

Intel[®] FPGA SDK for OpenCL[™]

Intel[®] Stratix[®] 10 GX FPGA Development Kit Reference Platform Porting Guide

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



Subscribe

Send Feedback

UG-20102 | 2017.11.21

Latest document on the web: [PDF](#) | [HTML](#)

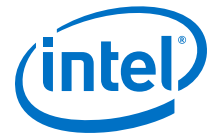


Contents

1 Intel® FPGA SDK for OpenCL™ Intel® Stratix® 10 GX FPGA Development Kit Reference Platform Porting Guide.....	4
1.1 Intel Stratix 10 GX FPGA Development Kit Reference Platform: Prerequisites.....	4
1.2 Features of the Intel Stratix 10 GX FPGA Development Kit Reference Platform.....	5
1.2.1 Intel Stratix 10 GX FPGA Development Kit Reference Platform Board Variants.....	6
1.3 Contents of the Intel Stratix 10 GX FPGA Development Kit Reference Platform.....	7
2 Developing Your Intel Stratix 10 Custom Platform	10
2.1 Initializing Your Intel Stratix 10 Custom Platform.....	10
2.2 Modifying the Intel Stratix 10 GX FPGA Development Kit Reference Platform Design.....	11
2.3 Integrating Your Intel Stratix 10 Custom Platform with the Intel FPGA SDK for OpenCL..	12
2.4 Setting up the Intel Stratix 10 Custom Platform Software Development Environment.....	14
2.5 Establishing Intel Stratix 10 Custom Platform Host Communication.....	15
2.6 Branding Your Intel Stratix 10 Custom Platform.....	15
2.7 Changing the Device Part Number.....	16
2.8 Connecting the Memory in the Intel Stratix 10 Custom Platform.....	17
2.9 Modifying the Kernel PLL Reference Clock.....	18
2.10 Integrating an OpenCL Kernel in Your Intel Stratix 10 Custom Platform.....	18
2.11 Guaranteeing Timing Closure in the Intel Stratix 10 Custom Platform.....	19
2.11.1 Generating the base.qar Post-Fit Netlist for Your Intel Stratix 10 Custom Platform.....	20
2.12 Troubleshooting Intel Stratix 10 GX FPGA Development Kit Reference Platform Porting Issues	21
3 Intel Stratix 10 GX FPGA Development Kit Reference Platform Design Architecture.....	22
3.1 Host-to-Intel Stratix 10 FPGA Communication over PCIe	22
3.1.1 Instantiation of Intel Stratix 10 PCIe Hard IP with Direct Memory Access.....	22
3.1.2 Device Identification Registers for Intel Stratix 10 PCIe Hard IP.....	24
3.1.3 Instantiation of the version_id Component.....	26
3.1.4 Definitions of Intel Stratix 10 FPGA Development Kit Reference Platform Hardware Constraints in Software Headers Files.....	26
3.1.5 PCIe Kernel Driver for the Intel Stratix 10 GX FPGA Development Kit Reference Platform.....	27
3.1.6 Direct Memory Access.....	28
3.1.7 Message Signaled Interrupt.....	30
3.1.8 Cable Autodetect.....	32
3.2 DDR4 as Global Memory for OpenCL Applications.....	32
3.2.1 DDR4 IP Instantiation.....	33
3.2.2 DDR4 Connection to PCIe Host.....	33
3.2.3 DDR4 Connection to the OpenCL Kernel.....	34
3.3 Host Connection to OpenCL Kernels.....	34
3.4 Intel Stratix 10 FPGA System Design.....	34
3.4.1 Clocks.....	34
3.4.2 Resets.....	35
3.4.3 Floorplan.....	36
3.4.4 Global Routing.....	38
3.4.5 Pipelining.....	38
3.4.6 DDR4 Calibration.....	39



