Express* (PCIe*)
- DDR3 external memory

1.1.1 Legacy Board Support

The Intel FPGA SDK for OpenCL attempts to automigrate your existing Custom Platform to the most recent Intel Quartus Prime Design Suite version.

---

[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.

Refer to the *Custom Platform Automigration for Forward Compatibility* section in the *Intel FPGA SDK for OpenCL Custom Platform Toolkit User Guide* for more information and instructions.

The Custom Platform Toolkit is available in the SDK's `board` directory (that is, *INTELFPGAOCLSDKROOT*`/board/custom_platform_toolkit`).

Caution: The Stratix V Network Reference Platform and the *Stratix V Network Reference Platform Porting Guide* are not compatible with Custom Platforms created prior to Altera SDK for OpenCL version 14.0.

Related Links

Custom Platform Automigration for Forward Compatibility

1.2 Features of the Stratix V Network 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.

Figure 1.   Hardware Features on a Hypothetical Stratix V Network Reference Platform Computing Card
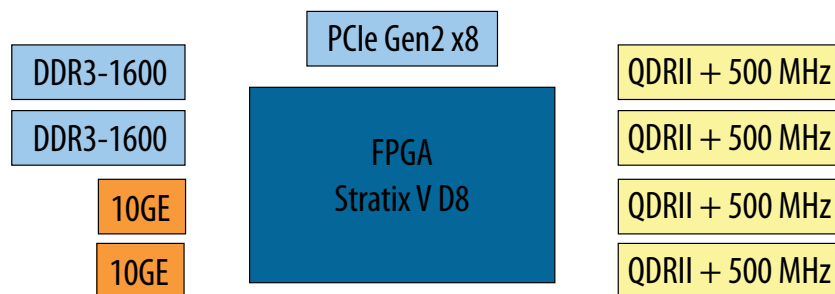
Features of s5_net:

1. OpenCL Host

    A PCIe-based host that connects to the Stratix V PCIe Gen2 x8 hard intellectual property (IP) core.

2. OpenCL Global Memory

    The hardware provides two separate 4-gigabyte (GB) DDR3 memory buffers. S5_net uses both banks together to create 8 GB of global memory.

3. Heterogeneous Memory

    S5_net uses the four on-board quad data rate II (QDRII) memory interfaces to implement a total of 64 megabytes (MB) of heterogeneous memory for the Intel FPGA SDK for OpenCL Offline Compiler. By default, the host application allocates memory into the OpenCL global memory (that is, DDR3) when an OpenCL kernel program loads into the OpenCL runtime. However, based on the kernel arguments, the host might relocate memory to other buffers available on the computing card (that is, QDRII). Accesses to heterogeneous memory buffers are advantageous for network applications because they require the fast random access bandwidth that QDR provides.
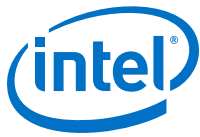
4. OpenCL I/O Channels

The two 10 Gbps Ethernet (10 GbE) I/Os connect to a full user datagram protocol (UDP) stack that provides an Avalon Streaming (Avalon-ST) interface for direct connection to OpenCL kernels.

5. FPGA Programming

The computing card uses the Configuration via Protocol (CvP)-capable PCIe hard IP. S5_net uses Intel FPGA CvP feature for implementing fast reprogramming over PCIe.

6. Guaranteed Timing

Guaranteed timing closure is achievable via the Intel Quartus Prime compilation flow for CvP. S5_net delivers a precompiled netlist in a `.personax` file that the offline compiler imports into each kernel compilation.

1.3 Contents of the Stratix V Network Reference Platform

The Stratix V Network Reference Platform is available for download on the Intel FPGA SDK for OpenCL FPGA Platforms page on the Altera website. Click **Custom** to reveal the download link.

Table 1.   Highlights of the Contents of s5_net

Windows File or Folder	Linux File or Directory	Description
`board_env.xml`	`board_env.xml`	eXtensible Markup Language (XML) file that describes s5_net to the SDK.
`windows64`	`linux64`	Contains memory-mapped device (MMD) library, kernel mode driver, and executables for the SDK utilities (that is, `install`, `flash`, `program`, `diagnose`, and `uninstall`) for your 64-bit operating system.
`hardware`	`hardware`	Contains the Intel Quartus Prime project template into which the Intel FPGA SDK for OpenCL Offline Compiler integrates kernels. The offline compiler then synthesizes the Intel Quartus Prime project files that implement the hardware of s5_net.
`source`	`source`	Contains source codes for the MMD library and SDK utilities in the `linux64` and `windows64` directories.
`include`	`include`	Contains header files necessary for compiling an OpenCL host application and accessing board-specific application programming interface (API) calls. For s5_net, these files are necessary for UDP initialization.

Related Links

Intel FPGA SDK for OpenCL FPGA Platforms page
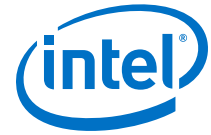
2 Developing Your Custom Platform

Use the tools available in the Stratix V Network Reference Platform 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:

1. *Intel FPGA SDK for OpenCL Custom Platform User Guide*
2. Contents of the Custom Platform Toolkit
3. *Stratix V Network Reference Platform Porting Guide*
4. Documentation for all the Intel FPGA IP in your Custom Platform
5. *Intel FPGA SDK for OpenCL Getting Started Guide*
6. *Intel FPGA SDK for OpenCL Programming Guide*

In addition, you must independently verify all the hard IPs on your computing card (for example, PCIe controllers, DDR3 external memory, and Ethernet).

Related Links

- Intel FPGA SDK for OpenCL Custom Platform Toolkit User Guide
- Intel FPGA SDK for OpenCL Getting Started Guide
- Intel FPGA SDK for OpenCL Programming Guide

2.1 Initializing Your Custom Platform

To initialize your Intel FPGA SDK for OpenCL Custom Platform, copy the Stratix V Network Reference Platform to another directory and rename it.

1. Download s5_net from the Intel FPGA SDK for OpenCL FPGA Platforms page on the Altera website. Click **Custom** to reveal to download link.
2. Store the `s5_net` directory into a directory that you own (that is, not a system directory) and then rename it (*<your_custom_platform_name>*).
3. Remove the *<your_custom_platform_name>*`/hardware/s5_net/persona` directory.
4. Rename the *<your_custom_platform_name>*`/hardware/s5_net` directory to match the name of your FPGA board (*<board_name>*).

ISO 9001:2008 Registered
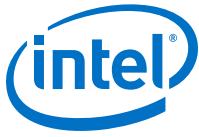
5. Modify the `name` attribute of the `board` XML element in the `board_spec.xml` file with *<board_name>*.

6. Modify the `board_env.xml` file so that the `name` and `default` fields match the changes you made in 4 on page 7 and 5 on page 8.

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

- Intel FPGA SDK for OpenCL FPGA Platforms page

- Listing the Available FPGA Boards in Your Custom Platform (-list-boards)

2.2 Removing Unused Hardware

After you store the Stratix V Network Reference Platform to your own directory and perform some preliminary modifications, modify the Intel Quartus Prime design files.

1. Instantiate your PCIe controller.

   For detailed instructions on instantiating your PCIe controller, refer to the *Getting Started with the Avalon-MM Stratix V Hard IP for PCI Express* section of the *Stratix V Avalon-MM Interface for PCIe Solutions User Guide*.

   For information on the design parameters for instantiating the PCIe controller in s5_net, refer to *Host-FPGA Communication over PCIe*, and the *Parameter Settings* section of the *Stratix V Avalon-MM Interface for PCIe Solutions User Guide*.

2. In Platform Designer (Standard), open the *<your_custom_platform_name>*/ `hardware/`*<board_name>*`/board.qsys` Platform Designer (Standard) system file. Remove the following components by selecting their names and then clicking **Remove** from the right-click menu:

   a. cpld_bridge_0

   b. qdr_0

   c. DDR3 memory controllers

      Because several components use the clock that ddr3a generates, it might be easier to remove only the second DDR3 controller (ddr3b) and reparameterize ddr3a to match your memory.

3. Remove the `cpld.sdc` file from the *<your_custom_platform_name>*/ `hardware/`*<board_name>* directory.

4. In Platform Designer (Standard), open the *<your_custom_platform_name>*/ `hardware/`*<board_name>*`/system.qsys` file. Remove the udp_0 component.

5. In the Platform Designer (Standard) **System** menu, click **Remove Dangling Connections** to remove invalid connection points between `system.qsys` and `board.qys`.

6. Modify both Intel Quartus Prime settings files (`.qsf`) to use only the pin-outs and settings for your system. Ensure that the only differences between the `base.qsf` and `top.qsf` files are in the settings in the `Revision Specific Settings` section of the files.

Related Links

- Getting Started with the Avalon-MM Hard IP for PCI Express
- Host-FPGA Communication over PCIe on page 17
- Parameter Settings

2.3 Integrating Your Custom Platform with the Intel FPGA SDK for OpenCL

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

1. Update the *<your_custom_platform_name>*/hardware/*<board_name>*/
   `board_spec.xml` file by removing the QDR and Ethernet channels from it.
   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`
   Platform Designer (Standard) system file.

   QDR section of the Stratix V Network Reference Platform's `board_spec.xml` file:

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

   Ethernet channels section of the `s5_net` `board_spec.xml` file:

   ```
   <channels>
       <interface name="udp_0" port="udp0_out"  type="streamsource"
   width="256"
        chan_id="eth0_in"/>
       <interface name="udp_0" port="udp0_in"  type="streamsink" width="256"
        chan_id="eth0_out"/>
       <interface name="udp_0" port="udp1_out"  type="streamsource"
   width="256"
        chan_id="eth1_in"/>
       <interface name="udp_0" port="udp1_in"  type="streamsink" width="256"
       chan_id="eth1_out"/>
   </channels>
   ```
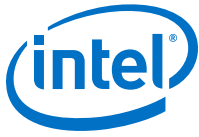
2. In the *<your_custom_platform_name>*/hardware/*<board_name>*/scripts
   directory, modify the `post_flow.tcl` file to not call the `create_fpga_bin.tcl`
   file. You can do so by commenting out the line of code containing the command
   `call_script_as_function scripts/create_fpga_bin.tcl`.

```
# Generate fpga.bin used for reprogramming
post_message "Generating fpga.bin"
if {[catch { call_script_as_function scripts/create_fpga_bin.tcl
$revision_name.sof
     $revision_name.core.rbf $revision_name.periph_hash $revision_name }
res]}
{
     post_message -type error "Error in create_fpga_bin.tcl! $res"
     exit 2
}
```

3. 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 `base.qsf` file in the *<your_custom_platform_name>*/
   hardware/*<board_name>* directory of your Custom Platform.

4. Perform the steps outlined in the *INTELFPGAOCLSDKROOT*/board/
   custom_platform_toolkit/tests/README.txt file to compile the
   INTELFPGAOCLSDKROOT/board/custom_platform_toolkit/tests/
   boardtest/boardtest.cl OpenCL kernel source file.

