AN 780: Compiling and Customizing an Intel® Arria® 10 Custom Platform for OpenCL*

Last updated for Quartus Prime Design Suite: 18.1
AN-780
2018.10.30

Subscribe
Send Feedback

Contents

Compiling and Customizing an Intel® Arria® 10 Custom Platform for OpenCL

2018.10.30

AN-780 Subscribe Send Feedback

This application note describes the procedures and design considerations for modifying the Intel® Arria® 10 GX FPGA Development Kit Reference Platform into your own Custom Platform by using the Intel Software Development Kit (SDK) for OpenCL™[(1)][(2)]

The information and customization techniques described in this document are applicable to any Intel Arria 10 Custom Platform. For reference information, consult the *Intel FPGA SDK for OpenCL Intel Arria 10 GX FPGA Development Kit Reference Platform Porting Guide*. Contact your Intel representative for a copy of this porting guide.

This document also describes how to set up your Custom Platform's operating environment and project with reference to a design example.

Related Information

- **Intel FPGA SDK for OpenCL Getting Started Guide**
- **Intel FPGA SDK for OpenCL Custom Platform Toolkit User Guide**
- **Intel FPGA SDK for OpenCL FPGA Platforms page**

Introduction to Custom Platforms

A Custom Platform provides a representation of the board hardware to the host. This hardware representation enables a host to communicate and offload acceleration tasks to the OpenCL kernel. The platform communicates with the kernel and allows data processing through a system. By customizing existing platforms, you can create tailor-made systems to suit specific architectural requirements.

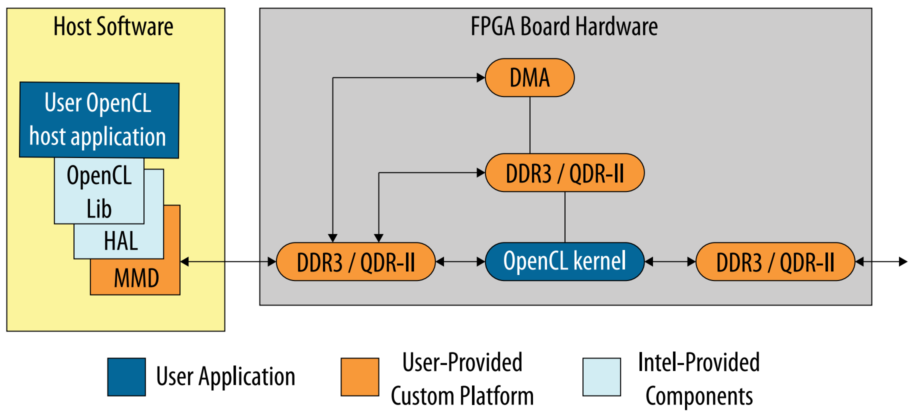
OpenCL System Architecture

An OpenCL System is comprised of a host software and an FPGA board hardware.

(1) 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**.

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

Figure 1-1: Overview of Typical OpenCL Hardware System and Custom Platform



The *Host Software* box in yellow shows the host application running on the host processor. The *FPGA Board Hardware* box in grey depicts the hardware accelerator board that the Custom Platform describes.

On the FPGA board hardware side, the Custom Platform provides the post-place-and-route netlist, which includes all of the hardware necessary to communicate with the host and the memory.

The netlist includes DDR memory interfaces, direct memory access (DMA), and any host interface (for example, PCI Express* (PCIe*)). If there are streaming interfaces to be implemented as channels, these interfaces are also included in the netlist to form an overall communication medium to the host.

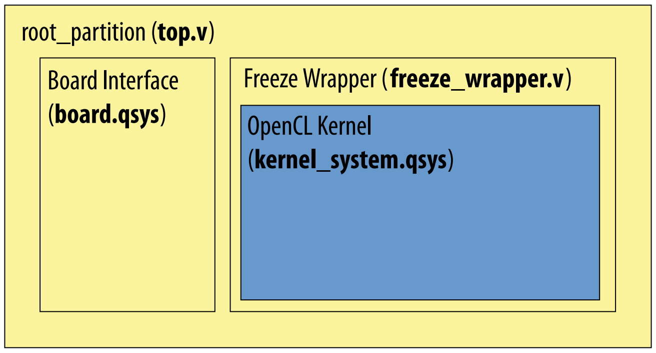
When the Intel FPGA SDK for OpenCL Offline Compiler compiles an OpenCL kernel on the FPGA board hardware side, the offline compiler generates a custom data flow circuit representing your kernel and connects the circuit to the Custom Platform hardware.

On the host side, the Custom Platform needs to provide the memory-mapped device (MMD) layer to allow the OpenCL libraries to communicate with your hardware. The SDK user provides the MMD layer in the form of a library. When compiling the host application, the host application links with both the Intel FPGA OpenCL library and the MMD library to create the host executable. The SDK user can then run the host executable, which launches kernels on the FPGA accelerator board.

Hierarchical Structure of the Intel Arria 10 GX FPGA Development Kit Reference Platform's Hardware

The Intel Arria 10 GX FPGA Development Kit Reference Platform consists of four main blocks, as implemented in a Intel Quartus® Prime project.

Figure 1-2: Hierarchical Structure of the OpenCL Hardware System on an Intel Arria 10 Device


root_partition (**top.v**)
Board Interface (**board.qsys**)
Freeze Wrapper (**freeze_wrapper.v**)
OpenCL Kernel (**kernel_system.qsys**)

Root Partition (top.v)

The `top.v` file describes the I/O ring of the FPGA, which specifies in RTL all of the interfaces to which the FPGA will connect on the PCB.

Board Interface (board.qsys)

The `board.qsys` file is a Platform Designer representation of the Reference Platform. This Platform Designer representation contains IP such as external memory interface (EMIF) to connect to external memory, and hard processor system (HPS) to act as an internal host. When modifying an existing platform, you must update the `board.qsys` file. This file typically contains the logic for the interfaces which are described in the `top.v` file.

Freeze Wrapper (freeze_wrapper.v)

The `freeze_wrapper.v` file is used for Partial Reconfiguration (PR). If your design does not use PR, this file simply acts as a wrapper around the OpenCL kernel. You must change the `freeze_wrapper.v` file if you wish to modify an existing platform that connects to the kernel. Refer to the *Kernel Reprogramming via Partial Reconfiguration* section of the *Intel FPGA SDK for OpenCL Intel Arria 10 GX FPGA Development Kit Reference Platform Porting Guide* for more details on how to implement the freeze wrapper.

OpenCL Kernel (kernel_system.qsys)

When the SDK user compiles an OpenCL kernel, the Intel FPGA SDK for OpenCL Offline Compiler creates the `kernel_system.qsys` file as part of the compilation flow. The SDK user designs this Platform Designer representation of the kernel to optimize its performance when using the FPGA as a hardware acceleration engine.

Intel Quartus Prime Software Revisions Describing the Custom Platform

From the perspective of a Intel Quartus Prime project, there are four software revisions that describe an Intel Arria 10 Custom Platform: `base.qsf`, `top.qsf`, and `flat.qsf`.

Base Revision (base.qsf)

The `base.qsf` revision recompiles and synthesizes the complete project, including the static portion (that is, the Custom Platform) and the kernel, to generate a new `base.qar` file. The `base.qar` file is a Intel

Quartus Prime Database Export File that contains the precompiled netlist of the static region of the design. This revision also contains Logic Lock Plus regions for PR.

Intel Quartus Prime Compilation Stages:

1. Analysis and Synthesis (`top.v`, `board.qsys`, `freeze_wrapper.v`, and `kernel_system.qsys`)
2. Fitter
3. Assembler
4. Timing Analyzer

Top Revision (top.qsf)

The `top.qsf` revision is the same as the `base.qsf` revision. This revision imports the final snapshot (that is, placement and routing contained in the `.qar` file) from the base compilation, and refits the synthesis netlist created from the top_synth revision.

Intel Quartus Prime Compilation Stages:

1. Analysis and Elaboration (`top.v`, `board.qsys`, `freeze_wrapper.v`, and `kernel_system.qsys`)
2. Fitter
3. Assembler
4. Timing Analyzer

Flat Revision (flat.qsf)

The `flat.qsf` file contains all the standard project assignments (for example, pinouts). The other revisions reference the `flat.qsf` file.

Descriptions of the Intel Arria 10 GX FPGA Development Kit Reference Platform Files

Ensure that your Intel Arria 10 Custom Platform includes similar files as those in the Intel Arria 10 GX FPGA Development Kit Reference Platform.

Table 1-1: Main Files Associated with the Intel Arria 10 GX FPGA Development Kit Reference Platform

This table is organized based on the hierarchical structure depicted in **Figure 1-2**.

File	Description
	XML FILES
`board_env.xml`	XML file that describes the Reference Platform to the Intel FPGA SDK for OpenCL.
`board_spec.xml`	XML file that provides the definition of the board hardware interfaces to the SDK.
	ROOT PARTITION
`top.v`	Top-level Verilog Design File for the OpenCL hardware system.
`top.qpf`	Intel Quartus Prime Project File for the OpenCL hardware system.

File	Description
`top.qsf`	Intel Quartus Prime Settings File for the SDK-user compilation flow. This `.qsf` file is used when a pre-placed and pre-routed Reference Platform is imported into the project.
`top.sdc`	Synopsys* Design Constraints File that contains board-specific timing constraints.
`top_post.sdc`	Platform Designer and SDK IP-specific timing constraints.
`flat.qsf`	Intel Quartus Prime Settings File for the flat project revision. This file includes all the common settings, such as pin location assignments, that are used in the other revisions of the project (that is, base, top, and top_synth). The `base.qsf` and `top.qsf` files include, by reference, all the settings in the `flat.qsf` file. The Intel Quartus Prime software compiles the flat revision with minimal location constraints. The flat revision compilation does not generate a `base.qar` file that you can use for future import compilations and does not implement the guaranteed timing flow.
`board.qsys`	Platform Designer system that implements the board interfaces (that is, the static region) of the OpenCL hardware system
`base.qsf`	Intel Quartus Prime Settings File for the base project revision. This file includes, by reference, all the settings in the `flat.qsf` file. Use this revision when porting the Reference Platform to your own Custom Platform. The Intel Quartus Prime Pro Edition software compiles this base project revision from source code.
`base.qar`	Intel Quartus Prime Database Export File that contains the precompiled netlist of the static region of the design. This file is generated by the `scripts/post_flow_pr.tcl` file during base revision compilations and is used during import revision compilations.
`base_compile.tcl`	Tcl script for the base revision compilation flow.
`import_compile.tcl`	Tcl script for the SDK-user compilation flow (that is, import revision compilation).
KERNEL FILES	
`ip/acl_kernel_clk_a10/acl_kernel_clk_a10.qsys`	Platform Designer component that defines the clock generation logic for the kernel clock.
FREEZE WRAPPER FILES	
`ip/freeze_wrapper.v`	Verilog Design File that implements the *freeze* logic placed at inputs and outputs of the PR region.

File	Description
IP FILES USED FOR COMPILATION	
`ip/acl_ddr4_a10/<file_name>`	Directory containing Platform Designer files that implement the DDR4 memory interface. These Platform Designer files are instantiated in `board.qsys`.
`ip/irq_controller/<file_name>`	IP that receives interrupts from the OpenCL kernel system and sends message signaled interrupts (MSI) to the host. Refer to the *Message Signaled Interrupts* section of the *Intel Arria 10 GX FPGA Development Kit Reference Platform Porting Guide* for more information.
SCRIPTS FOR COMPILATION	
`scripts/call_script_as_function.tcl`	Tcl wrapper function for a stand-alone Tcl script to allow the script to be called as a Tcl function.
`scripts/create_fpga_bin_pr.tcl`	Tcl script that generates the `fpga.bin` file. The `fpga.bin` file contains all the necessary files for configuring the FPGA. For more information on the `fpga.bin` file, refer to the *Define the Contents of the fpga.bin File for the Intel Arria 10 GX FPGA Development Kit Reference Platform* section of the *Intel Arria 10 GX FPGA Development Kit Reference Platform* Porting Guide.
`scripts/post_flow_pr.tcl`	Tcl script that implements the guaranteed timing closure flow, as described in the *Guaranteed Timing Closure of the Intel Arria 10 GX FPGA Development Kit Reference Platform Design* section of the *Intel Arria 10 GX FPGA Development Kit Reference Platform Porting Guide*.
`scripts/pre_flow_pr.tcl`	Tcl script that executes before the invocation of the Intel Quartus Prime software compilation. Running the script generates the Platform Designer HDL for `board.qsys` and `kernel_system.qsys`.
PROJECT FILES	
`quartus.ini`	Contains any special Intel Quartus Prime software options that you need when compiling OpenCL kernels for the Reference Platform.
DEVELOPMENT BOARD FILES	
`max5_150.pof`	Programming file for the MAX® V device on the Intel Arria 10 GX FPGA Development Kit that sets the memory reference clock to 150 MHz by default at power-up. You must program the `max5_150.pof` file onto your a10gx or a10gx_es3 board based on the speed at which the DDR4 interface will run.