3.5 Guaranteed Timing Closure of the Intel Stratix 10 GX FPGA Development Kit Reference Platform Design.....	39
3.5.1 Supply the Kernel Clock.....	39
3.5.2 Guarantee Kernel Clock Timing.....	40
3.5.3 Provide a Timing-Closed Post-Fit Netlist.....	40
3.6 Intel Quartus Prime Compilation Flow and Scripts.....	42
3.6.1 Intel Quartus Prime Compilation Flow for Board Developers	42
3.6.2 Intel Quartus Prime Compilation Flow for Custom Platform Users.....	43
3.6.3 Platform Designer System Generation.....	45
3.6.4 QDB File Generation.....	45
3.7 Addition of Timing Constraints.....	45
3.8 Connection of the Intel Stratix 10 GX FPGA Development Kit Reference Platform to the Intel FPGA SDK for OpenCL.....	45
3.8.1 Describe the Intel Stratix 10 GX FPGA Development Kit Reference Platform to the Intel FPGA SDK for OpenCL.....	46
3.8.2 Describe the Intel Stratix 10 GX FPGA Development Kit Reference Platform Hardware to the Intel FPGA SDK for OpenCL.....	46
3.9 Intel Stratix 10 FPGA Programming Flow.....	47
3.9.1 Define the Contents of the fpga.bin File for the Intel Stratix 10 GX FPGA Development Kit Reference Platform.....	47
3.10 Host-to-Device MMD Software Implementation.....	48
3.11 Implementation of Intel FPGA SDK for OpenCL Utilities.....	48
3.11.1 aocl install.....	48
3.11.2 aocl uninstall.....	49
3.11.3 aocl program.....	49
3.11.4 aocl flash.....	49
3.11.5 aocl diagnose.....	50
3.11.6 aocl list-devices.....	50
3.12 Intel Stratix 10 FPGA Development Kit Reference Platform Scripts.....	50
3.13 Considerations in Intel Stratix 10 GX FPGA Development Kit Reference Platform Implementation.....	51
4 Document Revision History.....	52



1 Intel® FPGA SDK for OpenCL™ Intel® Stratix® 10 GX FPGA Development Kit Reference Platform Porting Guide

The *Intel® Stratix® 10 GX FPGA Development Kit Reference Platform Porting Guide* describes procedures and design considerations for modifying the Intel Stratix 10 GX FPGA Development Kit Reference Platform (s10_ref) into your own Custom Platform for use with the Intel FPGA Software Development Kit (SDK) for OpenCL™ (1) (2).

1.1 Intel Stratix 10 GX FPGA Development Kit Reference Platform: Prerequisites

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

Prerequisites for the altera_s10pciedk Reference Platform:

- An Intel Stratix 10-based accelerator card with working PCI Express* (PCIe*) and memory interfaces

Test these interfaces together in the same design using the same version of the Intel Quartus® Prime Pro Edition software that you will use to develop your Custom Platform.

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

Alternatively, contact [support](#) for assistance.

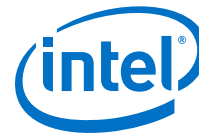
- Intel Quartus Prime Pro Edition software
- Designing with Logic Lock regions

General prerequisites:

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

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

(2) The Intel FPGA SDK for OpenCL is based on a published Khronos specification, and has passed the Khronos Conformance Testing Process. Current conformance status can be found at www.khronos.org/conformance.



- Tcl scripting
- PCIe
- DDR4 external memory

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

- Custom Platform Toolkit and the *Intel FPGA SDK for OpenCL Custom Platform Toolkit User Guide*
- Intel Arria® 10 Reference Platform (a10_ref) and the *Intel FPGA SDK for OpenCL Intel Arria 10 GX FPGA Development Kit Reference Platform Porting Guide*

The whole software stack in the s10_ref is derived from the a10_ref Reference Platform.