   The environment variable *INTELFPGAOCLSDKROOT* points to the location of the
   SDK installation.

   The hardware compilation stage will fail because of the absence of the `fpga.bin`
   file. However, the Intel Quartus Prime compilation should complete successfully
   and produce a `boardtest.aoco` Intel FPGA SDK for OpenCL Offline Compiler
   object file.

5. If compilation fails because of timing failures, fix the errors, or compile
   INTELFPGAOCLSDKROOT/board/custom_platform_toolkit/tests/
   boardtest.cl with different seeds by including the `-seed=`*<N>* option in the
   `aoc` command (for example, `aoc -seed=2 boardtest.cl`).

2.4 Setting up the Software Development Environment

Prior to building the software layer for your Intel FPGA SDK for OpenCL Custom
Platform, you must set up the software development environment.

2.4.1 Setting Up Software Development Environment for Windows

1. Install the GNU `make` utility on your Windows development machine.

   Note: Intel used the GNU `make` utility version 3.81a to build the software in the
   Stratix V Network Reference Platform.

2. Install Microsoft Visual Studio.
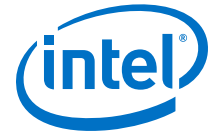
Note: Microsoft Visual Studio 2008 (9.0) was used to build the software in s5_net.

3. Set up the software development environment so that the Intel FPGA SDK for OpenCL user can invoke SDK commands and utilities at a command prompt.

4. 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.

5. In the `Makefile.common` file, set the *JUNGO_LICENSE* variable to your Jungo WinDriver license.

   For information on how to acquire a Jungo Windriver license, visit the Jungo Connectivity Ltd. website.

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

Related Links

Jungo Connectivity Ltd. website

2.4.2 Setting Up the Software Development Environment for Linux

1. Ensure that you use a Linux distribution that Intel supports.

   Note: Intel used the GNU Compiler Collection (GCC) version 4.2.3 to build the software in the Stratix V Network Reference Platform.

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

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

2.5 Building the Software in Your Custom Platform

You can build the software in your custom platform by modifying the library, driver, and source files provided in the Stratix V Network Reference Platform. To brand your custom platform, you must modify the MMD library, driver, and utilities provided in s5_net to include information specific to your board design.

1. In the software available with s5_net, ensure that you replace all references to s5_net to your Custom Platform.

2. Modify the `linklib` element in `<your_custom_platform_name>/board_env.xml` XML file to your custom MMD library name.

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

4. Modify the following files to include information of your Custom Platform:

   — For Windows, `<your_custom_platform_name>\source\include\hw_pcie_constants.h`

   — For Linux, `<your_custom_platform_name>/linux64/driver/hw_pcie_constants.h`

Update the following lines of code with information of your Custom Platform:

```
#define ACL_PCI_SUBSYSTEM_VENDOR_ID 0x1172
#define ACL_PCI_SUBSYSTEM_DEVICE_ID 0x0005
#define ACL_BOARD_PKG_NAME "s5_net"
#define ACL_VENDOR_NAME "Intel(R) Corporation"
#define ACL_BOARD_NAME "Network Reference Platform"
```

Note: The IDs must match the parameters in the PCIe controller hardware. For more information, refer to *PCIe Device Identification Registers*.

5. For Windows systems, update the `DeviceList` field in the *<your_custom_platform_name>*`\windows64\driver\acl_boards.inf` Setup Information file.

6. Run `make` in the*<your_custom_platform_name>*`/source` directory to generate the MMD library, driver, and utilities.

Related Links

PCIe Device Identification Registers on page 17

2.6 Establishing Host Communication

After you modify and rebrand the Stratix V Network Reference Platform to your own Custom Platform, use the tools and utilities in the Custom Platform to establish communication between your FPGA accelerator board and your host application.

1. Program your FPGA device with the *<your_custom_platform_name>*`/hardware/`*<board_name>*`/base.aocx` hardware configuration file and reboot your system.

2. Confirm that your operating system recognizes a PCIe device with your vendor and device IDs.

— For Windows, open the **Device Manager**

— For Linux, invoke the `lspci` command

3. Run the `aocl install <path_to_customplatform>` utility command to install the kernel driver on your machine.

4. Ensure that you properly set the *LD_LIBRARY_PATH* environment variable on Linux or the *PATH* environment variable on Windows.

For more information about the settings for *LD_LIBRARY_PATH* or *PATH*, refer to the *Setting the Intel FPGA SDK for OpenCL User Environment Variables* section of the *Intel FPGA SDK for OpenCL Getting Started Guide*.

5. To instruct the MMD software not to use CvP or flash memory to program the FPGA, perform one of the following tasks :

— To force the MMD to program via the `quartus_pgm` executable, set the environment variable *ACL_PCIE_FORCE_USB_PROGRAMMING* to a value of 1.

— To force the MMD to program via your custom programming method, modify the *<your_custom_platform_name>*`/source/host/mmd/acl_pcie_device.cpp` file. Trace the appearance of the environment variable *ACL_PCIE_FORCE_USB_PROGRAMMING* in the source code, and replace the existing instruction with your custom programming method.

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

7. Remake the MMD software.

8. Run the `aocl diagnose` utility command and confirm 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

- Host-FPGA Communication over PCIe on page 17
- Setting the Intel FPGA SDK for OpenCL User Environment Variables (Windows)
- Setting the Intel FPGA SDK for OpenCL User Environment Variables (Linux)
- Querying the Device Name of Your FPGA Board (diagnose)

2.7 Connecting the Memory

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 *DDR3 as Global Memory for OpenCL Applications* section.

2. Update the `<your_custom_platform_name>`/hardware/`<board_name>`/ `board_spec.xml` file to reflect the modifications.

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

4. Recompile the *INTELFPGAOCLSDKROOT*/board/custom_platform_toolkit/ `tests/boardtest/boardtest.cl` kernel source file.

   The environment variable *INTELFPGAOCLSDKROOT* points to the location of the SDK installation.

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

6. Modify the MMD source code to exit after checking the UniPHY status register in the function `wait_for_uniphy`. Rebuild the MMD software.

7. Run the `aocl diagnose` utility command 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`.

8. Consider using the Signal Tap logic analyzer to confirm the successful calibration of all memory controllers.

Related Links

DDR3 as Global Memory for OpenCL Applications on page 21

2.8 Integrating an OpenCL Kernel

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 *INTELFPGAOCLSDKROOT*`/board/` `custom_platform_toolkit/tests/README.txt` file to build the hardware configuration file from the *INTELFPGAOCLSDKROOT*`/board/` `custom_platform_toolkit/tests/boardtest/boardtest.cl` kernel source file.

   The environment variable *INTELFPGAOCLSDKROOT* points to the location of the Intel FPGA SDK for OpenCL installation.

2. Program your FPGA device with the `boardtest.aocx` Intel FPGA SDK for OpenCL Offline Compiler executable file and reboot your machine.

3. Remove the early-exit modification in the `version_id_test` function in the *<your_custom_platform_name>*`/source/host/mmd/` `acl_pcie_device.cpp` file that you implemented when you established communication between the board and the host interface.

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. The `.sln` file for Windows and the `Makefile` for Linux are available in the *INTELFPGAOCLSDKROOT*`/board/` `custom_platform_toolkit/tests/boardtest` directory.

   Attention: You must modify the `.sln` file to link it against the MMD library in your Custom Platform.

6. Set the environment variable *CL_CONTEXT_COMPILER_MODE_INTELFPGA* to a value of 3.

   For more information on this environment variable, refer to *Troubleshooting*.

Related Links

2.9 Programming Your FPGA Quickly Using CvP

After you verify that the host can program you FPGA device successfully, establish the CvP programming capability of your Custom Platform.

1. Invoke the following command to generate the CvP files:

   `quartus_cpf -c --cvp` *<revision_name>*`.sof` *<revision_name>*`.rbf`

   You may include this command in the *<your_custom_platform_name>*`/` `hardware/`*<board_name>*`/scripts/post_flow.tcl` file so that it generates the CvP files automatically after each compilation.

Your Intel Quartus Prime compilation directory should contain the files `<revision_name>`.sof, `<revision_name>`.periph.rbf, and `<revision_name>`.core.rbf files.

2. Program the `base.sof` file and then reboot your machine.

3. (Optional) You may use the Intel Quartus Prime Programmer to verify basic CvP functionality. Invoke the `quartus_cvp` command to program the `base.core.rbf` file.

4. Define the contents of your `fpga.bin` file by adding Tcl code to the `<your_custom_platform_name>`/hardware/`<board_name>`/scripts/ `post_flow.tcl` file that generates the `fpga.bin` file. Then, modify the MMD source code and the `program` utility so that you can use the file.

   You may use the existing format if you remove the proprietary host-to-flash programming over the cpld_bridge component from both the hardware and software.

5. If you set the environment *ACL_PCIE_FORCE_USB_PROGRAMMING* earlier, unset it. Then, set the environment variable *ACL_PCIE_FORCE_PERIPH_REPLACE_USB* to a value of 1. Alternatively, modify the `<your_custom_platform_name>`/ `source/host/mmd/acl_pcie_device.cpp` file to use CvP but not flash memory for reprogramming periphery changes. Flash programming is unavailable because of the removal of the cpld_bridge component.

6. Navigate to the directory containing the `boardtest.aocx` file. Invoke the command `aocl program` *`<device_name>`* `boardtest.aocx` to reprogram the device. Confirm that the message `Program succeed` appears.

   Note: By default, *`<device_name>`* is the acl number. If you have retained the default naming convention, invoke the `aocl program` command using acl0 as *`<device_name>`*. Alternatively, if you use another naming convention for *`<device_name>`*, use that in your `aocl` utility command.

Related Links

Programming the FPGA Offline or without a Host (program <device_name>)

2.10 Guaranteeing Timing Closure

When you modify the Stratix V Network 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 *FPGA System Design* and the *Intel FPGA SDK for OpenCL Custom Platform Toolkit User Guide*.

2. Compile several seeds of `boardtest.cl` until you generate a compiled design that achieves timing closure cleanly. Include the –seed=*`<N>`* option in your `aoc` command to specify the seed number.

3. Copy the `<path_to_s5_net>`/hardware/s5_net/persona/ `base.root_partition.personax` file into your Custom Platform.

4. Copy the `boardtest.aocx` file from the timing-closed compilation in Step 2 into your Custom Platform. Rename the file `base.aocx`.

5. Derive the top revision `top.qsf` file from your `base.qsf` file by including the changes described in the *CvP* section.

6. Remove the *ACL_QSH_COMPILE_CMD* environment variable.

7. Recompile `boardtest.cl`. In the Fitter Preservation section of the report, confirm that the Top partition is imported.

   The Incremental Compilation Placement Preservation section should show 100% placement for Top. Similarly, the Incremental Compilation Routing Preservation section should show 100% routing for Top.