File	Description
PARTIAL RECONFIGURATION	
`pr_base_id.txt`	Text file containing a unique number for a given base compilation that the runtime uses to determine whether it is safe to use PR programming. The `pr_base_id.txt` file is generated each time you perform a base compilation. The unique number in this file is included in the Intel FPGA SDK for OpenCL Offline Compiler Executable File (`.aocx`) that each import compilation generates.

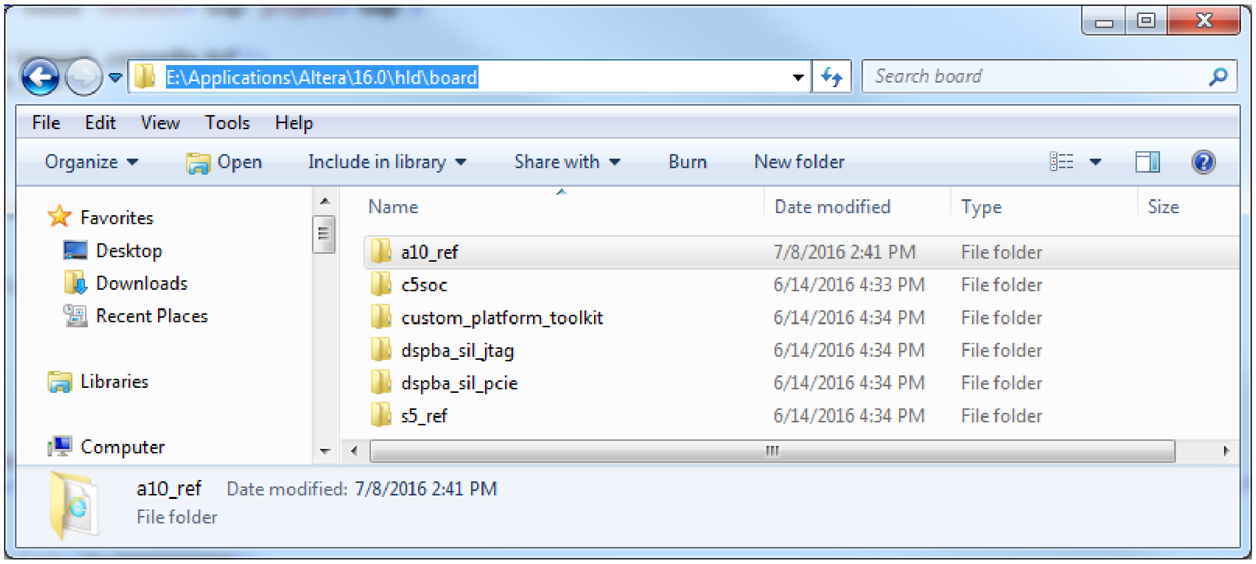
Location of the Intel Arria 10 GX FPGA Development Kit Reference Platform

The Intel FPGA SDK for OpenCL includes a Custom Platform Toolkit that you can use to create your custom platform based on an existing Reference Platform from Intel. The Custom Platform Toolkit includes a set of tools, hardware, templates, and header files to run and test the kernel.

The Custom Platform Toolkit provides the raw hardware, which includes the various FPGA interfaces, example kernels with which to test the board and the interfaces, and the MMD layer header files that include the application programming interface (API) you will need to implement in your Custom Platform.

The Custom Platform Toolkit is available in the *`INTELFPGAOCLSDKROOT`*`/board` directory, where *INTELFPGAOCLSDKROOT* points to the location of the SDK installation.

Figure 1-3: Example Path to the Custom Platform Toolkit Directory



E:\Applications\Altera\16.0\hld\board

Name	Date modified	Type	Size
a10_ref	7/8/2016 2:41 PM	File folder	
c5soc	6/14/2016 4:33 PM	File folder	
custom_platform_toolkit	6/14/2016 4:34 PM	File folder	
dspba_sil_jtag	6/14/2016 4:34 PM	File folder	
dspba_sil_pcie	6/14/2016 4:34 PM	File folder	
s5_ref	6/14/2016 4:34 PM	File folder	

a10_ref Date modified: 7/8/2016 2:41 PM
File folder

If you are using an Intel Preferred Board that is provided by an Intel Preferred Board Partner, download the Custom Platform from your board vendor. You do not need to build your own Custom Platform.

Intel FPGA SDK for OpenCL and User Environment Setup

Install the Intel FPGA SDK for OpenCL before compiling an OpenCL application. SDK installation instructions are available in the *Intel FPGA SDK for OpenCL Getting Started Guide*.

The software installation process installs the SDK application into a directory that you own. The *INTELFPGAOCLSDKROOT* environment variable references the path to the SDK installation directory.

Table 1-2: Structure of the SDK Installation Directory

Windows Folder	Linux Directory	Description
`bin`	`bin`	User commands in the SDK. Include this directory in your *PATH* environment variable setting.
`board`	`board`	The SDK Custom Platform Toolkit and Reference Platforms available with the software. The path to the Custom Platform Toolkit is `INTELFPGAOCLSDKROOT/board/custom_platform_toolkit`.
`ip`	`ip`	Intellectual property (IP) cores used to compile device kernels.
`host`	`host`	Files necessary for compiling and running your host application.
`host\include`	`host/include`	OpenCL Specification version 1.0 header files and software interface files necessary for compiling and linking your host application. The `host/include/CL` subdirectory also includes the C++ header file `cl.hpp`. The file contains an OpenCL version 1.1 C++ wrapper API. These C++ bindings enable a C++ host program to access the OpenCL runtime APIs using native C++ classes and methods. **Important:** The OpenCL version 1.1 C++ bindings are compatible with OpenCL Specification versions 1.0 and 1.1. Add this path to the `include` file search path in your development environment.

Windows Folder	Linux Directory	Description
`host\windows64\lib`	`host/linux64/lib`	OpenCL host runtime libraries that provide the OpenCL platform and runtime APIs. These libraries are necessary for linking your host application. To run an OpenCL application on Linux, include this directory in the *LD_LIBRARY_PATH* environment variable setting.
`host\windows64\bin`	`host/linux64/bin`	Runtime commands and libraries necessary for running your host application, wherever applicable. For 64-bit Windows system, include this directory in your *PATH* environment variable setting. For Windows system, this folder contains runtime libraries. For Linux system, this directory contains platform-specific binary for the SDK utility command.
`share`	`share`	Architecture-independent support files.

After you install the SDK on your machine and you are familiar with the directory structure, set up the environment to run the Intel FPGA SDK for OpenCL Offline Compiler and emulate the design. The following sections describe how to set the user environment for either the Windows or Linux operating systems.

You have the option to set the SDK user environment variables permanently or transiently. The environment variable settings describe the FPGA board and the host runtime to the software.

Specifying the Intel FPGA SDK for OpenCL User Environment Variable Settings on page 1-9
You have the option to set the Windows user environment variables permanently or transiently.

Specifying the Intel FPGA SDK for OpenCL User Environment Variables on Linux on page 1-10
You have the option to set the Linux user environment variables permanently or transiently.

Specifying the Intel FPGA SDK for OpenCL User Environment Variable Settings

You have the option to set the Windows user environment variables permanently or transiently.

- To apply the environment variable settings permanently on your system, set them in the **Environment Variable** dialog box via the **System Properties (Advanced)** tab.

Table 1-3: Intel FPGA SDK for OpenCL Windows User Environment Variable Settings

Environment Variable	Path to Include
PATH	1. `%INTELFPGAOCLSDKROOT%\bin` 2. `%INTELFPGAOCLSDKROOT%\host\windows64\bin` where *INTELFPGAOCLSDKROOT* points to the path of the software installation
AOCL_BOARD_PACKAGE_ROOT	Location of Custom or Reference Platform

- To apply transient environment variable settings, open a command window and run the `%INTELFPGAOCLSDKROOT%\init_opencl.bat` script.

  Example script output:

```
AOCL_BOARD_PACKAGE_ROOT path is not set in environment
Setting to default s5_ref board.
If you want to target another board, do
set AOCL_BOARD_PACKAGE_ROOT=board_pkg_dir and re-run this script
Adding %INTELFPGAOCLSDKROOT%\bin to PATH
Adding %INTELFPGAOCLSDKROOT%\host\windows64\bin to PATH
Adding %AOCL_BOARD_PACKAGE_ROOT%\windows64\bin to PATH
```

Verifying the Windows Intel FPGA SDK for OpenCL User Environment Variable Settings

After setting the Intel FPGA SDK for OpenCL user environment variables, run the SDK to ensure that the software installation is successful.

1. To verify the environment variables in the command shell, type `echo %INTELFPGAOCLSDKROOT%`.

   If the returned path does not point to the location of the SDK installation, edit the *INTELFPGAOCLSDKROOT* setting.
2. At a command prompt, invoke the `aocl version` utility command. An output similar to the one below notifies you of a successful installation:

```
aocl <version>.<build> (Intel FPGA SDK for OpenCL, Version <version> Build
<build>, Copyright (C) <year> Intel Corporation)
```

Specifying the Intel FPGA SDK for OpenCL User Environment Variables on Linux

You have the option to set the Linux user environment variables permanently or transiently.

- To apply permanent environment variable settings, open a shell and then type:

  `export <variable_name>="<variable_setting>":$<variable_name>`

  For example, the command `export PATH="$INTELFPGAOCLSDKROOT/bin":$PATH adds $INTELFPGAOCLSDKROOT/bin` to the list of *PATH* settings.

Table 1-4: Intel FPGA SDK for OpenCL Linux User Environment Variable Settings

Environment Variable	Path to Include
PATH	`$INTELFPGAOCLSDKROOT/bin` where *INTELFPGAOCLSDK-ROOT* points to the path of the software installation
LD_LIBRARY_PATH	`$INTELFPGAOCLSDKROOT/host/linux64/lib` `$AOCL_BOARD_PACKAGE_ROOT/linux64/lib`, where *AOCL_BOARD_PACKAGE_ROOT* points to the path of the Custom or Reference Platform
AOCL_BOARD_PACKAGE_ROOT	Location Custom or Reference Platform

- To apply transient environment variable settings, open a command-line terminal and type:

  `source $INTELFPGAOCLSDKROOT/init_opencl.sh`

  Example script output:

```
AOCL_BOARD_PACKAGE_ROOT path is not set in environment
Setting to default s5_ref board.
If you want to target another board, do
set AOCL_BOARD_PACKAGE_ROOT=board_pkg_dir
Adding $INTELFPGAOCLSDKROOT/bin to PATH
Adding $INTELFPGAOCLSDKROOT/host/linux64/lib to LD_LIBRARY_PATH
Adding $AOCL_BOARD_PACKAGE_ROOT/linux64/lib to LD_LIBRARY_PATH
```

Verifying the Linux Intel FPGA SDK for OpenCL User Environment Variable Settings

After setting the Intel FPGA SDK for OpenCL user environment variables, run the SDK to ensure that the software installation is successful.