8. Confirm that you can use the `.aocx` file to reprogram over CvP by invoking the `aocl program acl0 boardtest.aocx` command.

9. Ensure that the environment variable *CL_CONTEXT_COMPILER_MODE_INTELFPGA* is not set. Run the `boardtest_host` executable.

Related Links

- Intel FPGA SDK for OpenCL Custom Platform Toolkit User Guide
- CvP on page 36
- FPGA System Design on page 26

2.11 Troubleshooting

Set Intel FPGA SDK for OpenCL-specific environment variables to help troubleshoot Custom Platform design problems.

Table 2. **Intel FPGA SDK for OpenCL-Specific Environment Variables for Identifying Custom Platform Design Problems**

Environment Variable	Description
ACL_HAL_DEBUG	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.
ACL_PCIE_DEBUG	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.
ACL_PCIE_JTAG_CABLE	Set this variable to override the default `quartus_pgm` argument that specifies the cable number. The default is cable 1. If there are multiple Intel FPGA Download Cable, you can specify a particular one here.
ACL_PCIE_JTAG_DEVICE_INDEX	Set this variable to override the default `quartus_pgm` 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.
CL_CONTEXT_COMPILER_MODE_INTELFPGA	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. *Important:* When setting *CL_CONTEXT_COMPILER_MODE_INTELFPGA*, only use a value of 3.

3 Stratix V Network Reference Platform Design Architecture

Intel created the Stratix V Network Reference Platform 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.

3.1 Host-FPGA Communication over PCIe

To set up the PCIe hard IP that enables communication between the host and the FPGA board, you must configure the IP settings, and set various IDs, constants and parameters.

3.1.1 Parameter Settings for PCIe Instantiation

The Stratix V Network Reference Platform instantiates the Stratix V PCIe hard IP to implement a host-to-device connection over PCIe.

Dependencies

- Stratix V hard IP for PCI Express
- For Windows systems, Jungo WinDriver

Table 3.   Highlights of the Stratix V PCIe hard IP Configuration Settings

Parameter	Setting
Lanes	Lane rate: **Gen2 (5.0 Gbps)** Number of lanes: **x8** *Note:* This setting is the fastest configuration that can support CvP.
Rx buffer credit allocation	Low *Note:* This setting is derived experimentally.
Enable configuration via the PCIe link	On Click the check box to enable the setting.
Base Address Registers (BARs)	The design uses only a single BAR (BAR 0).
Address Translation Tables	Number of address pages: 256 *Note:* This setting is derived experimentally. Size of address pages: 12 bits *Important:* The number and size of the address pages must match the values in the MMD layer.

3.1.2 PCIe Device Identification Registers

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

Table 4.   Device Hardware-Related PCIe ID Registers

ID Register Name	Parameter Name in PCIe IP Core	Description
Vendor ID	vendor_id_hwtcl	Identifies the manufacturer of the FPGA device. Always set this register to 0x1172.
Device ID	device_id_hwtcl	Identifies the FPGA device. Set the device ID to the device code of the FPGA device on your accelerator board. For the Stratix V Network Reference Platform, this register is set to 0xD800 for the Stratix V D8 FPGA.
Subsystem Vendor ID	subsystem_vendor_id_hwtcl	Identifies the manufacturer of the accelerator board. Set this register to the vendor ID of manufacturer of your accelerator board. If you are a board vendor, set this register to your vendor ID.
Subsystem Device ID	subsystem_device_id_hwtcl	Identifies the accelerator board. The Intel FPGA SDK for OpenCL 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.

You can find these PCIe ID definitions in the PCIe controller instantiated in the
`board.qsys` system. These IDs are necessary in the driver and the SDK
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 programming
flow refers to the Device ID to ensure that it programs a device with a `.aocx` file
targeting that specific device.

3.1.3 Version ID

The Stratix V Network Reference Platform instantiates a version_id component that
connects to the PCIe Avalon master.

Before communicating with any part of the FPGA system, the PCIe 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 0 bus, or whenever the address
map changes.

3.1.4 Definitions of Hardware Constants in Software Header 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 Stratix V Network Reference Platform includes three header files that describe the hardware design to the software. The location of these header files are as follows:

- For Linux systems, the location is `<path_to_s5_net>`/linux64/driver

- For Windows systems, the location is `<path_to_s5_net>`\source\include

Table 5.   S5_net Header Files

Header File Name	Description
`hw_pcie_constants.h`	Header file that defines most of the hardware constants for the board design.
	Example constants in this file include the IDs described in the *PCIe Device Identification Registers* section, BAR number, and offset for different components in your design. In addition, this header file also defines the name strings of `ACL_BOARD_PKG_NAME`, `ACL_VENDOR_NAME` and `ACL_BOARD_NAME`.
	Keep the information in this file in sync with any changes to the board design.
`hw_pcie_dma.h`	Header file that defines direct memory access (DMA)-related hardware constants.
	Refer to *SG-DMA* for more information.
`hw_pcie_cvp_constants.h`	Header file that defines CvP-related hardware constants.
	Refer to *CvP* for more information.

Related Links

- SG-DMA on page 20
- CvP on page 36
- PCIe Device Identification Registers on page 17

3.1.5 PCIe Kernel Driver

A PCIe kernel driver is necessary for the OpenCL runtime library to access your board design via a PCIe bus.

The Stratix V Network Reference Platform PCIe kernel driver is in the following directory:

- For Windows systems, the driver is in the `<path_to_s5_net>` \windows64\driver folder

- For Linux systems, the driver is in the `<path_to_s5_net>`/linux64/driver directory

Use the Intel FPGA SDK for OpenCL `install` utility to install the kernel driver. Refer to *aocl install* for more information.

For Windows systems, the WinDriver API kernel driver 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 systems, an open-source, MMD-compatible kernel driver is available with s5_net.

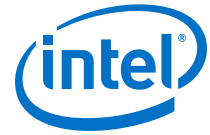**Table 6.** **Highlight of Files Available in the Linux Kernel Driver Directory**

File Name	Description
pcie_linux_driver_exports.h	Header file defining the special commands that the kernel driver supports. It defines the interface of the kernel driver. The MMD layer uses this header file to communicate with the device.
	After you install the kernel driver, it works as a character device. The basic operations to the driver are `open()`, `close()`, `read()`, and `write()`. To support more complex commands, an `acl_cmd struct` variable is necessary to pass the command of interest to the kernel driver through the `read()` or `write()` operation.
	To execute a command, perform the following tasks:
	1. Create a variable as type `acl_cmd_struct`.
	2. Specify the command you want to execute with the appropriate parameters.
	3. Send the command through a `read()` or `write()` operation.
aclpci.c	File that implements the basic structures and functions that a Linux kernel driver requires (for example, the `init` and `remove` functions, `probe` function, and functions that handle interrupts).
aclpci_fileio.c	File that implements the file I/O operations of the kernel driver. The s5_net Linux kernel driver supports four file I/O operations, namely `open()`, `close()`, `read()` and `write()`. Implementation of these file I/O operations allows the user application to access the kernel driver via file I/O system calls (`open/read/write/close`).
aclpci_cmd.c	File that implements the special commands defined in the `pcie_linux_driver_exports.h` file. Examples of these special commands include `SAVE_PCI_CONTROL_REGS`, `LOAD_PCI_CONTROL_REGS`, `DO_CVP`, and `GET_PCI_SLOT_INFO`.
aclpci_dma.c	File that implements DMA-related routine in the kernel driver. Refer to *SG-DMA* for more information.
aclpci_cvp.c	File that implements CvP-related routine in the kernel driver. Refer to *CvP* for more information.
aclpci_queue.c	File that implements a queue structure for use in the kernel driver. Such a queue structure eases programming.

Related Links

- aocl install on page 45
- SG-DMA on page 20
- CvP on page 36
- Jungo Connectivity Ltd. website

3.1.6 SG-DMA

The `acl_dma_core.qsys` file within the OpenCL SGDMA Controller IP encapsulates and parameterizes the modular scatter-gather SG-DMA hardware. The `board.qsys` system instantiates the `acl_dma.qsys` file within the OpenCL SGDMA Controller IP. For more information on SG-DMA, visit the Modular SG-DMA page on the Altera® Wiki website.

Hardware

The `acl_dma_core.qsys` file presents slave ports for the control and status registers (`dma_csr`) and the descriptors (`dma_descriptors`). It also provides separate masters for read and write operations. The `acl_dma.qsys` Platform Designer (Standard) System File adds the following features:

• An address span extender for non-DMA memory accesses

• A merged read/write master

The merged read/write master issues constant bursts of size 16, resulting in a 1/16 efficiency degradation from sharing the time interface. However, the bandwidth of this unit exceeds the bandwidth of the PCIe connection by more than this amount. Therefore, there is no observable host-to-memory bandwidth degradation.

Software

When the MMD receives a request for data transfer, it uses DMA when both of the following conditions are true:

1. The transfer size is bigger than 1024 bytes.

2. There are 64-byte alignments with the starting addresses for both the host buffer and the device offset.

Perform the following tasks to carry out a DMA transfer:

1. Check if there are remaining bytes to be sent.

2. Unpin the memory from the previous transfer.

3. Pin the memory for the new transfer.

4. Set up the Address Translation Tables on the PCIe.

5. Create and send the DMA descriptor.

6. Wait until the DMA finishes and then repeat Step 1.

Attention: For the Stratix V Network Reference Platform, this implementation is in the Linux kernel driver in the `<path_to_s5_net>`/`linux64/driver/aclpci_dma.c` file. For Windows systems, the implementation is in the `<path_to_s5_net>`\source\host \mmd\acl_pcie_dma_windows.cpp file.

Related Links
Modular SG-DMA page on Altera Wiki

3.2 DDR3 as Global Memory for OpenCL Applications

The Stratix V Network Reference Platform targets a computing card that has two banks of 4 GB x72 DDR3-160 SDRAM. Completion of the tasks below are necessary to access these banks as global memory for OpenCL applications.

For more information on the DDR3 UniPHY IP, refer to the *DDR2, DDR3, and DDR4 SDRAM Board Design Guidelines* section in Volume 2 of the *External Memory Interface Handbook*.

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

3.2.1 DDR3 IP Instantiation

The Stratix V Network Reference Platform uses two DDR3 controllers with UniPHY IP to communicate with the physical memories.

Table 7.   DDR3 SDRAM Controller with UniPHY IP Configuration Settings

IP Parameter	Configuration Setting
Timing Parameters	As per the computing card's data specifications.
Phase-locked loop (PLL)/delay-locked loop (DLL) Sharing	s5_net is configured such that both memory controllers can share the same PLL and DLL.
Avalon Width Power of 2	Currently, OpenCL does not support non-power-of-2 bus widths. As a result, s5_net uses the option that forces the DDR3 controller to power of 2. Use the additional pins of this x72 core for error checking between the memory controller and the physical module.
Byte Enable Support	OpenCL requires byte-level granularity to all memories; therefore, byte-enable support is necessary in the core.
Performance	Enabling reordering 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.
Debug	Debug is disabled for production.

After you instantiate the UniPHY IP, you typically need to run the `<variation_name>_pin_assignments.tcl` Tcl script to add additional constraints to the Intel Quartus Prime project. For more information on this process, refer to the *Adding Pins and DQ Group Assignments* section in Volume 2 of the *External Memory Interface Handbook*.
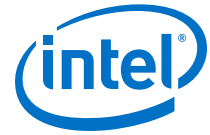
Related Links

Adding Pins and DQ Group Assignments

3.2.2 DDR3 Connection to PCIe Host

Connect all global memory systems to the host via the OpenCL Memory Bank Divider component.

The DDR3 UniPHY IP core has two banks where their width and address configurations match those of the DDR3 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 bank divider connect to their respective memory controllers. The Avalon slave connects to the PCIe and DMA cores. Implementations of appropriate clock crossing and pipelining are based on the design floorplan and 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 check for the successful calibration of the memory controller.

The `board.qsys` system uses a custom IP component named UniPHY Status to AVS to aggregate different UniPHY status conduits into a single Avalon slave port named `s`. This slave 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 DDR3 Connection to OpenCL Kernel

The OpenCL kernel needs to connect directly to the memory controller 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 bank divider connecting to the host. Appropriate pipelining also exists between the clock crosser and the memory controller.

3.3 QDRII as Heterogeneous Memory for OpenCL Applications

The OpenCL heterogeneous memory feature allows Intel FPGA SDK for OpenCL users to take advantage of the nonuniform memory architecture in a Custom Platform.
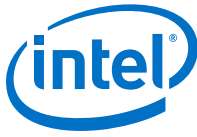
An SDK Custom Platform groups memories with similar characteristics into a single global memory system. Each Custom Platform has a designated default global memory system. In the case of the Stratix V Network Reference Platform, the default global memory system consists of the two DDR3 memory banks. The default global memory system must start at base address 0 from the host's perspective. Both the hardware design and the `board_spec.xml` file in the Custom Platform reflect this address assignment. In s5_net, the DDR global memory system is named *DDR*.

In addition to the DDR global memory system, the computing card that s5_net targets includes four banks of QDR memory. These four banks belong to a global memory system named *QDR*. SDK users can only allocate memory in the QDR global memory system using an attribute on their global memory buffers. All addressable global memory must be contiguous from the host's perspective; therefore, the QDR memory base address must start where the DDR memory ends.

For more information on the QDR UniPHY IP, refer to the *QDR II and QDR IV SRAM Board Design Guidelines* in Volume 2 of the *External Memory Interface Handbook*.

The procedure for implementing the QDR subsystem is similar to the one outlined in the *Developing Your Custom Platform* section. Below is a list of high-level tasks:

1. Instantiate and parameterize the UniPHY memory controllers.

2. Connect the UniPHY memory controllers to the host via a new OpenCL Memory Bank Divider instance.

3. Connect the UniPHY memory controller to the UniPHY Status to AVS component.

4. Export the UniPHY memory controller to the OpenCL kernel via clock-crossing bridges.

Below are special QDR subsystem design considerations for s5_net:

- QDR provides separate read and write ports.

  By default, the OpenCL Memory Bank Divider produces a single bidirectional master for each memory controller. In Platform Designer (Standard), select the **Separate read/write** ports option to support separate read and write masters. With respect to the kernel, instantiate clock crosses and separate read and write interfaces.

- The 275 MHz QDR afi clock and 4-to-1 multiplexing in the bank divider make it difficult to meet timing.

  To achieve timing closure robustly, open the `qdr.qsys` file in Platform Designer (Standard), and select the option to pipeline the outputs in the OpenCL Memory Bank Divider `memory_bank_divider_1`. Doing so adds a pipeline stage for each master that the bank divider creates.

Related Links

- Developing Your Custom Platform on page 7
- QDR II and QDR IV SRAM Board Design Guidelines

3.4 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.
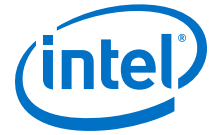
The Stratix V Network Reference Platform instantiates the OpenCL Kernel Interface component and sets the **Number of global memory systems** parameter to 2. The parameter setting is 2 because s5_net has DDR and QDR memories. Below is a list of connection settings in s5_net:

- For the default DDR memory, the generated `memorg_host0x018` conduit must connect to the DDR bank divider (`memory_bank_divider_0`).

- For the default DDR memory, the `config_addr` attribute in the `board_spec.xml` file must be set to `0x018`.

- For the QDR memory, the `memorg_host0x100` conduit must connect to the QDR bank divider (`memory_bank_divider_1`).

- For the QDR memory, the `config_addr` attribute in the `board_spec.xml` file must be set to `0x100`.

3.5 Implementation of UDP Cores as OpenCL Channels

OpenCL kernels can communicate directly with I/O using the Intel FPGA SDK for OpenCL channels extension.

For the Stratix V Network Reference Platform, Intel uses the PLDA QuickUDP IP to implement a full UDP stack on top of the available 10 GbE channels on the card. QuickUDP provides an Avalon-ST interface that can connect directly to the OpenCL kernel, allowing it to send and receive UDP network traffic without concern for UDP or lower-level protocols.

Attention: The UDP Hardware Stack QuickUDP IP is a licensed IP from PLDA. Refer to the PLDA website for information on acquiring and installing the appropriate license.

Caution: Improper installation of the QuickUDP IP license causes the SDK users to encounter the following error message when they compile with a Custom Platform that contains the QuickUDP IP:

```
Error (292014): Can't find valid feature line for core PLDA
QUICKTCP (73E1_AE12) in current license.
```

The error has no actual dependency on the PLDA QuickTCP IP.

Related Links

PLDA website

3.5.1 QuickUDP IP Instantiation

The Stratix V Network Reference Platform targets a computing card that has two 10 GbE channels. To access these channels, s5_net instantiates two PLDA QuickUDP IP cores.

The two 10 Gigabit Media Independent Interface (XGMII) interfaces from these cores connect to a single 10GBASE-R PHY with two channels. The Verilog instantiation of the PHY IP core is in the `<path_to_s5_net>`/hardware/s5_net/ip/quickudp/ `quickudp_wrapper.v` file. This file contains parameters such as the multitenant unit (MTU) and the number of sessions supported. Most parameters are accessible via QuickUDP's Avalon Memory-Mapped (Avalon-MM) slave interface.

3.5.2 QuickUDP Configuration via PCIe-Based Host

The Stratix V Network Reference Platform provides access to the PLDA QuickUDP IP configuration space to the host over PCIe by connecting `pipe_stage_host_ctrl` to the `config_udp0` and `config_udp1` interfaces of the s5_net `udp.qsys` subsystem.

Intel FPGA SDK for OpenCL users need to set their own parameters such as media access control (MAC), IP address, ports, and destinations. With the host access to QuickUDP via PCIe, the SDK users can configure the QuickUDP settings in their host software using the API in the `<path_to_s5_net>`/include/aocl_net.h header file.

3.5.3 QuickUDP Connection to OpenCL Kernel

Each PLDA QuickUDP IP core produces a read stream and a write stream, for a total of four Intel FPGA SDK for OpenCL channels available to the kernel. These streams cross into the kernel `clk` domain and are listed in the `board spec.xml` file.

Attention: The SDK supports only basic Avalon-ST with no packet support.

QuickUDP provides an Avalon-ST interface with full packet support along with additional metadata about the payload. Because OpenCL does not support the packet extensions, the packetization signals are converted to data, and the OpenCL application must handle all packetization.

QuickUDP also provides additional metadata that the application can use. For a full explanation of these signals, refer to the QuickUDP documentation on the PLDA website. In the Stratix V Network Reference Platform, the payload, packetization signals, and metadata are concatenated into a single 256-bit-wide vector exported as an Intel FPGA SDK for OpenCL channel.

Use the information in the following table to access the desired components of the channel's data:

Table 8.   Bit Mapping for the 256-Bit Intel FPGA SDK for OpenCL Channel to QuickUDP

Bit Range	Name	Description
[0:127]	payload	Packet payload
[128]	sop	Start of packet signal
[129]	eop	End of packet (EOP) signal
[130:133]	empty	On EOP, this field indicates how many bytes are unused
[134:149]	payload_size	Size of the packet Set to 0 for outbound packets
[150:181]	rem_ip	Indicates the remote IP for incoming packets
[182:197]	rem_port	Indicates the remote port for incoming packets
[198:205]	channel	Avalon channel
[206]	error	Avalon error signal

Related Links

PLDA website

3.6 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. These design complexities include a robust reset sequence, establishment of a design floorplan, global routing management, pipelining, and IP encryption. Optimizations of these design complexities occur in tandem with one another in order to meet timing and board hardware optimization requirements.
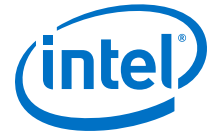
3.6.1 Clocks

The following clock domains affect the Platform Designer (Standard) hardware system:

- 250 MHz PCIe clock

- 200 MHz DDR3 clock

- 275 MHz QDR clock

- 156.25 MHz Ethernet clock

- 100 MHz general clock (`config_clk`)

- Kernel clock that can take on any clock frequency

With the exception of the kernel clock, the Stratix V Network Reference Platform is responsible for closing timing 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 this clock domain. It is crucial that this logic is minimal and achieves an Fmax higher than typical kernel performance.

Related Links
Guaranteed Timing Closure on page 31

3.6.2 Resets

The FPGA system design includes the implementation of the following reset drivers:

1. The `por_reset_counter` in the `board.qsys` system implements the power-on-reset. This reset issues a reset for a number of cycles after the FPGA completes configuration. It resets all the hardware on the device.

2. The PCIe bus issues a PERST reset that resets all hardware on the device.

3. The OpenCL Kernel Interface component issues the `kernel_reset` that resets all logic in the kernel clock domain.

The first two resets are combined into a single `global_reset`; therefore, there are only two reset sources in the system. However, these resets are explicitly synchronized across the various clock domains, resulting in several reset interfaces.

Important notes regarding resets:

1. Synchronizing resets to different clock domains might cause several high fan-out resets.

   Platform Designer (Standard) automatically synchronizes resets to the clock domain of each connected component. In doing so, Platform Designer (Standard) 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_ddr3a`; however, they are synchronized to the PCIe and DDR3 clock domains, respectively. Because both of these resets have high fan-out signals, they are assigned to global routing in the `.qsf` file.

2. 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 necessary for synchronous resets. In addition, a clock holding in reset might prevent 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`.

- Never reset the UniPHY controllers.

- Clock the reset controller for the Ethernet PHYs by its free-running reference clock.

3. 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 Stratix V Network Reference Platform, `kernel_reset` resets all the kernel clock-crossing bridges between DDR, QDR, and UDP on both the `m0_reset` and `s0_reset` interfaces.