1. To verify the environment variables in the command shell, type `env %INTELFPGAOCLSDKROOT%`.

   If the returned path does not point to the location of the Intel FPGA SDK for OpenCL installation, edit the *INTELFPGAOCLSDKROOT* setting.
2. At a command prompt, invoke the `aocl version` utility command. An output similar to the one below notifies you of a successful installation:

```
aocl <version>.<build> (Intel FPGA SDK for OpenCL, Version <version> Build
<build>, Copyright (C) <year> Intel Corporation)
```

Intel Arria 10 Custom Platform Project Setup and Customization Procedure

This section provides an overview of the Intel Arria 10 GX FPGA Development Kit Reference Platform's directory structure and files. It also outlines the procedures for acquiring the Reference Platform, checking the default Reference Platform compilations, and modifying the Reference Platform name.

To acquire the Intel Arria 10 GX FPGA Development Kit Reference Platform, please contact your Intel representative.

Table 1-5: Directory Structure of the Intel Arria 10 GX FPGA Development Kit Reference Platform

Windows Folder	Linux Directory	Description
`board_env.xml`	`board_env.xml`	eXtensible Markup Language (XML) file that describes the Reference Platform to the Intel FPGA SDK for OpenCL.
`hardware`	`hardware`	Contains the Intel Quartus Prime project templates for the supported board variants. Each Reference Platform board variant implements the entire OpenCL hardware system on a given Intel Arria 10 GX FPGA Development Kit. Specify the name of this directory in the `board_env.xml` file. Within this directory, the SDK assumes that any subdirectory containing a `board_spec.xml` file is a board.
`windows64`	`linux64`	Contains the MMD library, kernel mode driver, and executable files of the SDK utilities (that is, install, uninstall, flash, program, diagnose) for your 64-bit operating system.
`source_windows64`	`source`	`source_windows64`: contains source codes for the MMD library and SDK utilities. The MMD library and the SDK utilities are in the `windows64` folder. `source`: contains source codes for the MMD library and SDK utilities. The MMD library and the SDK utilities are in the `linux64` directory.

After developing your Intel Arria 10 Custom Platform, store it in the `*INTELFPGAOCLSDKROOT*/board` directory. The SDK user will then set the *AOCL_BOARD_PACKAGE_ROOT* environment variable to point to the location of the Custom Platform's `board_env.xml` file in order to target an OpenCL kernel compilation to the Intel Arria 10 Custom Platform.

At a minimum, the Custom Platform that you design must include all of the following components:

- The `board_env.xml` file, which contains information about the custom platform.
- The `hardware` directory, which contains all of the hardware design information required for the Intel FPGA SDK for OpenCL Offline Compiler to generate a custom FPGA. The `board_spec.xml` file resides in this directory.
- The OS platform directory (for example, `windows64` or `linux64`).

The OS Platform directory contains the board-specific libraries that must link to the host program, and the executables that run when the SDK user invokes an SDK utility. You must include an OS platform directory in your Custom Platform for each supported host operating system (for example, 64-bit Windows and Linux support).

Each board variant in the Custom Platform consists of a Intel Quartus Prime project and a `board_spec.xml` file that describes the system to the Intel FPGA SDK for OpenCL Offline Compiler. The `board_spec.xml` file also describes the interfaces necessary to connect to the kernel. The Intel FPGA SDK for OpenCL Offline Compiler generates a custom circuit based on the data from the `board_spec.xml` file. Then it incorporates the OpenCL kernel into the Platform Designer system that you create for all non-kernel logic.

Custom Platform Versioning

Any Custom Platform that you create from the Intel Arria 10 GX FPGA Development Kit Reference Platform will only function with the same version of the Intel Quartus Prime Pro Edition software that you used to generate the Custom Platform.

Board XML Files

Your Custom Platform must include the XML files that describe your Custom Platform and each of your hardware systems to the Intel FPGA SDK for OpenCL. You may create these XML files in simple text editors (for example, WordPad for Windows, and vi for Linux). There are two XML files for each Custom Platform: `board_env.xml` and `board_spec.xml` files.

The board_env.xml File

The `board_env.xml` file describes your Custom Platform to the Intel FPGA SDK for OpenCL Offline Compiler. Store this file in the top-level directory of your Custom Platform.

Together with the other contents of the Custom Platform, the `board_env.xml` file sets up the board installation that enables the Intel FPGA SDK for OpenCL Offline Compiler to target a specific accelerator board.

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

Below is an example `board_env.xml` file that describes the Intel Arria 10 GX FPGA Development Kit Reference Platform’s board installation to the Intel FPGA SDK for OpenCL Offline Compiler.

```
<?xml version="1.0"?>
<board_env version="18.1" name="a10_ref_18.1">
  <hardware dir="hardware" default="a10gx_fifo"></hardware>
  <platform name="linux64">
    <mmdlib>%b/linux64/lib/libaltera_a10_ref_mmd.so</mmdlib>
    <linkflags>-L%b/linux64/lib</linkflags>
    <linklibs>-laltera_a10_ref_mmd</linklibs>
    <utilbindir>%b/linux64/libexec</utilbindir>
  </platform>

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

Creating the board_env.xml File

To create a `board_env.xml` file for your Custom Platform, specify the elements and attributes in the `board_env.xml` file template.
Refer to *The board_env.xml File* section for a sample `board_env.xml` file.

For the Intel FPGA SDK for OpenCL Offline Compiler to target a Custom Platform, the Intel FPGA SDK for OpenCL user has to set the environment variable *AOCL_BOARD_PACKAGE_ROOT* to point to the Custom Platform directory in which the `board_env.xml` file resides.

Table 1-6: Specifications of XML Elements and Attributes in the board_env.xml File

Element	Attribute Description
`board_env`	`version`: The Intel FPGA SDK for OpenCL Custom Platform Toolkit release you use to create your Custom Platform. **Attention:** The Custom Platform version must match the SDK version you use to develop the Custom Platform. `name`: Name of the board installation directory containing your Custom Platform.
`hardware`	`dir`: Name of the subdirectory, within the board installation directory, that contains the board variants. `default`: The default board variant that the Intel FPGA SDK for OpenCL Offline Compiler targets when the SDK user does not specify an explicit argument for the `--board` *`<board_name>`* Intel FPGA SDK for OpenCL Offline Compiler option.
`platform`	`name`: Name of the operating system (OS). For a list of supported OS, refer to **Operating System Support** page on the Intel website.
`mmdlib`	A string that specifies the path to the MMD library of your Custom Platform. To load multiple libraries, specify them in an ordered, comma separated list. The host application will load the libraries in the order that they appear in the list.
`linkflags`	A string that specifies the linker flags necessary for linking with the MMD layer available with the board. **Tip:** You can use `%a` to reference the SDK installation directory and `%b` to reference your board installation directory.
`linklibs`	A string that specifies the libraries the SDK must link against to use the MMD layer available with the board. **Note:** Include the `alterahalmmd` library, available with the SDK, in this field because the library is necessary for all devices with an MMD layer.

Element	Attribute Description
`utilbindir`	Directory in which the SDK expects to locate the SDK utility executables (that is, `install`, `uninstall`, `program`, `diagnose`, and `flash`) . **Tip:** You can use `%a` to reference the SDK installation directory and `%b` to reference your board installation directory.

1. Within the `board_env` top-level XML element, under the `name` attribute, specify the name for the board installation.

   Usually, this field matches the directory in which the `board_env.xml` file is located. In this example, this directory is `a10_ref_16.0`.
2. For the `hardware` element, under the `dir` attribute, specify the name of the hardware subdirectory within your board installation directory in which all the board variants are located.

   For Intel-provided Reference Platforms, this subdirectory is named `hardware`.
3. Also for the `hardware` element, under the `default` attribute, specify the default board variant to use.

   This board variant is the board that will be targeted in compilation if a board is not specified. In this example, the default board variant is a10gx_fifo.
4. A `platform` section exists for each host OS that is supported by the board installation. Under the `name` attribute of the `platform` element, specify the name as one of the operating systems that is supported by the SDK. Usually, the name is windows64, linux64, ibm power 64 , or arm32 for Intel SoCs.
5. For each supported OS, specify the following fields:

   - `mmdlib`—a string that lists the paths to the MMD libraries of your Custom Platform.
   - `linkflags`—a string that lists the compiler flags necessary for linking with the MMD driver software layer. The MMD layer is delivered with the board during compilation.
   - `linklibs`—a string that lists the libraries that the compiler must link to in order to use the MMD layer with the board.

     Remember: Include `alterahalmmd` in the `linklibs` field along with your libraries.
   - `utilbindir`—the directory where the SDK expects to find the board utility executables. When the SDK user runs an SDK utility such as `install`, `uninstall`, `program`, `diagnose`, or `flash`, the SDK looks in the `utilbindir` directory to find the actual executable.

Related Information

The board_env.xml File on page 1-13

The board_spec.xml File

The `board_spec.xml` file contains metadata necessary to describe your hardware system to the Intel FPGA SDK for OpenCL. Include the `board_spec.xml` file as part of your custom platform deliverable.

The information conveyed in the XML file includes device resource such as ALMs and DSP blocks available, memory component characteristics and channel information if your device supports streaming applications, and kernel interface information.

Below is an example of the `board_spec.xml` file for the Intel Arria 10 GX FPGA Development Kit Reference Platform.

```
<?xml version="1.0"?>
<board version="18.1" name="a10gx">

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

 <device device_model="10ax115s2f4512sg_dm.xml">
  <used_resources>
   <alms num="36710"/> <!--Total ALMs-ALMs available to kernel_system_inst-->
   <ffs num="146840"/>
   <dsps num="67"/>
   <rams num="224"/>
  </used_resources>
 </device>