3.6.3 Floorplan

The Intel FPGA SDK for OpenCL requires all board logic to be constrained along the edges of the FPGA device. This constraint provides a large contiguous space for OpenCL kernel implementation, which generally leads to better circuit performance (that is, Fmax).

The Stratix V Network Reference Platform floorplan below shows that all board interface logic are along the edges of the device. The logic in the center is the OpenCL kernel. At the bottom of the device are the PCIe and the two DDR3 cores. The QDR controllers are along the top of the device, and the two UDP stacks are on the right. The Stratix V global clock buffers are all around the middle of the device. This floorplan accommodates access to the global clock buffers by extending the bottom region edges up the left and right sides. This extension allows the placement of reset and other global routing drivers in the bottom region to be near the global clock buffer.

Figure 2.   S5_net Floorplan
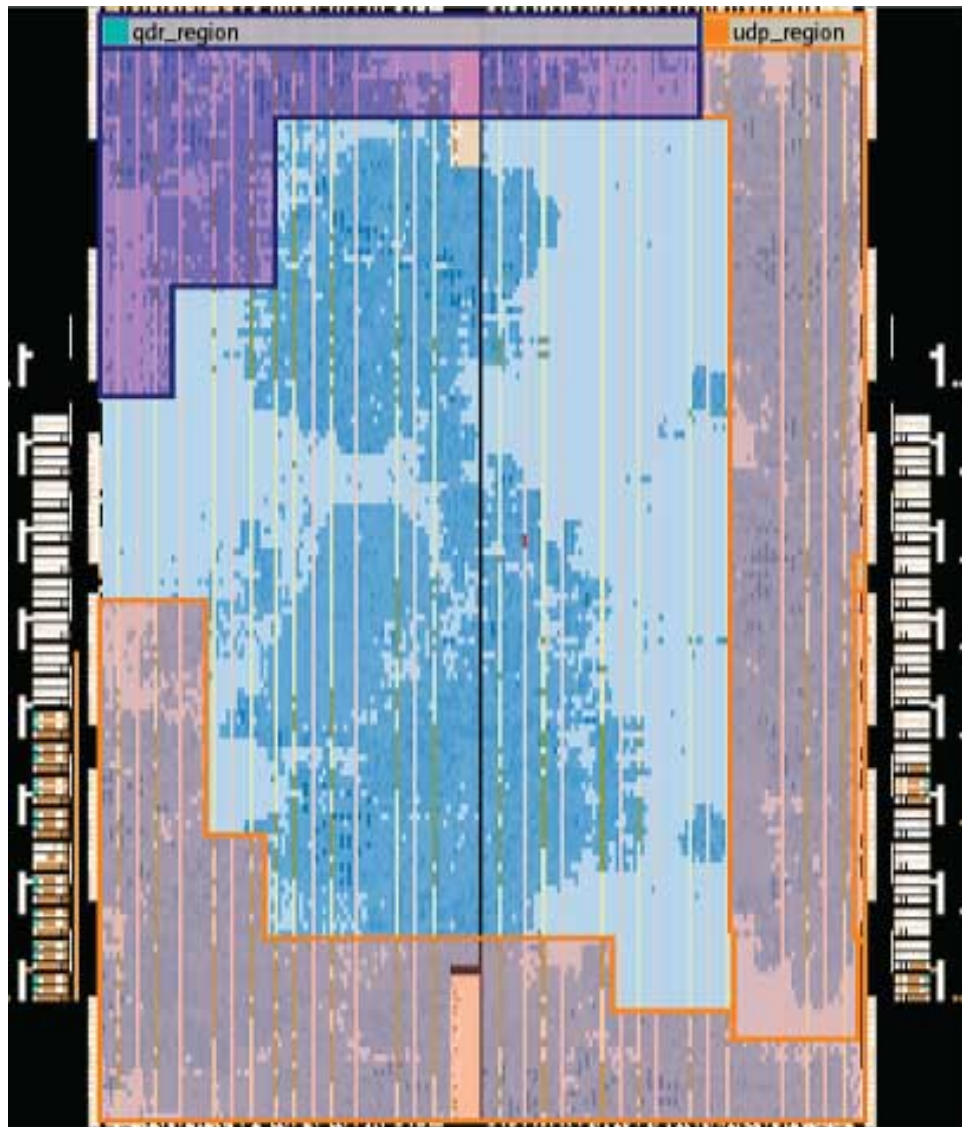
You can derive a floorplan for any board by following these steps:

1. Compile the design without any region constraints.

2. Examine the placement location of each of the IP cores in the Chip Planner.

3. Apply Logic Lock regions to push the IP cores to the edges of the device.

3.6.4 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 Stratix V Network Reference Platform targets, global routing can distribute high fan-out signals in the following manners:

1. Regional—Across any quadrant of the device
2. Dual-regional—Across any half of the device
3. Global—Across the entire device

Because there is no restriction on the placement location of the OpenCL kernel on the device, the kernel clocks and kernel reset must perform global distribution.

The DDR3 clock clocks all DMA logic and carries data into the QDR region at the top of the device. As a result, this clock and the reset synchronized to this clock domain also perform global distribution.

3.6.5 Pipelining

To implement pipelining in Platform Designer (Standard), refer to the *Platform Designer (Standard) Interconnect* chapter of the *Intel Quartus Prime Standard Edition Handbook* for more information.

Below are some specific examples of pipelining:

* Signals that traverse long distances because of the floorplan require additional pipelining.

    The DMA at the bottom of the FPGA must connect to the QDR memory at the top of the FPGA. QDR provides a 64-bit-wide interface at 275 MHz. The DMA is 512 bits in width at 200 MHz. This latter connection is converted to a 128-bit-wide 200 MHz interface in the `pipe_stage_qdr_host_0` module, which is pipelined in both command and response. This narrower bus enables crossing from the bottom region to the QDR region at the top, where it goes directly into `pipe_stage_qdr_host_1`. The configuration of the `pipe_stage_qdr_host_1` module is similar to `pipe_stage_qdr_host_0` to ensure no logic insertion between the two regions. This setup effectively implements pipelined routing in the 200 MHz clock domain, which the clock crosses into the 275 MHz domain. Finally, the width of the clock domain is adapted to ensure that this entire connection can still fully saturate the QDR bandwidth.

* The OpenCL kernel might need to connect the DDR interfaces at the bottom of the device and the QDR kernel interfaces at the top of the device.

    The kernel interfaces for QDR memory are located in the top region of the FPGA. However, the host and DDR connections originate from the bottom region of the FPGA. This distribution can force the kernel to stretch across the vertical span of the device, resulting in a slower Fmax. To minimize the decrease in Fmax, enable additional pipelining in the kernel when connecting to QDR memory. Add an `addpipe` attribute to each of the QDR `interface` element in `board spec.xml`, and assign it a value of 1.

Related Links

Platform Designer (Standard) Interconnect
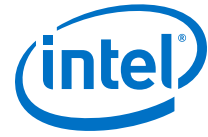
3.6.6 Encrypted IPs

The Stratix V Network Reference Platform incorporates two encrypted IP cores. They are the PLDA QuickUDP IP and the CPLD_bridge IP. The CPLD_bridge IP enables communication between the FPGA and external CPLD.

Incorporation of these IP cores in s5_net demonstrates that it is feasible to use the Intel FPGA encryption infrastructure to encrypt IPs within a Custom Platform.

Contact your field application engineer for more information on how to encrypt IP for use with the Intel Quartus Prime software.

3.7 Guaranteed Timing Closure

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 IP to generate the kernel clock, along with a post-flow script that ensures this clock is configured with a safe operating frequency confirmed by timing analysis. The Custom Platform imports a post-fit netlist that has already achieved timing closure on all nonkernel clocks.
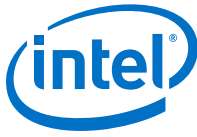
3.7.1 Supply the Kernel Clock

The OpenCL Kernel Clock Generator component provides the kernel clock and its 2x variant. For more information, refer to the *OpenCL Kernel Clock Generator* section of the *Intel FPGA SDK for OpenCL Custom Platform Toolkit User Guide*.

Figure 3.   The OpenCL Kernel Clock Generator Parameter Editor GUI

This figure shows where the **REF_CLK_RATE** parameter specifies the frequency of the reference clock that connects to the `pll_refclk`. In this case, the frequency is 100 MHz.
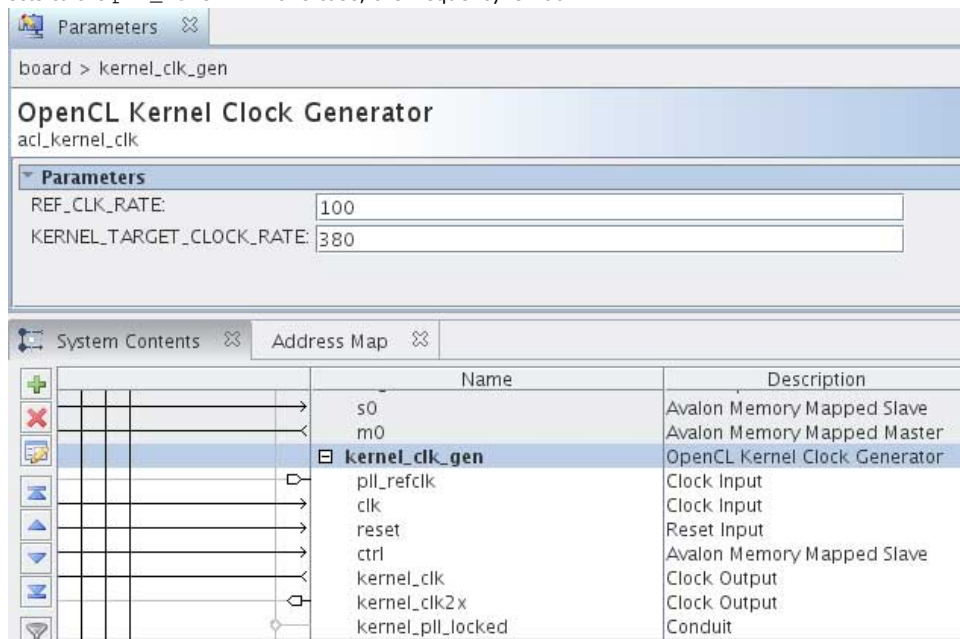
The **KERNEL_TARGET_CLOCK_RATE** parameter specifies the frequency that the Intel Quartus Prime software attempts to achieve during compilation. The board hardware contains some logic that the kernel clock clocks; at a minimum it includes the clock crossing hardware. To prevent this logic from limiting 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 Stratix V C2 device that the Stratix V Network Reference Platform targets, the **KERNEL_TARGET_CLOCK_RATE** is 380 MHz.

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

When developing your Custom Platform and attempting to close timing, add an overriding Synopsys Design Constraints (SDC) definition to relax the timing of the kernel. The following code example from the `<path_to_s5_net>/hardware/s5_net/top_post.sdc` file applies a 5 ns (200 MHz) maximum delay constraint on the OpenCL kernel during base revision compilations:

```
if {! [string equal $::TimeQuestInfo(nameofexecutable) "quartus_map"]}
{
  if { [get_current_revision] eq "base" }
  {
    post_message -type critical_warning "Compiling with slowed OpenCL Kernel clock."
    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
    }
  }
}
```

Caution: Applying this 5 ns SDC definition constrains both the kernel clock and the 2x clock to 5 ns, resulting in significantly slower kernel speeds.

Related Links

OpenCL Kernel Clock Generator

3.7.2 Guarantee Kernel Clock Timing

The OpenCL Kernel Clock Generator works together with a script that the Intel Quartus Prime database interface executable (`quartus_cdb`) runs after every Intel Quartus Prime software compilation as a post-flow script.

The following setting in the `base.qsf` and `top.qsf` files invokes the `<path_to_s5_net>/hardware/s5_net/scripts/post_flow.tcl` Tcl script in the Stratix V Network Reference Platform after every Intel Quartus Prime software compilation using `quartus_cdb`:
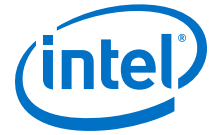`set_global_assignment -name POST_FLOW_SCRIPT_FILE "quartus_cdb:scripts/post_flow.tcl"`

Within this script, the following statement calls the OpenCL script to determine and configure the kernel clock to a functional frequency:
`source $::env(INTELFPGAOCLSDKROOT)/ip/board/bsp/adjust_plls.tcl`

where *INTELFPGAOCLSDKROOT* points to the path to the Intel FPGA SDK for OpenCL installation.

Important: Ensure that this flow executes during every Intel Quartus Prime software compilation of an OpenCL kernel.

3.7.3 Provide a Timing-Closed Post-Fit Netlist

Provide a timing-closed post-fit netlist that imports placement and routing information for all nodes clocked by nonkernel clocks.

Intel provides several mechanisms for preserving the placement and routing of some previously-compiled logic and for importing it into a new compilation. For the Stratix V Network Reference Platform, the following features are desirable from such a flow:

1. Timing preservation

2. Version compatibility to allow the import of the netlist into a newer Intel Quartus Prime software version

3. Strict preservation of the FPGA periphery to guarantee successful CvP programming

The Intel FPGA CvP compilation flow for the Stratix V device provides all of these features through an exported `.personax` file for the top-level partition. This means s5_net is configured with the project revisions and partitions necessary for implementing this flow. By default, the Intel FPGA SDK for OpenCL invokes the Intel Quartus Prime software on revision `top`. This revision is configured to import the `persona/base.root_partition.personax` file, which has been precompiled and exported from a base revision compilation.

For more information, refer to the *CvP* section.

Related Links

3.8 Addition of Timing Constraints

A Custom Platform must apply the correct timing constraints to the Intel Quartus Prime project. In the Stratix V Network Reference Platform, the `top.sdc` file contains all timing constraints applicable before IP instantiation in Platform Designer (Standard). The `top_post.sdc` file contains timing constraints applicable after Platform Designer (Standard). The order of the application is based on the order of appearance of the `top.sdc` and `top_post.sdc` in the `top.qsf` file.

One noteworthy constraint in s5_net 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. Although Intel requires all logic to exit reset mode in the same clock cycle, it is not necessary for the exit to happen in the same clock cycle as reset deassertion. Therefore, Intel adds a multicycle setup constraint of 2 and multicycle hold of 1 to the kernel reset. Without these additions, even with reset drivers located directly adjacent to global clock buffers, the highest kernel Fmax that Intel achieves is around 320 MHz.

3.9 Connection 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 that describes the hardware.

The following sections describe the implementation of these files for the Stratix V Network Reference Platform.

3.9.1 Describe s5_net to the Intel FPGA SDK for OpenCL

The `board_env.xml` file describes a Custom Platform to the Intel FPGA SDK for OpenCL. Details of each field in the `board_env.xml` file is available in the *Creating the board_env.xml File* section of the *Intel FPGA SDK for OpenCL Custom Platform Toolkit User Guide*.

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

The s5_net Reference Platform also supplies an end-user API for UDP initialization. The header in the *<path_to_s5_net>*`/include/aocl_net.h` file provides this API. The `compileflags` element in the `board_env.xml` file points the compiler to this directory when the SDK user invokes the `aocl compile-config` utility command to derive compiler arguments.

Related Links
Creating the board_env.xml File

3.9.2 Describe the s5_net Hardware to the Intel FPGA SDK for OpenCL

The Stratix V Network Reference Platform includes a `board_spec.xml` file that describes the hardware to the Intel FPGA SDK for OpenCL in the contexts described below.

Board

The `version` attribute of the `board` element must match the Intel Quartus Prime software version you use to develop the Custom Platform.

Device

The `device` section contains the name of the device model file available in the *INTELFPGAOCLSDKROOT*`/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. The value of `used_resources` equals the difference between the number of

adaptive logic modules (ALMs) used in final placement and the number of ALMs used for registers. You can derive this value from the Partition Statistic section of the Fitter report. Consider the following ALM categories within an example Fitter report:

```
; Statistic                                              ;
+-------------------------------------------------------
; Logic utilization (ALMs needed/total ALMs on device)   ;
; ALMs needed [=A-B+C]                                    ;
;   [A] ALMs used in final placement [=a+b+c+d]           ;
;     [a] ALMs used for LUT logic and registers           ;
;     [b] ALMs used for LUT logic                         ;
;     [c] ALMs used for registers                         ;
;     [d] ALMs used for memory (up to half of total ALMs) ;
;   [B] Estimate of ALMs recoverable by dense packing     ;
;   [C] Estimate of ALMs unavailable [=a+b+c+d]           ;
;     [a] Due to location constrained logic               ;
;     [b] Due to LAB-wide signal conflicts                ;
;     [c] Due to LAB input limits                         ;
;     [d] Due to virtual I/Os                             ;
```

The value of `used_resources` equals `ALMs used in final placement` minus `ALMs used for registers` (that is, `[A]` - `[c]`).

Global Memory

In the `board_spec.xml` file, there are separate `global_mem` sections for DDR and QDR memory, namely `DDR` and `QDR`, respectively. Assign `DDR` and `QDR` to the `name` attribute of the `global_mem` element. The **board** instance in Platform Designer (Standard) provides all of these interfaces; therefore, `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 s5_net targets, its `address` attribute begins at zero. Its `config_ddr` is `0x018` to match the `memorg` conduit used to connect to the corresponding Memory Bank Divider for DDR.

  Attention: The width and burst sizes must match the parameters in the Memory Bank Divider for DDR (`memory_bank_divider_0`).

- QDR

  The `QDR` section begins its `address` attribute directly after the DDR address space stops, and its `config_addr` is `0x100`, as indicated in the name of its `memorg` conduit. Because QDR provides separate read and write ports, each port is described to the Intel FPGA SDK for OpenCL Offline Compiler in a separate `port` attribute.

  As discussed in the *Pipelining* section, the `addpipe` option is necessary because the QDR kernel interfaces are at the top of the FPGA and the rest of the kernel interface signals are along the bottom of the device.

  Attention: The width and burst sizes must match the parameters in the Memory Bank Divider for QDR (`memory_bank_divider_1`).

Channels

The `channels` section describes the send and receive Avalon-ST channels for each of the UDP cores, for a total of four 256-bit Intel FPGA SDK for OpenCL Offline Compiler channels. These channel interfaces originate in the hardware from the udp_0 instance. Therefore, `udp_0` is specified in all the `name` attributes. The `port` attribute identifies the name of the Platform Designer (Standard) interface to which a channel connects. The `chan_id` attribute is the identifier with which the SDK user declares the channel.

Interfaces

The `interfaces` section describes kernel clocks, reset, CRA, and snoop interfaces. The Memory Bank Divider for the default memory (in this case, `memory_bank_divider_0`) exports the snoop interface described in the `interfaces` section. The width of the snoop interface should match the width of the corresponding streaming interface.

Compile

The `compile` section describes your Intel Quartus Prime project and provides the commands necessary to generate and synthesize the RTL in your design. In addition, it explicitly registers itself with the Automigration platform. If you derive your Custom Platform design from s5_net, set the `platform_type` parameter of the `auto_migrate` attribute to `s5_net`.

Related Links

- Creating the board_spec.xml File
- Pipelining on page 30

3.10 FPGA Programming Flow

There are three ways to program the FPGA. To replace only the FPGA core, use CvP programming. To replace both the FPGA periphery and the core, use Flash programming (if available), or the Intel Quartus Prime Programmer command-line executable (`quartus_pgm`) programming via cables such as Intel FPGA Download Cable.

The default FPGA programming flow is to compare the periphery currently programmed on the FPGA with the periphery of a new design. If they match, programming through CvP replaces the existing FPGA core with the new core. If they differ, programming through external flash memory replaces the existing FPGA periphery with the new design. FPGA programming using `quartus_pgm` via Intel FPGA Download Cable is an old approach. Only use this programming method 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. The `quartus_pgm` approach remains for development and testing purposes, and for use on boards that do not have an alternative method (such as Flash) for periphery replacement.

3.10.1 CvP

The CvP feature enables core logic update over the PCIe hard IP for Stratix V and Arria V GZ devices. Refer to the *Configuration via Protocol (CvP) Implementation in V-series FPGA Devices User Guide* for more information.

Related Links

3.10.1.1 Replacing FPGA Core Logic via CvP Programming

To successfully program the FPGA core logic, the Fitter must ensure that all FPGA periphery programming bits remain unchanged. The CvP revision flow expresses this hard constraint to the Fitter. Use the Stratix V Network Reference Platform CvP revision flow to achieve reliable CvP programming of the core logic.

1. Create a base revision. In the Stratix V Network Reference Platform, the base revision is the `<path_to_s5_net>`/hardware/s5_net/base.qsf file.

2. Create a CvP update revision.

   This update version is derived from the base revision and includes an imported `.personax` file. The `.personax` file is created during a base revision compilation. It includes the root partition imported from the base revision compilation. In s5_net, this CvP update revision is the `top.qsf` file, which becomes the project revision that the Intel FPGA SDK for OpenCL Offline Compiler compiles by default.

3. Create a kernel partition in both base and update revisions (marked as having multiple personas).

4. Store the base revision compilation programming file output in the Flash memory as the power-up configuration.

5. Use the CvP update revision compilation programming file output for all subsequent FPGA configurations.

3.10.1.2 Specifying Configuration via PCI Express Options

To enable Configuration via Protocol programming of the FPGA core logic, specify the configuration via PCIe options in the Intel Quartus Prime software.

1. In the **Stratix V HIP for PCI Express** parameter editor GUI, under **System Settings**, select **Enable configuration via the PCIe link** to enable CvP on the PCIe IP.