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

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

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

</board>
```

Creating the board_spec.xml File

To create a `board_spec.xml` file for your Custom Platform, specify the elements and attributes that describe your board hardware to the Intel FPGA SDK for OpenCL.

A template of the `board_spec.xml` file is available in the *`INTELFPGAOCLSDKROOT`*`/board/custom_platform_toolkit/board_package/hardware/template` directory of the Custom Platform Toolkit.

All of the information in the `board_spec.xml` file should match the actual hardware files.

Table 1-7: Specifications of XML Elements and Attributes in the board_spec.xml File

Element	Attribute Description
`board`	`version`, `name`

Element	Attribute Description
`device`	`device_model`, `used_resources`
`global_mem`	`name`, `max_bandwidth`,`interleaved_bytes`, `config_addr`, `[default]`, `interface`
`host`	`kernel_config`
`[channels]`	`interface`
`interfaces`	`interface`, `kernel_clk_reset`
`compile`	`project`, `revision`, `qsys_file`, `generic_kernel`, `generate_cmd`, `synthesize_cmd`, `auto_migrate`

1. For the top-level `board` element, specify the board name (that is, `name`) and the targeted Intel Quartus Prime software version (that is, `version`).

   In the example `board_spec.xml` file, the board name is a10gx_es3 and the Intel Quartus Prime software version is 16.0.
2. Specify the device model file.

   The Intel FPGA SDK for OpenCL includes device models for the most relevant devices in the *`INTELFPGAOCLSDKROOT`*`/share/models/dm` directory. Identify your FPGA's `.xml` device model file in the `dm` directory and specify its file name in the `device_model` attribute of the `device` element. If the device model file pertaining to your FPGA is not listed in the `dm` directory, create a device model file for your FPGA and then store the file in your Custom Platform subdirectory in which the `board_spec.xml` file resides.
3. For the `used_resources` element, specify information about the FPGA resources that are consumed by the Custom Platform.
   a. Update these values after the Intel Quartus Prime software finishes compiling the Custom Platform.

      The actual amount of resources available to custom kernels will be the total amount of hardware resources minus the resources used by the Custom Platform hardware for components such as the memory controller and the PCIe IP core.
4. If your board contains global memory such as DDR3 or QDR, specify the `global_mem` element and corresponding attributes to describe the characteristics of the memory interface.
5. If your board contains streaming interface such as Ethernet, specify the `channels` element and the corresponding `interface` attribute to identify the I/O channels as either sinks or sources and to describe their characteristics.
6. With respect to information about the host interface to the kernel, specify the `kernel_config` attribute of the `host` element to instruct the compiler at what offset the kernel resides from the perspective of the kernel control register access master on the `kernel_interface` module.

   The offset value should be 0 because the access master does not master anything except for kernels. Leave the `size` attribute at the default value of `0x0100000`.
7. Specify the `interfaces` element and its corresponding attributes to describe the kernel interfaces that connect to the generated OpenCL kernels and control their behaviors.
   a. For each kernel interface, include one of the following interface type: `master`, `irq`, and `streamsource`.

Similar to global memory interfaces, specify the `name`, `port`, and `width` attributes. For the `streamsource` interface type, specify the `clock` attribute with the name of the clock that is used for the snoop stream. Usually, this clock is the kernel clock.

Customization Flow

The Intel Arria 10 Custom Platform customization flow involves compiling an existing Reference Platform and the modifying the Reference Platform according to your specifications.

The customization flow has five steps:

1. Check default Reference or Custom Platform with a standard OpenCL kernel.
2. Set up project for customization.
3. Add components to Custom Platform, including kernel and I/O ring.
4. Check compilation results and debug.
5. Update Custom Platform with modified files.

The following sections describe how to compile the original Reference Platform with and without platform regeneration and also without any customization. Performing these compilations checks the Intel FPGA SDK for OpenCL environment and project setup.

1. **Compiling a Kernel without Regenerating the Custom Platform** on page 1-18
   Step 1 in the Intel Arria 10 Custom Platform customization flow is to verify the functionality of the existing Reference Platform by compiling it with a simple OpenCL kernel but without regenerating the platform.
2. **Preparing an Existing Custom Platform for Customization** on page 1-19
   Step 2 in the Intel Arria 10 Custom Platform Customization Flow is to prepare the Custom Platform for customization.

Compiling a Kernel without Regenerating the Custom Platform

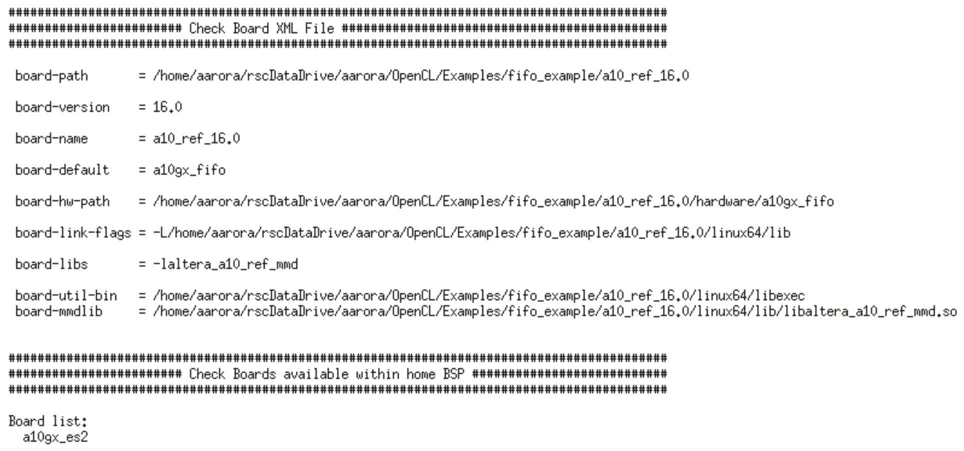
Step 1 in the Intel Arria 10 Custom Platform customization flow is to verify the functionality of the existing Reference Platform by compiling it with a simple OpenCL kernel but without regenerating the platform. Compiling the existing platform checks the platform's setup and verifies that the Intel FPGA SDK for OpenCL Offline Compiler works as expected. A main advantage to compiling a kernel without regenerating the Reference Platform is that it preserves placement and routing as well as timing, which saves compilation time.

Intel assumes that you have set up your Windows or Linux environment correctly to run the Intel FPGA SDK for OpenCL Offline Compiler.

1. Obtain the Intel Arria 10 GX FPGA Development Kit Reference Platform (for example, `a10_ref_18.1_b222.zip`) from your Intel representative.
2. Unpack the Reference Platform and store it in a directory named <*`your_custom_platform`*>. For this example, <*`your_custom_platform`*> is `a10gx_ref_18.1`.
3. Choose one of the board variants in the `a10gx_ref_18.1/hardware` directory as the basis of your design (for example, a10gx).
4. Open the `a10gx_ref_18.1/board_env.xml` file in a text editor and perform the following tasks:
   a. Change the `board name` setting from `a10_ref` to `a10gx_ref_18.1`.
   b. Verify that the `board default` setting is `a10gx_es3`.
   c. Save and then close the `board_env.xml` file.

5. Open the `a10gx_ref_18.1/hardware/a10gx/board_spec.xml` file in a text editor and perform the following tasks:
   a. Verify that the `board name` setting is `a10gx_es3`.
   b. Save and then close the `board_spec.xml` file.
6. To set the *AOCL_BOARD_PACKAGE_ROOT* environment variable, at a command prompt, invoke the `set AOCL_BOARD_PACKAGE_ROOT=<`*`path to a10gx_ref_18.1`*`>` command, where `a10gx_ref_18.1` is the new design directory.
7. To test the environment, first invoke the `aocl board-xml-test` command to read the `board_env.xml` file and display the Custom Platform information on-screen.
8. Invoke the `aoc --list-boards` command to display the board variants that are available in the a10gx_ref_18.1 Custom Platform.

Figure 1-4: Sample Output from the aocl board-xml-test and aoc --list-boards Commands



```
###########################################################################################
######################## Check Board XML File #############################################
###########################################################################################

 board-path       = /home/aarora/rscDataDrive/aarora/OpenCL/Examples/fifo_example/a10_ref_16.0

 board-version    = 16.0

 board-name       = a10_ref_16.0

 board-default    = a10gx_fifo

 board-hw-path    = /home/aarora/rscDataDrive/aarora/OpenCL/Examples/fifo_example/a10_ref_16.0/hardware/a10gx_fifo

 board-link-flags = -L/home/aarora/rscDataDrive/aarora/OpenCL/Examples/fifo_example/a10_ref_16.0/linux64/lib

 board-libs       = -laltera_a10_ref_mmd

 board-util-bin   = /home/aarora/rscDataDrive/aarora/OpenCL/Examples/fifo_example/a10_ref_16.0/linux64/libexec
 board-mmdlib     = /home/aarora/rscDataDrive/aarora/OpenCL/Examples/fifo_example/a10_ref_16.0/linux64/lib/libaltera_a10_ref_mmd.so

###########################################################################################
######################## Check Boards available within home BSP ###########################
###########################################################################################