2. Include the following INI settings in the `quartus.ini` file:

   ```
   skip_hssi_gen3_pcie_hip_cvp_enable_rule = on
   skip_hssi_gen3_pcie_hip_hip_hard_reset_rule = on
   skip_hssi_gen3_pcie_hip_hrdrstctrl_en_rule = on
   ```

3. In the Intel Quartus Prime software, click **Assignments ➤ Device ➤ Device and Pin Options** to open the **Device and Pin Options** dialog box.

4. Under **General**, select **Enable autonomous PCIe HIP mode**.

5. Under **CvP Settings**, perform the following tasks:

   a. Set **Configuration via Protocol** to **Core update**.

   b. Select **Enable CvP_CONFDONE pin**.

   c. Select **Enable open drain on CvP_CONFDONE pin**.
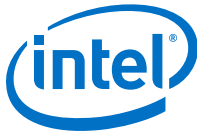
The PCIe core must have the `force_hrc` parameter set to a value of `1` in the `board.qsys` file. Because you cannot set this parameter using the **Platform Designer (Standard)** GUI, you must save and exit Platform Designer (Standard), and then edit the setting in the `board.qsys` file.

Attention: Depending on whether the PCIe core is used in the base or CvP revision, additional modifications to the PCIe controller behavior might be necessary. An additional multpersona partition named `cvp_update_reset_partition` is implemented to work in conjunction with the following file edits:

- Replacement of `altpcie_sv_hip_ast_hwtcl.v` and `altpcie_hip_256_pipen1b.v` in the *<path_to_OpenCL_kernel_filename_directory>*`/system/synthesis/ submodules` directory.

- Addition of `cvp_update_reset.v` (for base revision) and `cvp_update_reset_zero.v` (for CvP update revision) in the *<path_to_OpenCL_kernel_filename_directory>*`/system/synthesis/ submodules` directory.

After Platform Designer (Standard) Verilog generation and before launching the Intel Quartus Prime compilation, the *<path_to_s5_net>*`/hardware/s5_net/scripts/ pre_flow.tcl` Tcl script performs these file edits automatically. The following Verilog source files reside in the *INTELFPGAOCLSDKROOT*`/ip/board/migrate/ cvpupdatefix` directory, where *INTELFPGAOCLSDKROOT* points to the Intel FPGA SDK for OpenCL installation directory:

- `altpcie_sv_hip_ast_hwtcl.v`

- `altpcie_hip_256_pipen1b.v`

- `cvp_update_reset.v`

- `cvp_update_reset_zero.v`

3.10.1.3 Base versus CvP Update Revisions in CvP Programming

Base revision and CvP update revision have separate but almost identical Intel Quartus Prime Settings Files named `base.qsf` and `top.qsf`, respectively.

The `base.qsf` file includes the following parameter settings:

```
set_global_assignment -name VERILOG_FILE system/synthesis/submodules/
cvp_update_reset.v
set_global_assignment -name CVP_REVISION top
set_global_assignment -name ROUTING_BACK_ANNOTATION_FILE
super_kernel_clock.rcf
```

The `top.qsf` file includes the following parameter settings:

```
set_global_assignment -name VERILOG_FILE system/synthesis/submodules/
cvp_update_reset_zero.v
set_global_assignment -name REVISION_ TYPE CVP
set_global_assignment -name BASE_REVISION base
set_global_assignment -name INPUT_PERSONA persona/
base.root_partition.personax -section_id Top
```
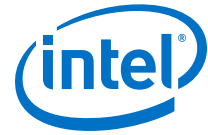
Figure 4. **Base and CvP Revisions in top.qpf**

When you open the top-level Intel Quartus Prime Project File `<path_to_s5_net>/hardware/s5_net/top.qpf` in the Intel Quartus Prime software, the Project Navigator displays the base revision (base) and the CvP revision (top).

Figure 5. **Base Revision Design Partitions**

When you click **Assignments ➤ Design Partitions Window**, the **Design Partitions** window lists two defined design partitions for the base revision.

- `acl_kernel_partition` contains all kernel-related logic.
- `cvp_update_reset_partition` contains the fix that works in conjunction with the `pre_flow.tcl` script to modify the PCIe controller behavior.

For both partitions, the **Allow Multiple Personas** parameter is set to **On**. This setting indicates that the board interface logic remains unchanged, but these partitions might change across different compilations. The `acl_kernel_partition` comprises of different kernel logic resulting from the compilation of the OpenCL kernel source code, as shown below:

Figure 6. **CvP Revision Design Partitions**

The Top partition, which contains everything other than `acl_kernel_partition` and `cvp_update_reset_partition`, preserves placement-and-routing by importing the input persona `persona/base.root_partition.personax` file.

In the case of a base revision compilation, invoking the `export_persona –overwrite -partition Top` Tcl command exports the `persona/base.root_partition.personax` file from the `<path_to_s5_net>/hardware/s5_net/scripts/post_flow.tcl` file. This `.personax` file is imported into all subsequent CvP update revision compilations. It preserves the placement and routing of all non-kernel-related logic so that CvP updates only change the logic residing in the kernel partition. Within the kernel partition, the `kernel_system.qsys` system is automatically generated at compilation time. It is a Platform Designer (Standard) subhierarchy of `system.qsys`. It contains one or more kernel IPs and wrapper logic to connect the kernel partition to the OpenCL board logic in `board.qsys` and the UDP logic in `udp.qsys`.

By comparing the hash of the periphery currently programmed to the FPGA against that of the new design, the following function in the `acl_pcie_flash.cpp` MMD file decides whether CvP programming is sufficient. If not, a fall-back method such as Flash programming is necessary to reprogram the device periphery: `ACL_PCIE_FLASH::does_programmed_periphery_differ_from_fpga_bin()`

Note: S5_net uses a `SHA-1` hash function.

Attention: Populating the periphery hash within the FPGA configuration bitstream changes the periphery hash value derived from the bitstream.

To avoid this update loop, the Intel Quartus Prime compilation uses a ROM initialization value of all zeros. The output from this compilation goes into the computation of the periphery hash. Then, when you run `quartus_cdb --update_mif`, it replaces the ROM value with the output hash from the original compilation.

Communicating with Flash Memory over PCIe

In s5_net, the host communicates with Flash memory through a CPLD that connects to a set of FPGA pins. The CPLD is located between the FPGA and Flash, and it masters communication between the FPGA and off-chip peripherals. A CPLD_bridge IP block is instantiated in the locked-down interface portion of the FPGA design. It provides memory-mapped communication between the PCIe controller on the FPGA and the external CPLD communication bus.

The CPLD uses a custom packet-based communication protocol for communication with the FPGA. The MMD host code creates the necessary packets, and transmits them over PCIe to the CPLD_bridge on the FPGA. The bridge in turn communicates the packets to the CPLD for further processing and routing. Flash programming commands are embedded within these packets.

Flash Memory Programming

A full programming bitstream is stored in the Flash memory. The operations necessary for programming are specific to the Flash chip. For more information on the configuration protocol, refer to the source code of the `acl_pcie_flash.cpp` MMD file and the Flash device datasheet.

The high-level Flash memory programming tasks are as follows:

1. Erase the Flash lines that you want to program.

2. Program the data lines.

3. Read back the data to verify that the programming bitstream is correct.

In s5_net, raw binary file (RBF) bitstreams are programmed to the Flash memory because the configuration hardware on the board expects the `.rbf` file format. Alternative file formats might be necessary on boards with different configuration methods. You can use Intel Quartus Prime software utilities, such as `quartus_cdb`, to perform file format conversion using the post-flow scripts (`scripts/post_flow.tcl` and subscripts).

Note: For simplicity, the FPGA bitstreams are not compressed in s5_net.

S5_net verifies the successful programming of the Flash memory (that is, no bit errors). In a production environment where programming speed is of concern, you can take multiple steps to reduce the Flash programming time. For example, you can use compressed bitstreams, reduce or eliminate the verification of Flash contents, and remove multiple busy wait loops in the Flash programming code. Because s5_net is intended as an instructional proof of concept, it is not optimized for programming time.

If the device is configured with a compressed bitstream, then CvP must also use a compressed bitstream.

Base and CvP Revisions for Flash Programming

The Intel Quartus Prime compilation of an OpenCL kernel can produce two different compilation revisions: base and CvP. For more information on these revisions, refer to the *CvP* section.

S5_net uses Flash programming for two purposes:

1.   Modification of the FPGA periphery configuration.

2.   Replacement of the power-on Flash configuration image (using the Intel FPGA SDK for OpenCL `flash` utility).

Modifying the FPGA periphery requires a bitstream from a CvP revision compilation. Replacing the power-on image requires an RBF from a base revision compilation to guarantee CvP reliability. RBF bitstreams are large. To avoid storing both the base and CvP revision compilation RBF files for every design, only include the base revision RBF in the `fpga.bin` file. As a result, you can use the SDK `flash` utility to replace the power-on bitstream with the RBF in the `fpga.bin` file. However, periphery replacement in a live system becomes more complicated. The base revision compilation contains the correct periphery for an SDK user's design, but it does not contain the design itself. The design is only available in a CvP revision compilation. The solution is to replace the periphery through Flash programming using the base revision compilation, and then to immediately CvP program the user's design on top of that periphery. The result is identical to programming a full RBF bitstream from the user's CvP revision compilation, but without storing that RBF.

FPGA Reprogramming from Flash

After you program the Flash memory with the new configuration bitstream from a base revision compilation, you must reconfigure the FPGA in the live system by performing the following tasks:

1.   Reprogram the FPGA from the bitstream in Flash memory.

2.   Wait for device programming to complete.

3.   Restore PCIe link and verify communication with the FPGA.

4.   Program the SDK user design core onto the FPGA via CvP. Refer to the *CvP* section for more information.

S5_net performs FPGA reprogramming from Flash via a control command to the CPLD by the host, through the bridge on the FPGA. Details of the command are specific to the board hardware and are different across manufacturers.

Flash programming restores the PCIe link in the same way as programming via `quartus_pgm`. The PCIe configuration space is saved on the host before reprogramming. After reprogramming, the registers are restored by copying the original configuration data from the host to the device configuration space across PCIe. PCIe advanced error reporting (AER) is disabled during the programming operation because the FPGA effectively disappears from the PCIe bus during programming, which is typically a fatal PCIe event (Basic Input/Output System (BIOS) often halts the CPU). By restoring the PCIe configuration space registers after reprogramming, the device can communicate with the host using the same configuration as the original power-on PCIe enumeration.

From the host computer's perspective, the FPGA PCIe endpoint remains unchanged. After PCIe communicates with the FPGA that has the new and verified periphery configuration, CvP programming populates the FPGA core with the OpenCL kernel design. Refer to the *CvP* section for more details.

Important: S5_net targets a board with Flash memory that stores the power-on FPGA configuration bitstream.

When changing the periphery through Flash programming at runtime, to avoid overwriting the power-on bitstream, you may use a different region of Flash memory as the intermediate storage location. However, this technique requires a means to specify the Flash memory address from which the FPGA will be reprogrammed. For boards without the ability to load from multiple Flash regions dynamically, you might need to overwrite the power-on programming bitstream.