Board list:
  a10gx_es2
```

9. To compile an OpenCL kernel without regenerating a10gx_ref_18.1, perform the following tasks:
   a. Download the Vector Addition design example from the **OpenCL Design Examples** page.
   b. Copy the `vector_add.cl` file to the project directory `a10gx_ref_18.1`.
   c. Invoke the `aoc vector_add.cl -v --no-interleaving default` command.
   d. After the compilation is completed, you can review the resulting files in the `vector_add` directory within your working directory.
10. Refer to the *Analyzing the Results from Compilation* section to check the Fitter, Timing Analyzer reports and placement in the Floorplanner.

Related Information

Analyzing the Results from Compilation on page 1-30

Preparing an Existing Custom Platform for Customization

Step 2 in the Intel Arria 10 Custom Platform Customization Flow is to prepare the Custom Platform for customization. Before customizing your Custom Platform, make a copy of the existing platform to keep the original platform settings intact.

Intel assumes that you have completed the steps outlined in the *Compiling a Kernel (vector_add.cl) without Regenerating the Custom Platform* section.

To set up the project for customization, perform the following tasks:

1. Make a copy of the `a10gx_ref_18.1/hardware/a10gx` directory.
2. Rename the copied directory from `a10gx` to `a10gx_fifo`. This new directory will contain any new files and changes resulted from the customization.
3. Open the `a10gx_ref_18.1/board_env.xml` file in a text editor and perform the following tasks:
   a. Change the `board default` setting from `a10gx_es3` to `a10gx_fifo`.
   b. Save and then close the `board_env.xml` file.
4. Open the `a10gx_ref_18.1/hardware/a10gx_fifo/board_spec.xml` file a text editor and perform the following tasks:
   a. Change the `board name` setting from `a10gx_es3` to `a10gx_fifo`.
   b. Save and then close the `board_spec.xml` file.
5. Navigate to the `a10gx_ref_18.1` project directory and invoke the `aoc vector_add.cl -v --no-interleaving default` command.
6. After the compilation is completed, you can review the resulting files in the `vector_add` directory within your working directory.
7. Refer to the *Analyzing the Results from Compilation* section to check the Fitter, Timing Analyzer reports and placement in the Floorplanner.

Related Information

- **Analyzing the Results from Compilation** on page 1-30
- **Compiling a Kernel without Regenerating the Custom Platform** on page 1-18

Intel Arria 10 Custom Platform Customization Example

This section describes the process of modifying the a10gx_ref_18.1 Custom Platform that you prepared for customization.

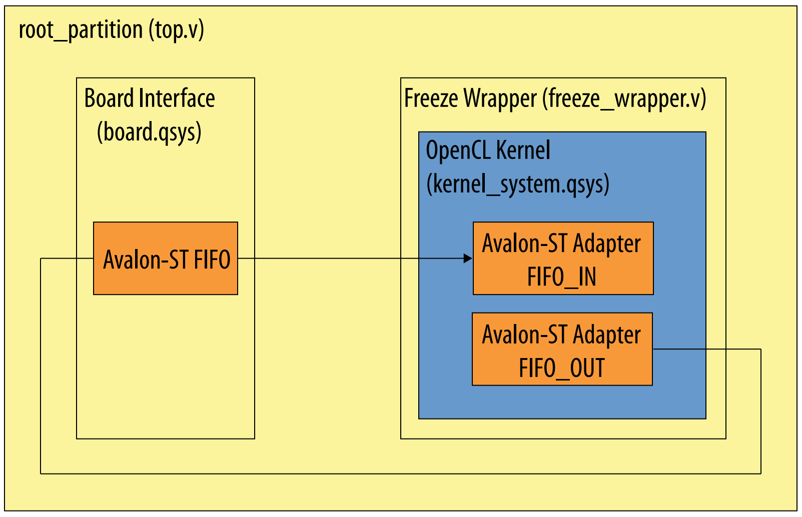
Prerequisites for customization:

- You have prepared the original Custom Platform for customization, as outlined in the *Preparing an Existing Custom Platform for Customization* section.
- You have a `<your kernel file name>` directory within the Custom Platform directory. The `vector_add` directory mentioned herein was created after you compiled the a10gx_ref_18.1 Custom Platform for the first time using the Intel FPGA SDK for OpenCL Offline Compiler, as described in the *Compiling a Kernel without Regenerating the Custom Platform* and *Preparing an Existing Custom Platform for Customization* sections.

The following information pertains to Steps 3 to 5 of the Customization Flow.

The figure below illustrates the customized a10gx_ref_18.1 Custom Platform's hardware. Refer to **Figure 1-2** for an illustration of the original Intel Arria 10 GX FPGA Development Kit Reference Platform's architecture. Customization (shown in orange) includes adding an Avalon® Streaming (Avalon-ST) Single Clock FIFO component to the `board.qsys` file and then connecting it to the kernel via the freeze wrapper. Because the customization creates a streaming interface, you must alter the `board_spec.xml` file and change the channel property.

Figure 1-5: Architectural Representation of a Customized Custom Platform Based on the Intel Arria 10 GX FPGA Development Kit Reference Platform


root_partition (top.v)
Board Interface (board.qsys)
Avalon-ST FIFO
Freeze Wrapper (freeze_wrapper.v)
OpenCL Kernel (kernel_system.qsys)
Avalon-ST Adapter FIFO_IN
Avalon-ST Adapter FIFO_OUT

Customizing the a10gx_ref_18.1 Custom Platform involves the following tasks:

1. **Modifying the board.qsys File in the Custom Platform** on page 1-21
   Modify the `board.qsys` file by adding an Avalon-ST Single Clock FIFO component.
2. **Modifying the Kernel (freeze_wrapper.v and board_spec.xml)** on page 1-24
   Modify the `freeze_wrapper.v` and `board_spec.xml` files by adding an Avalon-ST Adapter component.
3. **Updating the Top-Level I/O Ring with the Modified board.qsys and freeze_wrapper.v Files** on page 1-28
   Add ports and signals to the board and freeze_wrapper instances in the `top.v` file.
4. **Updating the Original Custom Platform Directory with the New Custom Platform Modifications** on page 1-29
   The final step to customizing your Custom Platform is to copy all modified files back into the original Custom Platform directory (that is, the `a10gx_ref_18.1/hardware/a10gx_fifo` directory).
5. **Compilation Log Files** on page 1-30
   The compilation log files record verbose information while the software tools synthesize and compile the Custom Platform and the kernel.
6. **Analyzing the Results from Compilation** on page 1-30
   After the full compilation flow has completed, check the results in the Intel Quartus Prime Pro Edition software GUI.

Related Information

- **Preparing an Existing Custom Platform for Customization** on page 1-19
- **Compiling a Kernel without Regenerating the Custom Platform** on page 1-18
- **Customization Flow** on page 1-18

Modifying the board.qsys File in the Custom Platform

Modify the `board.qsys` file by adding an Avalon-ST Single Clock FIFO component.

1. **Opening an Existing Intel Quartus Prime Project and the board.qsys Platform Designer System Design** on page 1-22
   Open the `board.qsys` file in the Platform Designer system integration tool.
2. **Adding the Avalon-ST Single Clock FIFO Component into the Platform Designer System** on page 1-22
   Add an Avalon-ST Single Clock FIFO component to the `board.qsys` Platform Designer system.
3. **Connecting the Avalon-ST Single Clock FIFO Component's Exported Signals in the Top-Level Platform Designer System** on page 1-24
   After adding the Avalon-ST Single Clock FIFO component to `board.qsys`, connect the component's exported signals by generating HDL.

Opening an Existing Intel Quartus Prime Project and the board.qsys Platform Designer System Design

Open the `board.qsys` file in the Platform Designer system integration tool.

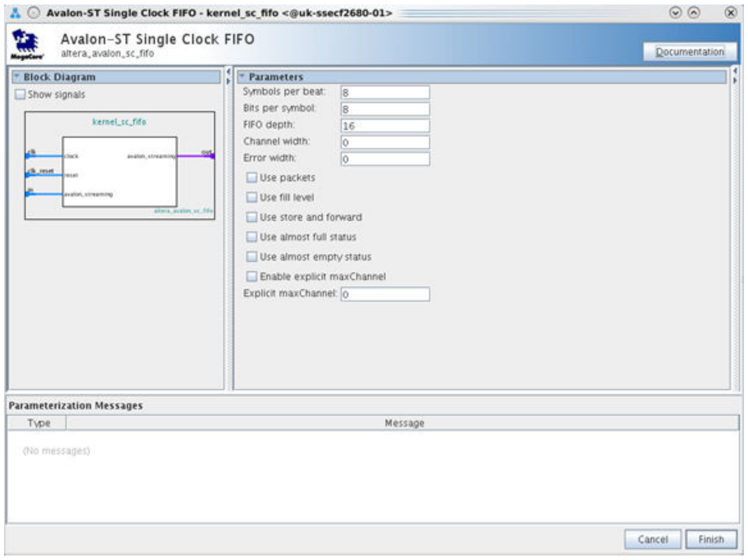
1. Open the Intel Quartus Prime Pro Edition software.
2. Open the Intel Quartus Prime project file `a10gx_ref_18.1/vector_add/top.qpf`.
3. Open Platform Designer from the **Tools** menu or the toolbar.
4. Open the system named `board.qsys`.

Adding the Avalon-ST Single Clock FIFO Component into the Platform Designer System

Add an Avalon-ST Single Clock FIFO component to the `board.qsys` Platform Designer system.

1. With `board.qsys` opened in Platform Designer, add an Avalon-ST Single Clock FIFO component from the IP catalog. Open the parameter editor and specify the following configuration settings:

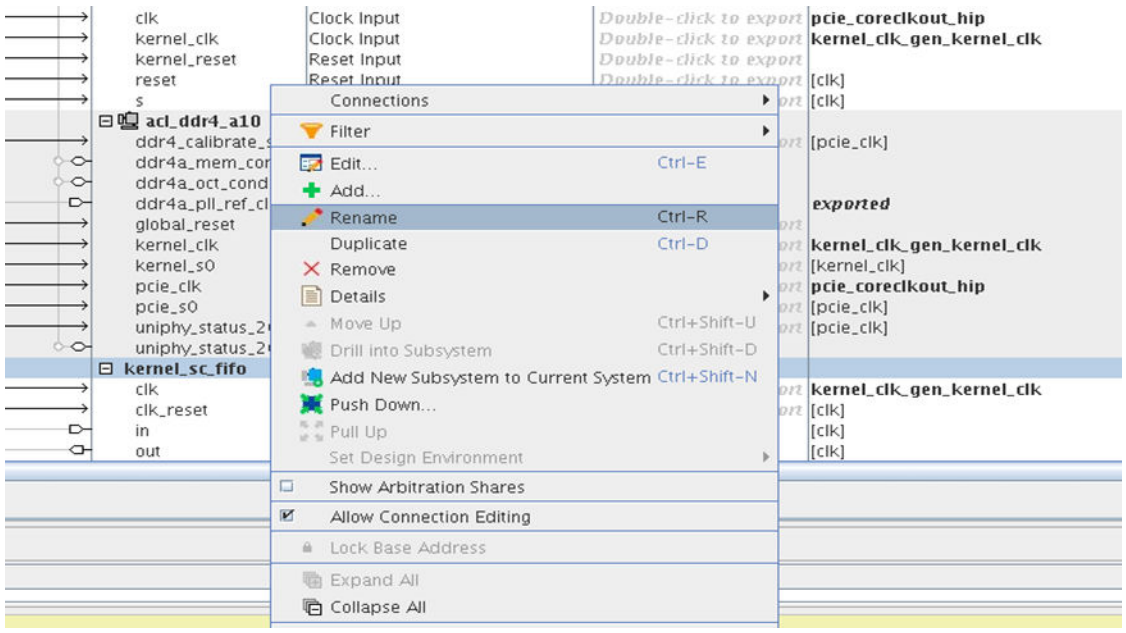
Figure 1-6: Configuration Settings of the Avalon-ST Single Clock FIFO Component



Symbols per beat: 8
Bits per symbol: 8
FIFO depth: 16
Channel width: 0
Error width: 0
☐ Use packets
☐ Use fill level
☐ Use store and forward
☐ Use almost full status
☐ Use almost empty status
☐ Enable explicit maxChannel
Explicit maxChannel: 0

2. Click **Finish**.
3. Right-click the **board_sc_fifo** component at the bottom of the **System Contents** tab and select **Rename**.

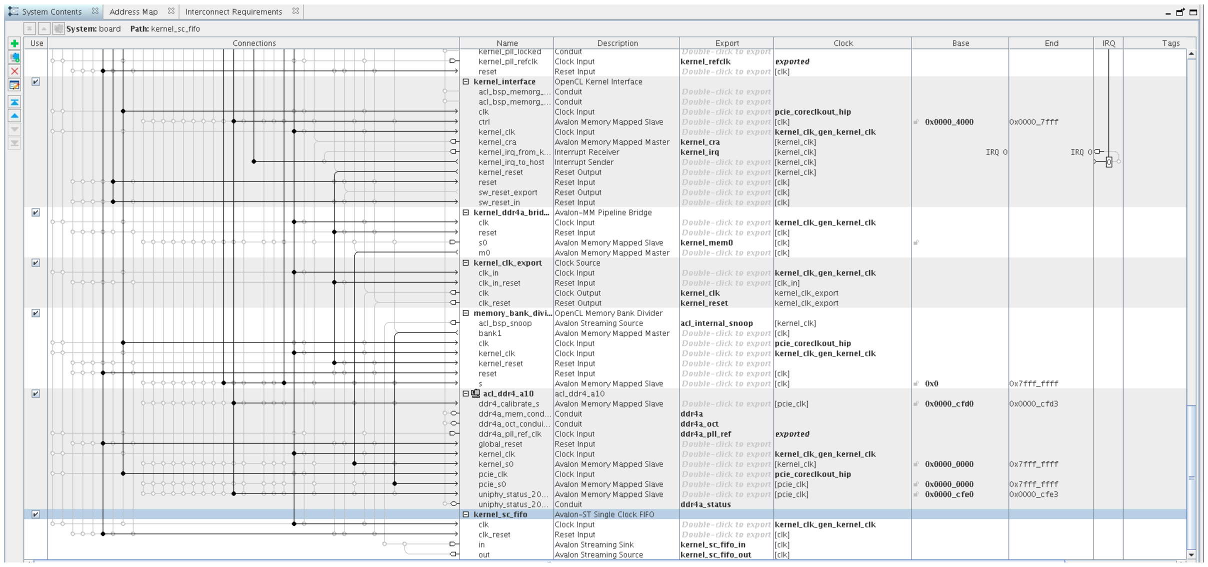
Figure 1-7: Renaming the board_sc_fifo component to kernel_sc_fifo



4. Change the component's name to kernel_sc_fifo.
5. Connect the kernel_sc_fifo component's **clock** input interface to the kernel clock by performing the following tasks:
    a. Right-click the **clock** interface of the kernel_sc_fifo component.
    b. Click **Connections** > **kernel_sc_fifo.clock** and then select **kernel_clk_gen.kernel_clk**.
6. Connect the **reset** interface of the kernel_sc_fifo component to the PCIe reset.
    a. Right-click the **reset** interface of the kernel_sc_fifo component.
    b. Click **Connections** > **kernel_sc_fifo.reset** and then select **kernel_interface.kernel_reset**.
7. Export the **in** and **out** interfaces of the Avalon-ST Single Clock FIFO component by double-clicking the **Export** column in the **System Contents** tab.

    The in and out ports are named `kernel_sc_fifo_in` and `kernel_sc_fifo_out`, respectively.

Figure 1-8: In and Out Ports of the Avalon-ST Single Clock FIFO Component



Name	Description	Export	Clock	Base	End
kernel_pll_locked	Conduit	Double-click to export			
kernel_pll_refclk	Clock Input	kernel_refclk	exported		
reset	Reset Input	Double-click to export	[clk]		
⊟ kernel_interface	OpenCL Kernel Interface				
acl_bsp_memorg_...	Conduit	Double-click to export			
acl_bsp_memorg_...	Conduit	Double-click to export			
clk	Clock Input	Double-click to export	pcie_coreclkout_hip		
ctrl	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_4000	0x0000_7fff
kernel_clk	Clock Input	Double-click to export	kernel_clk_gen_kernel_clk		
kernel_cra	Avalon Memory Mapped Master	kernel_cra	[kernel_clk]		
kernel_irq_from_k...	Interrupt Receiver	kernel_irq	[kernel_clk]	IRQ 0	IRQ 0
kernel_irq_to_host	Interrupt Sender	Double-click to export	[kernel_clk]		
kernel_reset	Reset Output	Double-click to export	[kernel_clk]		
reset	Reset Input	Double-click to export	[clk]		
sw_reset_export	Reset Output	Double-click to export	[clk]		
sw_reset_in	Reset Input	Double-click to export	[clk]		
⊟ kernel_ddr4a_brid...	Avalon-MM Pipeline Bridge				
clk	Clock Input	Double-click to export	kernel_clk_gen_kernel_clk		
reset	Reset Input	Double-click to export	[clk]		
s0	Avalon Memory Mapped Slave	kernel_mem0	[clk]		
m0	Avalon Memory Mapped Master	Double-click to export	[clk]		
⊟ kernel_clk_export	Clock Source				
clk_in	Clock Input	Double-click to export	kernel_clk_gen_kernel_clk		
clk_in_reset	Reset Input	Double-click to export	[clk_in]		
clk	Clock Output	kernel_clk	kernel_clk_export		
clk_reset	Reset Output	kernel_reset	kernel_clk_export		
⊟ memory_bank_divi...	OpenCL Memory Bank Divider				
acl_bsp_snoop	Avalon Streaming Source	acl_internal_snoop	[kernel_clk]		
bank1	Avalon Memory Mapped Master	Double-click to export	[clk]		
clk	Clock Input	Double-click to export	pcie_coreclkout_hip		
kernel_clk	Clock Input	Double-click to export	kernel_clk_gen_kernel_clk		
kernel_reset	Reset Input	Double-click to export			
reset	Reset Input	Double-click to export	[clk]		
s	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0	0x7fff_ffff
⊟ acl_ddr4_a10	acl_ddr4_a10				
ddr4_calibrate_s	Avalon Memory Mapped Slave	Double-click to export	[pcie_clk]	0x0000_cfd0	0x0000_cfd3
ddr4a_mem_cond...	Conduit	ddr4a			
ddr4a_oct_condui...	Conduit	ddr4a_oct			
ddr4a_pll_ref_clk	Clock Input	ddr4a_pll_ref	exported		
global_reset	Reset Input	Double-click to export			
kernel_clk	Clock Input	Double-click to export	kernel_clk_gen_kernel_clk		
kernel_s0	Avalon Memory Mapped Slave	Double-click to export	[kernel_clk]	0x0000_0000	0x7fff_ffff
pcie_clk	Clock Input	Double-click to export	pcie_coreclkout_hip		
pcie_s0	Avalon Memory Mapped Slave	Double-click to export	[pcie_clk]	0x0000_0000	0x7fff_ffff
uniphy_status_20...	Avalon Memory Mapped Slave	Double-click to export	[pcie_clk]	0x0000_cfe0	0x0000_cfe3
uniphy_status_20...	Conduit	ddr4a_status			
⊟ kernel_sc_fifo	Avalon-ST Single Clock FIFO				
clk	Clock Input	Double-click to export	kernel_clk_gen_kernel_clk		
clk_reset	Reset Input	Double-click to export	[clk]		
in	Avalon Streaming Sink	kernel_sc_fifo_in	[clk]		
out	Avalon Streaming Source	kernel_sc_fifo_out	[clk]		

8. Verify that there are no errors in the message window.

Connecting the Avalon-ST Single Clock FIFO Component's Exported Signals in the Top-Level Platform Designer System

After adding the Avalon-ST Single Clock FIFO component to `board.qsys`, connect the component's exported signals by generating HDL.

1. Save the `board.qsys` system.
2. Click **Close**.
3. From the **Generate** menu in Platform Designer, select **Generate HDL**. Alternatively, click **Generate HDL** in the lower right corner of the Platform Designer window.
4. Click **Generate**.
5. Click **Close** when HDL generation is completed. Ignore any warnings that might appear.

Modifying the Kernel (freeze_wrapper.v and board_spec.xml)

Modify the `freeze_wrapper.v` and `board_spec.xml` files by adding an Avalon-ST Adapter component.

1. **Opening an Existing Intel Quartus Prime Project and the kernel_system.qsys Platform Designer System Design** on page 1-25
   Open the `kernel_system.qsys` file in the Platform Designer system integration tool.
2. **Adding an Avalon-ST Adapter Component into the Platform Designer System** on page 1-25
   Add an Avalon-ST Adapter component to the `kernel_system.qsys` Platform Designer system.
3. **Connecting the Avalon-ST Adapter Component's Exported Signals in the Top-Level Platform Designer System** on page 1-27
   After adding the Avalon-ST Adapter component to `kernel_system.qsys`, connect the component's exported signals by generating HDL.
4. **Modifying the board_spec.xml File** on page 1-27
   Add streaming FIFO channel information into the `board_spec.xml` file.

5. **Modifying the freeze_wrapper.v File** on page 1-28
   Create ports on the freeze_wrapper and kernel_system modules for the Avalon-ST Single Clock FIFO component.

Opening an Existing Intel Quartus Prime Project and the kernel_system.qsys Platform Designer System Design

Open the `kernel_system.qsys` file in the Platform Designer system integration tool.

1. Open the Intel Quartus Prime Pro Edition software.
2. Open the Intel Quartus Prime project file `a10gx_ref_18.1/vector_add/top.qpf`.