Related Links

CvP on page 36

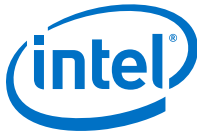## 3.10.3 Defining the Contents of the fpga.bin File

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.

Table 9.   Contents of the s5_net fpga.bin File

The contents of the `fpga.bin` file in the Stratix V Network Reference Platform are defined as an Executable and Linkable Format (ELF) library that organizes the various fields.

Field	Description
`.acl.sof`	The full programming bits for the compiled design.
`.acl.core.rbf`	The CvP programming bits for the compiled design.
`.acl.periph.hash`	The hash of the `periph.rbf` file that the current compilation generates. This hash is also embedded in the on-chip Hash ROM. The Hash ROM is compared against this hash to determine, ahead of time, whether CvP programming will succeed.
`.acl.compile_revision`	The name of the compiled Intel Quartus Prime project revision.
	continued...

Field	Description
`.acl.pcie.dev`	The device ID of the PCIe controller. The PCIe device ID is set to match the FPGA part number (for example, D8). This field is compared to the FPGA part number to ensure that the programming files correspond to the device undergoing programming.
`.acl.base_revision.rbf`	The full-FPGA `.rbf` file of the base revision compilation used to generate the post-fit netlist. This `.rbf` file must be the power-on image of the FPGA. All other designs can be programmed via CvP on top of this image.
`.acl.base_revision.periph.hash`	The hash of the `periph.rbf` file of the base revision compilation from which the post-fit netlist is derived. This field is retrieved from the `base.aocx` file and should match the `.acl.periph.hash` field for any SDK user compilation.

3.11 Host-to-Device MMD Software Implementation

The MMD layer is a thin software layer for communicating with 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 *<TOP_DEST_DIR>*/source/include/aocl_mmd.h file, where *<TOP_DEST_DIR>* 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 *<TOP_DEST_DIR>*/source/host/mmd directory. 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*.

acl_pcie.cpp

The `acl_pcie.cpp` file implements the MMD API and provides multiple devices support. This file also handles the PCIe interrupt. For Linux, the kernel driver uses signal to notify the MMD about an interrupt from the PCIe. For Windows, the MMD use the Jungo WinDriver API to handle the interrupt from PCIe.
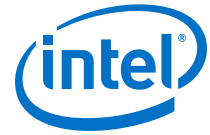
In addition, this file includes a signal handler for Ctrl-C event. The MMD needs to capture the Ctrl-C event to ensure that the program does not terminate itself during unsafe operation, such as programming the device or running `quartus_pgm`.

acl_pcie_device.cpp

The `acl_pcie_device.cpp` file implements a class to represent a device, and abstracts details to allow easier handling of multiple devices. Examples of the supported operations of the device by this class include `write_block`, `read_block`, `reprogram`, and `flash`.

During the instantiation of an instant for the device class, the following verifications take place:

1. Ensures that the kernel driver is installed and that its version matches the MMD version.

2. Ensures that the device with the given name can be found.

3. Ensures that the `Version ID` of the device matches the supported ID in the software.

4. Waits for UniPHY IP calibration.

acl_pcie_mm_io.cpp

The `acl_pcie_mm_io.cpp` file implements a class to allow access to the device as a memory-mapped I/O. It provides access to `GLOBAL-MEM`, `PCIE-CRA`, `DMA-CSR`, `DMA-DESCRIPTOR`, `KERNEL`, etc.

acl_pcie_dma_linux.cpp or acl_pcie_dma_windows.cpp

The `acl_pcie_dma_linux.cpp` or `acl_pcie_dma_windows.cpp` file implements DMA-related functions. For more information, refer to the *SG-DMA* section.

acl_pcie_config.cpp

The `acl_pcie_config.cpp` file implements functions for configuring the device. For more information, refer to the *FPGA Programming Flow* section.

acl_pcie_flash.cpp

The `acl_pcie_flash.cpp` file implements Flash-related functions. For more information, refer to the *Flash* section.

acl_pcie_quickudp.cpp

The `acl_pcie_quickudp.cpp` file implements UDP-related functions. For more information, refer to the *Implementation of UDP Cores as OpenCL Channels* section.

acl_pcie_debug.cpp

The `acl_pcie_debug.cpp` file defines the commonly used debug functions and parameters. It is included by most of the other files.

acl_pcie_timer.cpp

The `acl_pcie_timer.cpp` file implements a timer module to measure performance.

Related Links

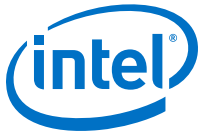- SG-DMA on page 20
- FPGA Programming Flow on page 36
- Flash on page 40
- Implementation of UDP Cores as OpenCL Channels on page 24
- MMD API Descriptions

3.12 OpenCL Utilities Implementation

A Custom Platform requires a set of Intel FPGA SDK for OpenCL utilities for managing the FPGA board.

3.12.1 aocl install

The `install <path_to_customplatform>` utility 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.

Attention: You must have write privileges to the SDK directory to install the kernel directory.

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 <path_to_customplatform>` utility 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.
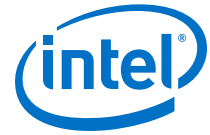
3.12.3 aocl program

The `program` utility programs the board with the specified `.aocx` file. Calling the `aocl_mmd_reprogram()` MMD API function implements the `program` utility.

3.12.4 aocl flash

The `flash` utility configures the power-on image for the FPGA using the specified `.aocx` file. Calling into the MMD library implements the `flash` utility.

3.12.5 aocl diagnose

The `diagnose` utility 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.

You can run the `diagnose` utility for multiple devices (that is, `aocl diagnose <device_name1> <device_name2> <device_name3>`). If you want to run the `diagnose` utility for all devices, use the `all` option (that is `aocl diagnose all`).

3.12.6 aocl list-devices

The `list-devices` utility lists all the devices installed in a host machine, grouped by board packages.

The `list-devices` utility is similar to the `diagnose` utility. It first verifies the installation of the kernel driver and then lists all the devices.

3.13 Stratix V Network Reference Platform Implementation Considerations

The implementation of the Stratix V Network Reference Platform includes some workarounds that address certain Intel Quartus Prime software known issues.

1. The `quartus_map` 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_map` has read the file before checking the current version.

2. Configuration via PCIe requires the `force_hrc` parameter to have a value of `1`, and the inclusion of the three PCIe INI settings described in the *CvP* section.

3. The kernel clock requires a lot of connectivity. Therefore, Intel recommends compiling the base revision using the `super_kernel_clock.rcf` Routing Constraints File.

4. Use the INI setting `bpm_hard_block_partition=off` to improve version compatibility.

5. Use the INI setting `qic_pf_no_input_rotation=on` to prevent certain failures to route.

6. The CvP revision (that is, top), which imports the `.personax` file, must include auto global clock promotion for clocks, resets, and clock enable signalss.

7. To avoid certain routing failures, set the **Fitter Preservation Level** for the Top partition to **Netlist Only**. You may assign the setting via the **Design Partitions Window** or the **Tcl Console**.

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

1. The `pll_rom.hex` file exists before compilation.

2. Intel Quartus Prime compilation is only ever performed after the Intel FPGA SDK for OpenCL Offline Compiler has embedded an OpenCL kernel inside the system.

3. Perform Intel Quartus Prime compilation after you install the Intel FPGA SDK for OpenCL and set the *INTELFPGAOCLSDKROOT* environment variable to point to the SDK installation.

4. The name of the directory where the Intel 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.

5. The *PATH* or *LD_LIBRARY_PATH* environment variable must point to the MMD library in the Custom Platform.

Related Links

CvP on page 36

A Document Revision History

Table 10. **Document Revision History of the Stratix V Network Reference Platform Porting Guide**

Date	Version	Changes
November 2017	2017.11.03	• Rebranded the following: — Environment variable *ALTERAOCLSDKROOT* to *INTELFPGAOCLSDKROOT*. — Environment variable *CL_CONTEXT_COMPILER_MODE_ALTERA* to *CL_CONTEXT_COMPILER_MODE_INTELFPGA* — USB Blaster Cable to Intel FPGA Download Cable. — SignalTap II Logic Analyzer to Signal Tap logic analyzer. — Quartus Prime to Intel Quartus Prime — LogicLock to Logic Lock — Qsys to Platform Designer (Standard) • Implemented single dash and `-option=<value>` conventions in the following topics: — Initializing Your Custom Platform on page 7 — Integrating Your Custom Platform with the Intel FPGA SDK for OpenCL on page 9 — Guaranteeing Timing Closure on page 15 • Added a new topic aocl list-devices on page 47. • Updated the topic aocl diagnose on page 46 to include support for diagnosing multiple devices and all devices. • Updated the topics aocl install on page 45 and aocl uninstall on page 46 to include the path to custom platform during installation and uninstallation. • In Establishing Host Communication on page 12, removed reference to the environment variable *AOCL_BOARD_PACKAGE_ROOT* since it is deprecated and updated instances of `aocl install` updated as `aocl install <path_to_customplatform>`.
May 2017	2017.05.08	• Maintenance release.
October 2016	2016.10.31	• Rebranded Altera SDK for OpenCL to Intel FPGA SDK for OpenCL. • Rebranded Altera Offline Compiler to Intel FPGA SDK for OpenCL Offline Compiler. • In *Building the Software in Your Custom Platform*, updated the code snippet in Step 4 from `#define ACL_VENDOR_NAME "Altera Corporation"` to `#define ACL_VENDOR_NAME "Intel(R) Corporation"`.
May 2016	2016.05.02	Maintenance release.
November 2015	2015.11.02	Maintenance release, and made the following updates: • Changed instances of *Quartus II* to *Quartus Prime*. • Changed instances of *Altera Complete Design Suite* to *Intel Quartus Prime Design Suite*®.

ISO 9001:2008 Registered

Date	Version	Changes
May 2015	15.0.0	Maintenance release.

December 2014	14.1.0	• Renamed the document as *Altera Stratix V Network Reference Platform Porting Guide*.
		• Updated the *Legacy Board Support* section. Added reference to the *Custom Platform Automigration for Forward Compatibility* section in the *Altera SDK for OpenCL Custom Platform Toolkit User Guide* for automigration information and instructions.
		• In the *Implementation of UDP Cores as OpenCL Channels* section, added notes indicating the following:
		  1. The QuickUDP IP requires a license from PLDA.
		  2. Improper installation of the PLDA QuickUDP license causes the Altera Software Development Kit (SDK) for OpenCL (AOCL) users to encounter a compilation error.
		• In the *Describe the s5_net Hardware to the AOCL* section:
		  1. Elaborated on the calculation of the value for `used_resources` based on the values reported in the Fitter report.
		  2. Added information on the `compile` and `board board_spec.xml` XML elements.
		• Added an *aocl uninstall* subsection under *OpenCL Utilities Implementation* to document the locations of the `uninstall` script for Windows and Linux systems.

July 2014	14.0.0	Initial Release.