3. Open Platform Designer tool from the **Tools** menu or the toolbar.
4. Open the system named `kernel_system.qsys`.

   The Intel FPGA SDK for OpenCL Offline Compiler created the `kernel_system.qsys` file after it finished compiling the Custom Platform project for the first time, as described in the *Compiling a Kernel without Regenerating the Custom Platform* section.

Related Information

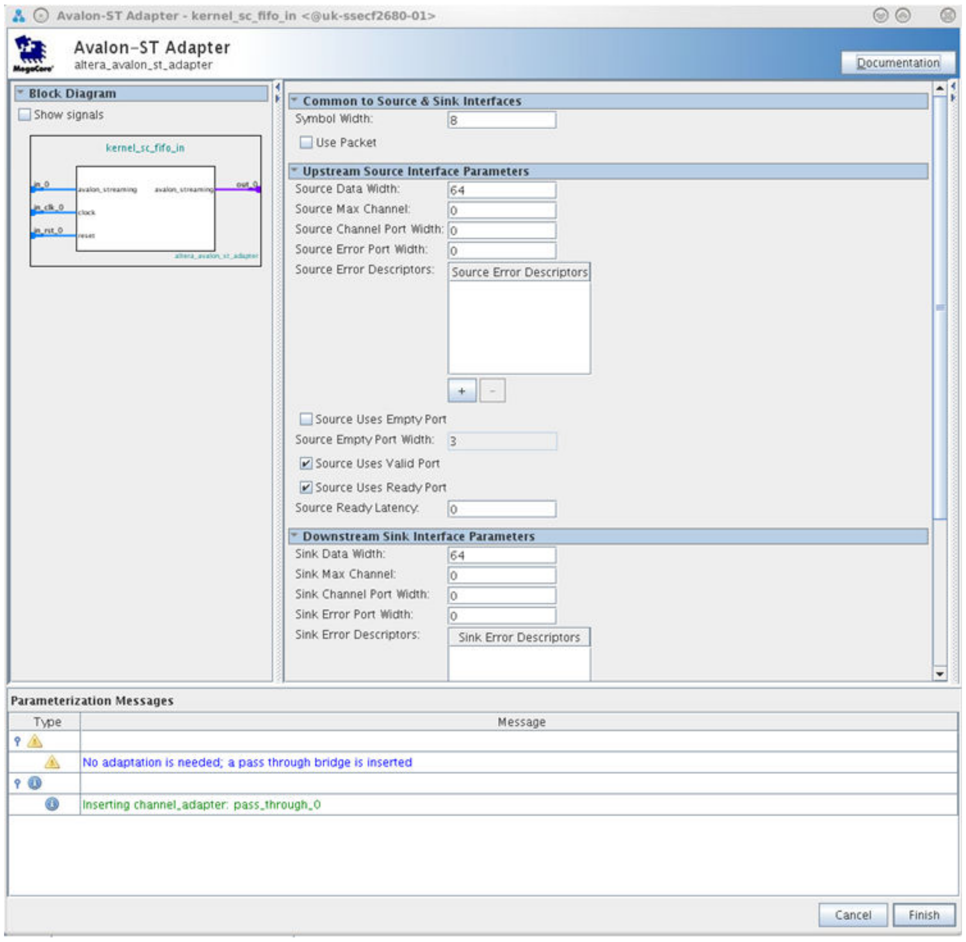
Compiling a Kernel without Regenerating the Custom Platform on page 1-18

Adding an Avalon-ST Adapter Component into the Platform Designer System

Add an Avalon-ST Adapter component to the `kernel_system.qsys` Platform Designer system.

1. With `kernel_system.qsys` opened in Platform Designer, add an Avalon-ST Adapter component from the IP catalog. Open the parameter editor and specify the following configuration settings:

Figure 1-9: Configuration Settings of the Avalon-ST Adapter Component



Common to Source & Sink Interfaces
Symbol Width: 8
[ ] Use Packet

Upstream Source Interface Parameters
Source Data Width: 64
Source Max Channel: 0
Source Channel Port Width: 0
Source Error Port Width: 0
Source Error Descriptors: Source Error Descriptors
[ ] Source Uses Empty Port
Source Empty Port Width: 3
[✓] Source Uses Valid Port
[✓] Source Uses Ready Port
Source Ready Latency: 0

Downstream Sink Interface Parameters
Sink Data Width: 64
Sink Max Channel: 0
Sink Channel Port Width: 0
Sink Error Port Width: 0
Sink Error Descriptors: Sink Error Descriptors

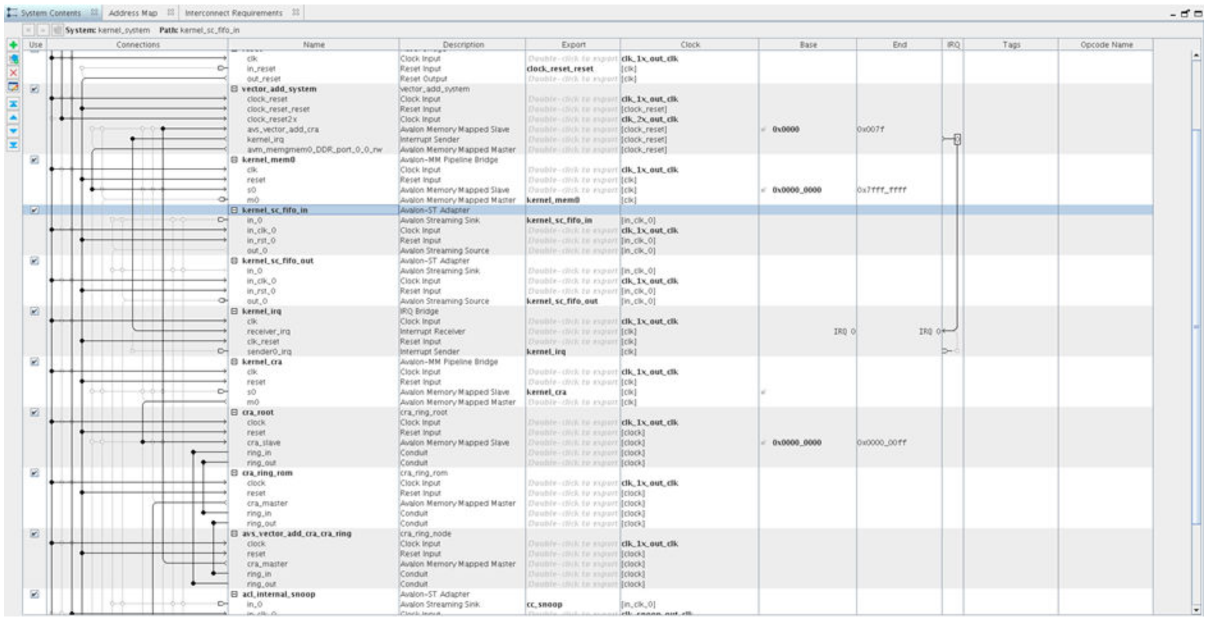
Parameterization Messages

Type	Message
⚠	
⚠	No adaptation is needed; a pass through bridge is inserted
ⓘ	
ⓘ	Inserting channel_adapter: pass_through_0

2. Click **Finish**.
3. Right-click the **kernel_system_st_adapter** component at the bottom of the **System Contents** tab and select **Rename**.
4. Change the component's name to kernel_sc_fifo_in.
5. Repeat step 1 to step 4 and rename this component's name to kernel_sc_fifo_out.
6. Connect the kernel_sc_fifo_in component's **clock** input interface to the kernel clock by performing the following tasks:
   a. Right-click the **clock** interface of the kernel_sc_fifo_in component.
   b. Click **Connections** > **kernel_sc_fifo_in.in_clk_0** and then select **kernel_clk_gen.clk_1x.out_clk**.
7. Connect the **reset** interface of the kernel_sc_fifo_in component to the PCIe reset.
   a. Right-click the **reset** interface of the kernel_sc_fifo_in component.
   b. Click **Connections** > **kernel_sc_fifo_in.in_rst_0** and then select **reset.out_reset**.
8. Repeat step 6 and step 7 for the kernel_sc_fifo_out instance.
9. Export the **in** and **out** interfaces of the Avalon-ST Adapter component by double-clicking the **Export** column in the **System Contents** tab.

   The in and out ports are named `kernel_sc_fifo_in` and `kernel_sc_fifo_out`, respectively.

Figure 1-10: In and Out Ports of the Avalon-ST Adapter Component


System Contents
Address Map
Interconnect Requirements
System: kernel_system Path: kernel_sc_fifo_in

Name	Description	Export	Clock	Base	End	IRQ
clk	Clock Input	Double-click to export	clk_1x_out_clk			
in_reset	Reset Input	clock_reset_reset	[clk]			
out_reset	Reset Output	Double-click to export	[clk]			
⊟ vector_add_system	vector_add_system					
clock_reset	Clock Input	Double-click to export	clk_1x_out_clk			
clock_reset_reset	Reset Input	Double-click to export	[clock_reset]			
clock_reset2x	Clock Input	Double-click to export	clk_2x_out_clk			
avs_vector_add_cra	Avalon Memory Mapped Slave	Double-click to export	[clock_reset]	0x0000	0x007f	
kernel_irq	Interrupt Sender	Double-click to export	[clock_reset]			
avm_memgmem0_DDR_port_0_0_rw	Avalon Memory Mapped Master	Double-click to export	[clock_reset]			
⊟ kernel_mem0	Avalon-MM Pipeline Bridge					
clk	Clock Input	Double-click to export	clk_1x_out_clk			
reset	Reset Input	Double-click to export	[clk]			
s0	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0000	0x7fff_ffff	
m0	Avalon Memory Mapped Master	kernel_mem0	[clk]			
⊟ kernel_sc_fifo_in	Avalon-ST Adapter					
in_0	Avalon Streaming Sink	kernel_sc_fifo_in	[in_clk_0]			
in_clk_0	Clock Input	Double-click to export	clk_1x_out_clk			
in_rst_0	Reset Input	Double-click to export	[in_clk_0]			
out_0	Avalon Streaming Source	Double-click to export	[in_clk_0]			
⊟ kernel_sc_fifo_out	Avalon-ST Adapter					
in_0	Avalon Streaming Sink	Double-click to export	[in_clk_0]			
in_clk_0	Clock Input	Double-click to export	clk_1x_out_clk			
in_rst_0	Reset Input	Double-click to export	[in_clk_0]			
out_0	Avalon Streaming Source	kernel_sc_fifo_out	[in_clk_0]			
⊟ kernel_irq	IRQ Bridge					
clk	Clock Input	Double-click to export	clk_1x_out_clk			
receiver_irq	Interrupt Receiver	Double-click to export	[clk]	IRQ 0	IRQ 0	
clk_reset	Reset Input	Double-click to export	[clk]			
sender0_irq	Interrupt Sender	kernel_irq	[clk]			
⊟ kernel_cra	Avalon-MM Pipeline Bridge					
clk	Clock Input	Double-click to export	clk_1x_out_clk			
reset	Reset Input	Double-click to export	[clk]			
s0	Avalon Memory Mapped Slave	kernel_cra	[clk]			
m0	Avalon Memory Mapped Master	Double-click to export	[clk]			
⊟ cra_root	cra_ring_root					
clock	Clock Input	Double-click to export	clk_1x_out_clk			
reset	Reset Input	Double-click to export	[clock]			
cra_slave	Avalon Memory Mapped Slave	Double-click to export	[clock]	0x0000_0000	0x0000_00ff	
ring_in	Conduit	Double-click to export	[clock]			
ring_out	Conduit	Double-click to export	[clock]			
⊟ cra_ring_rom	cra_ring_rom					
clock	Clock Input	Double-click to export	clk_1x_out_clk			
reset	Reset Input	Double-click to export	[clock]			
cra_master	Avalon Memory Mapped Master	Double-click to export	[clock]			
ring_in	Conduit	Double-click to export	[clock]			
ring_out	Conduit	Double-click to export	[clock]			
⊟ avs_vector_add_cra_cra_ring	cra_ring_node					
clock	Clock Input	Double-click to export	clk_1x_out_clk			
reset	Reset Input	Double-click to export	[clock]			
cra_master	Avalon Memory Mapped Master	Double-click to export	[clock]			
ring_in	Conduit	Double-click to export	[clock]			
ring_out	Conduit	Double-click to export	[clock]			
⊟ acl_internal_snoop	Avalon-ST Adapter					
in_0	Avalon Streaming Sink	cc_snoop	[in_clk_0]			

10. Verify that there are no errors in the message window.

Connecting the Avalon-ST Adapter Component's Exported Signals in the Top-Level Platform Designer System

After adding the Avalon-ST Adapter component to `kernel_system.qsys`, connect the component's exported signals by generating HDL.

1. Save the `kernel_system.qsys` system.
2. Click **Close**.
3. From the **Generate** menu in Platform Designer, select **Generate HDL**. Alternatively, click **Generate HDL** in the lower right corner of the Platform Designer window.
4. Click **Generate**.
5. Click **Close** when HDL generation is completed. Ignore any warnings that might appear.

Modifying the board_spec.xml File

Add streaming FIFO channel information into the `board_spec.xml` file.

1. Open the `board_spec.xml` file in the `a10gx_ref_18.1/hardware/a10gx_fifo` directory.
2. Add the following streaming FIFO channel information into the `board_spec.xml` file:

```
<channels>
   <interface name="board" width="64" type="streamsource"
port="kernel_sc_fifo_out" chan_id="kernel_sc_fifo_in"/>
   <interface name="board" width="64" type="streamsink" port="kernel_sc_fifo_in"
chan_id="kernel_sc_fifo_out"/>
<channels>
```

3. Save the `board_spec.xml` file.

Modifying the freeze_wrapper.v File

Create ports on the freeze_wrapper and kernel_system modules for the Avalon-ST Single Clock FIFO component.

1. Open the `ip/freeze_wrapper.v` file in the `vector_add` directory.
2. In the `freeze_wrapper.v` file, create ports on the freeze_wrapper module for the 64-bit Avalon-ST Single Clock FIFO component.

```
input   [63:0]   board_kernel_sc_fifo_in_data,
input            board_kernel_sc_fifo_in_valid,
output           board_kernel_sc_fifo_in_ready,
output  [63:0]   board_kernel_sc_fifo_out_data,
output           board_kernel_sc_fifo_out_valid,
input            board_kernel_sc_fifo_out_ready
```

3. In the `freeze_wrapper.v` file, create ports on the kernel_system instance module to match the ports you added in the `board_spec.xml` file. Connect these signals to the top-level ports of the freeze_wrapper module.

```
.kernel_sc_fifo_in_data(board_kernel_sc_fifo_in_data)
.kernel_sc_fifo_in_valid(board_kernel_sc_fifo_in_valid),
.kernel_sc_fifo_in_ready(board_kernel_sc_fifo_in_ready),
.kernel_sc_fifo_out_data(board_kernel_sc_fifo_out_data),
.kernel_sc_fifo_out_valid(board_kernel_sc_fifo_out_valid),
.kernel_sc_fifo_out_ready(board_kernel_sc_fifo_out_ready)
```

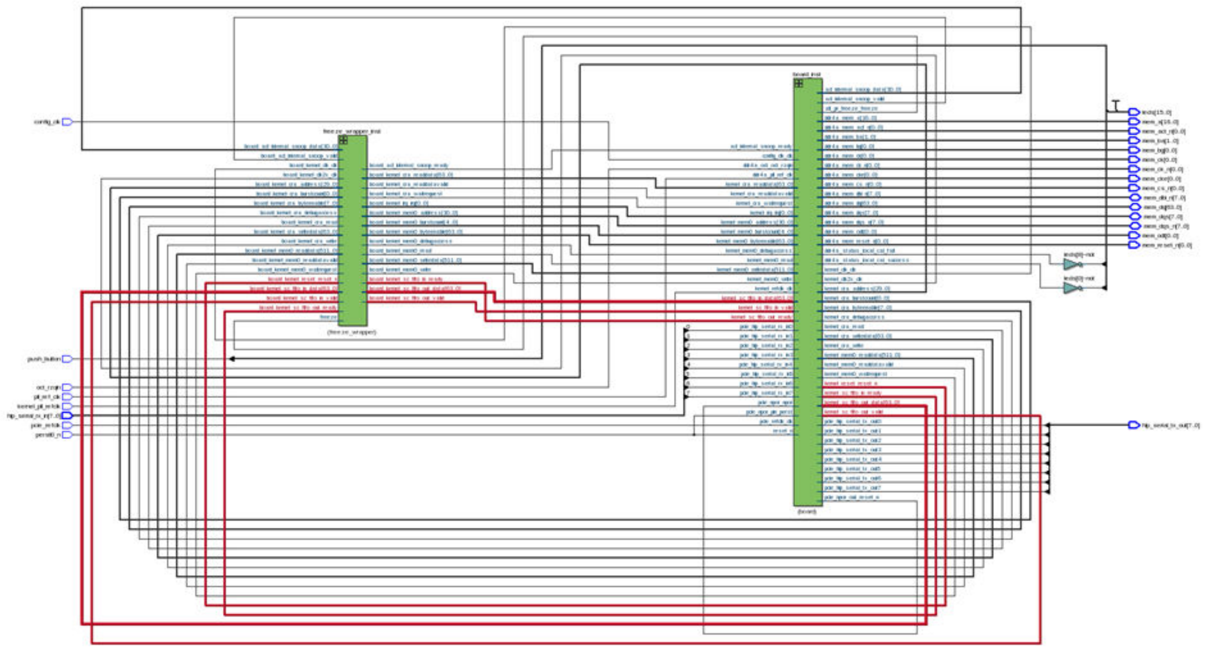
4. Save the `freeze_wrapper.v` file.

Updating the Top-Level I/O Ring with the Modified board.qsys and freeze_wrapper.v Files

Add ports and signals to the board and freeze_wrapper instances in the `top.v` file.

1. Open the `top.v` file in the Intel Quartus Prime Pro Edition software.
2. Add the new ports to the board instance.
3. Add the new ports to the freeze_wrapper instance.
4. Add signal (wires) to connect the board instance to the freeze_wrapper instance.
5. Save the `top.v` file.
6. In the Intel Quartus Prime Pro Edition software, run Analysis and Synthesis to check the syntax of your RTL and fix any errors.

Figure 1-11: RTL Netlist of the New Board Interface Connected to the New Freeze Wrapper



Updating the Original Custom Platform Directory with the New Custom Platform Modifications

The final step to customizing your Custom Platform is to copy all modified files back into the original Custom Platform directory (that is, the `a10gx_ref_18.1/hardware/a10gx_fifo` directory). By updating the files in the `a10gx_fifo` directory, the Intel FPGA SDK for OpenCL Offline Compiler will use the new customized Custom Platform when it performs subsequent compilations that target your Intel Arria 10 board.

1. Copy the following files back into the `hardware/a10gx_fifo` directory.

Table 1-8: Files to be Copied into the hardware/a10gx_fifo Directory

Files	Changes
	ROOT PARTITION
`top.v`	Added extra ports between the board and freeze wrapper components.
	BOARD INTERFACE
`board.qsys`	Added a FIFO component to the Platform Designer framework.
`base.qar`	Copied and replaced the `base.qar` file from the current directory back into the `hardware/a10gx_fifo` directory.
	FREEZE WRAPPER FILES

Files	Changes
`freeze_wrapper.v`	Added extra ports for the FIFO component that is part of the kernel logic.

2. After modifying the files in the original Custom Platform directory, regenerate your new Custom Platform by performing the following tasks:
   a. Ensure that the `vector_add.cl` file is in the `a10gx_ref_18.1` project directory. If not, download the design example and copy the `vector_add.cl` file to the `a10gx_ref_18.1` directory.
   b. At a command prompt, invoke the `aoc vector_add.cl –v --no-interleaving default` command to compile the vector_add kernel to hardware. If the Intel FPGA SDK for OpenCL Offline Compiler reports any errors, refer to the *Compilation Log Files* section for more information that can help you debug your kernel.
   c. After the Intel FPGA SDK for OpenCL Offline Compiler finishes compiling the vector_add kernel, refer to the *Analyzing the Results from Compilation* section to check the Fitter, Timing Analyzer reports and placement in the Floorplanner.

Related Information

- **Analyzing the Results from Compilation** on page 1-30
- **Compilation Log Files** on page 1-30

Compilation Log Files

The compilation log files record verbose information while the software tools synthesize and compile the Custom Platform and the kernel.

Check the following files for Intel FPGA SDK for OpenCL Offline Compiler compilation errors:

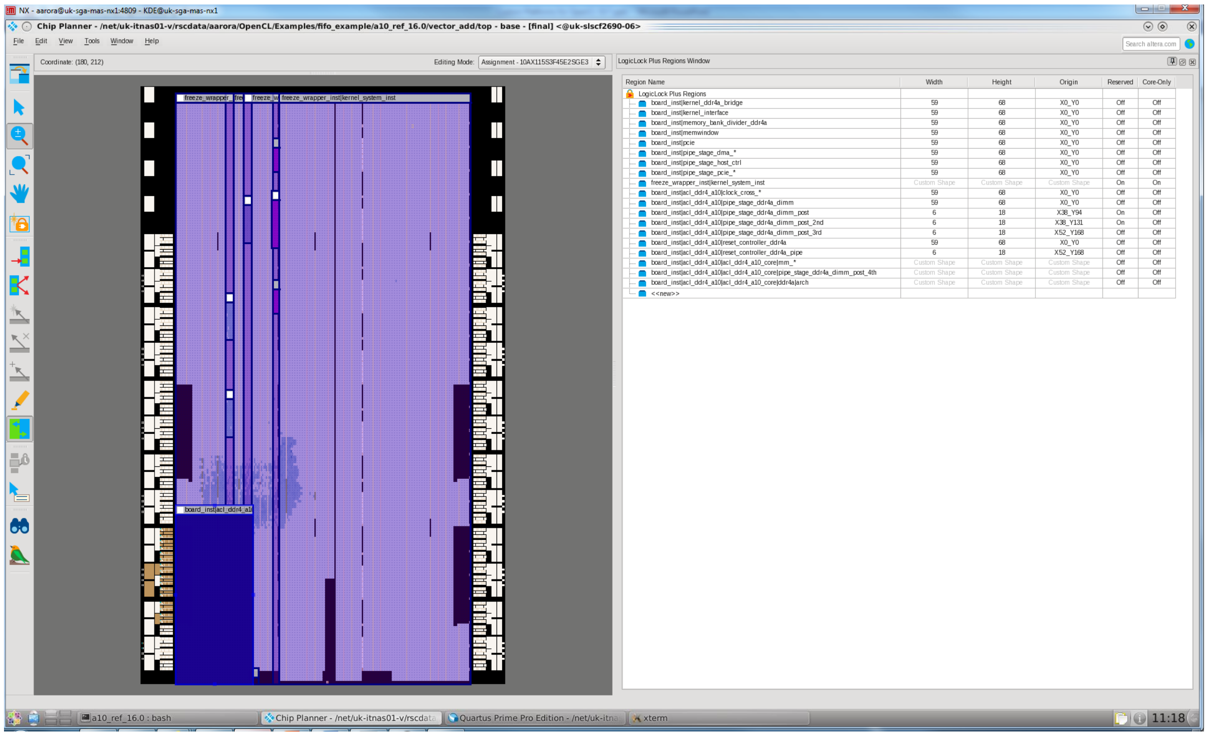
- The `<kernel_name>/<kernel_name>.log` file describes how the Intel FPGA SDK for OpenCL Offline Compiler optimizes the contents of the kernel file to target the FPGA. In the case of the vector_add design example, the corresponding `vector_add.log` file is in the `vector_add` directory.
- The `<kernel_name>/quartus_sh_compile.log` file describes how the Intel FPGA SDK for OpenCL Offline Compiler executes a complete hardware compilation flow from Analysis and Synthesis to the Fitter stage, timing analysis, and generation of the programming files.

Analyzing the Results from Compilation

After the full compilation flow has completed, check the results in the Intel Quartus Prime Pro Edition software GUI.

1. Start the Intel Quartus Prime Pro Edition software version 18.1.
2. From the **File** menu, select **Open Project**.
3. Open the project file `<project_directory>/top.qpf`.
4. To open the compilation report, select **Compilation Report** from the **Processing** menu.
5. In the compilation report, navigate to the Timing Analyzer section in the Table of Contents.
6. Verify that the results in the Timing Analysis Report are satisfactory.
7. Open the **Chip Planner** by clicking the corresponding tool bar button.
8. Verify that the placement results in the Chip Planner are satisfactory.

Figure 1-12: Floorplan and Placement Results for the Modified Intel Arria 10 Custom Platform



Chip Planner - /net/uk-itnas01-v/rscdata/aarora/OpenCL/Examples/fifo_example/a10_ref_16.0/vector_add/top - base - [final] <@uk-slscf2690-06>

Coordinate: (180, 212)

Editing Mode: Assignment - 10AX115S3F45E2SGE3

LogicLock Plus Regions Window

Region Name	Width	Height	Origin	Reserved	Core-Only
LogicLock Plus Regions					
board_inst|kernel_ddr4a_bridge	59	68	X0_Y0	Off	Off
board_inst|kernel_interface	59	68	X0_Y0	Off	Off
board_inst|memory_bank_divider_ddr4a	59	68	X0_Y0	Off	Off
board_inst|memwindow	59	68	X0_Y0	Off	Off
board_inst|pcie	59	68	X0_Y0	Off	Off
board_inst|pipe_stage_dma_*	59	68	X0_Y0	Off	Off
board_inst|pipe_stage_host_ctrl	59	68	X0_Y0	Off	Off
board_inst|pipe_stage_pcie_*	59	68	X0_Y0	Off	Off
freeze_wrapper_inst|kernel_system_inst	Custom Shape	Custom Shape	Custom Shape	On	On
board_inst|acl_ddr4_a10|clock_cross_*	59	68	X0_Y0	Off	Off
board_inst|acl_ddr4_a10|pipe_stage_ddr4a_dimm	59	68	X0_Y0	Off	Off
board_inst|acl_ddr4_a10|pipe_stage_ddr4a_dimm_post	6	18	X38_Y94	On	Off
board_inst|acl_ddr4_a10|pipe_stage_ddr4a_dimm_post_2nd	6	18	X38_Y131	On	Off
board_inst|acl_ddr4_a10|pipe_stage_ddr4a_dimm_post_3rd	6	18	X52_Y168	Off	Off
board_inst|acl_ddr4_a10|reset_controller_ddr4a	59	68	X0_Y0	Off	Off
board_inst|acl_ddr4_a10|reset_controller_ddr4a_pipe	6	18	X52_Y168	Off	Off
board_inst|acl_ddr4_a10|acl_ddr4_a10_core|mm_*	Custom Shape	Custom Shape	Custom Shape	Off	Off
board_inst|acl_ddr4_a10|acl_ddr4_a10_core|pipe_stage_ddr4a_dimm_post_4th	Custom Shape	Custom Shape	Custom Shape	Off	Off
board_inst|acl_ddr4_a10|acl_ddr4_a10_core|ddr4a|arch	Custom Shape	Custom Shape	Custom Shape	Off	Off
<<new>>					

File
Edit
View
Tools
Window
Help
Search altera.com
freeze_wrapper_inst\|kernel_system_inst
board_inst\|acl_ddr4_a1
a10_ref_16.0 : bash
Chip Planner - /net/uk-itnas01-v/rscdata
Quartus Prime Pro Edition - /net/uk-itn
xterm
11:18

Updating Your Custom Platform to Target a Different Device

If you want to target a different device variant that is based on but not included in an existing Custom Platform, update your Custom Platform accordingly.

Targeting a Device that Has a Migration Path in the Intel Quartus Prime Software

A migration path in the Intel Quartus Prime software allows you to migrate from Device A to Device B if they have the same package size (for example, F45 (1932 pins)).

To update your Custom Platform, perform the following tasks:

1. Change the target device in your Intel Quartus Prime project by editing the `top.qsf` file.
2. Edit the `device.tcl` file to specify the new target device.
3. Change the device model file listed in the in `board_spec.xml` file. You may find the device model file for the target device in the Intel Quartus Prime software installation directory. Prior to updating the Custom Platform, the device model file is `10ax115h3f34e2sg_dm.xml`.
4. Regenerate the Custom Platform.

Targeting a Device that Has a Different Package Size

To target a device that has a different package size from the current device, you must edit the `top.qsf`, `flat.qsf`, and `base.qsf` Intel Quartus Prime Settings Files that are in the Custom Platform.

To update your Custom Platform, perform the following tasks:

1. Change target device in the Intel Quartus Prime project by editing the `top.qsf` file.
2. Comment out any location constraints and any PR regions in the `flat.qsf` and `base.qsf` files. Location constraints and PR regions will be recreated when you rebuild the new package variant.
3. Edit the `device.tcl` file to specify the new target device. Change the device model file listed in the `board_spec.xml` file. You may find the device model file for the target device in the Intel Quartus Prime software installation directory. Prior to updating the Custom Platform, the device model file is `10ax115h3f34e2sg_dm.xml`.
4. Regenerate the Custom Platform.

Migrating the Custom Platform between Different Intel Quartus Prime Software Versions

Ensure that the version of your Intel Arria 10 Custom Platform matches the versions of the Intel Quartus Prime Pro Edition software and the Intel FPGA SDK for OpenCL.

Custom Platform Automigration for Forward Compatibility

The automigration feature updates an existing Intel-preferred Custom Platform for use with the current version of the Intel Quartus Prime Pro Edition software and the Intel FPGA SDK for OpenCL.

Important: Automigration is more likely to complete successfully if your Custom Platform resembles an Intel FPGA SDK for OpenCL Reference Platform as closely as possible.

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

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

A successfully-migrated Custom Platform will preserve its original functionality. In most cases, new features in a new version of the Intel Quartus Prime Design Suite will not interfere with Custom Platform functionality.

When the Intel FPGA SDK for OpenCL Offline Compiler compiles a kernel, it probes the `board_spec.xml` file for the following information:

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

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

The automigration process does not modify the installed Custom Platform.

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

Customizing Automigration

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

Disable automigration in one of the following manners:

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

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

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

Overall Summary

To successfully execute the OpenCL design flow, you must create and modify a Custom Platform correctly to suit your system. By following the guidelines outlined in this application note, you can modify a Custom Platform efficiently.

Document Revision History for Compiling and Customizing an Intel Arria 10 Custom Platform for OpenCL

Document Version	Intel Quartus Prime Version	Changes
2018.10.30	18.1	• Changed `base.qdb` to `base.qar` throughout. • Changed `a10gx_ref_16.0` to `a10gx_ref_18.1` throughout. • Changed `ALTERAOCLSDKROOT` to `INTELFPGAOCLSDKROOT` throughout. • Changed the device model file from `10ax115h3f34e2sge3_dm.xml` to `10ax115h3f34e2sg_dm.xml` throughout. • Rebranded the following occurrences: • Arria 10 to Intel Arria 10 • Altera SDK for OpenCL to Intel FPGA SDK for OpenCL • Altera to Intel • Altera Offline Compiler (AOC) to Intel FPGA SDK for OpenCL Offline Compiler • Qsys Pro to Platform Designer • LogicLock to Logic Lock • TimeQuest Timing Analyzer to Timing Analyzer • Quartus Prime Pro Edition to Intel Quartus Prime Pro Edition • Updated the titles of guides to point to the rebranded Intel FPGA SDK for OpenCL guides. • In **Modifying the freeze_wrapper.v File** on page 1-28, fixed a typo error in step 2 and added a missing entry in the code block.

Date	Version	Changes
December 2016	2016.12.09	Converted content to DITA with minor editorial changes.
October 2016	2016.10.21	Initial release.