Related Links

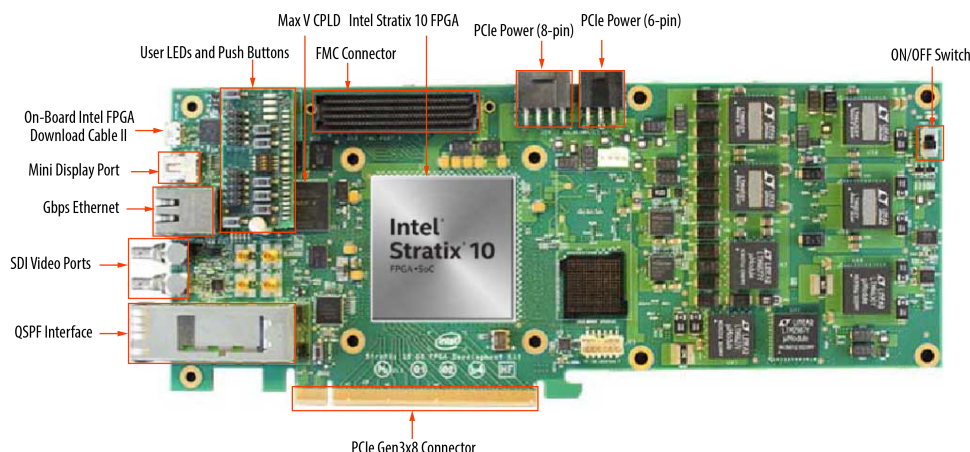
- [Intel FPGA SDK for OpenCL Custom Platform Toolkit User Guide](#)
- [Intel FPGA SDK for OpenCL Intel Arria 10 GX FPGA Development Kit Reference Platform Porting Guide](#)
- [Intel FPGA SDK for OpenCL Intel Arria 10 SoC Development Kit Reference Platform Porting Guide](#)
- [Intel FPGA SDK for OpenCL Intel Cyclone V SoC Development Kit Reference Platform Porting Guide](#)
- [Intel FPGA SDK for OpenCL Intel Stratix V Network Reference Platform Porting Guide](#)

1.2 Features of the Intel Stratix 10 GX FPGA Development Kit Reference Platform

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

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

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



Features of the s10_ref Reference Platform:

- **OpenCL Host**
The s10_ref Reference Platform uses a PCIe-based host that connects to the Intel Stratix 10 PCIe Gen2 x8 hard IP core.
- **OpenCL Global Memory**
The hardware provides one 2-gigabyte (GB) DDR4 SDRAM daughtercard that is mounted on the HiLo connector (J14 in [Figure 1](#) on page 6).
- **FPGA Programming**
Via external cable and the Intel Stratix 10 GX FPGA Development Kit's on-board Intel FPGA Download Cable II interface.
- **Guaranteed Timing**
The s10_ref Reference Platform relies on the Intel Quartus Prime Pro Edition compilation flow to provide guaranteed timing closure. The timing-clean s10_ref Reference Platform is preserved in the form of a precompiled post-fit netlist (that is, the base.qdb Intel Quartus Prime Database Export File). The Intel FPGA SDK for OpenCL Offline Compiler imports this preserved post-fit netlist into each OpenCL kernel compilation.

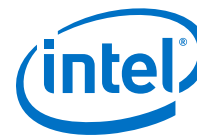
1.2.1 Intel Stratix 10 GX FPGA Development Kit Reference Platform Board Variants

The Intel Stratix 10 GX FPGA Development Kit Reference Platform has one board variant (that is, s10gx_ea_htile) that targets the Intel Stratix 10 GX FPGA Development Kit containing the RevA silicon for Intel Stratix 10 FPGA (-3 speed grade) and DDR4-1866 SDRAM.

To compile your OpenCL kernel for a specific board variant, include the `-board=<board_name>` option in your `aoc` command (for example, `aoc -board=s10gx_ea_htile myKernel.cl`).

Related Links

[Compiling a Kernel for a Specific FPGA Board \(-board=<board_name>\)](#)



1.3 Contents of the Intel Stratix 10 GX FPGA Development Kit Reference Platform

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

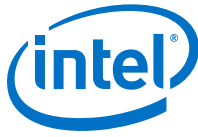
Table 1. Highlights of the Intel Stratix 10 GX FPGA Development Kit Reference Platform Directory

Windows File or Folder	Linux File or 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 s10gx_ea_htile board variant. See Contents of the s10gx_ea_htile Directory for a list of files in this directory.
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	For Windows, the source_windows64 folder contains source codes for the MMD library and SDK utilities. The MMD library and the SDK utilities are in the windows64 folder. For Linux, the source directory contains source codes for the MMD library and SDK utilities. The MMD library and the SDK utilities are in the linux64 directory.

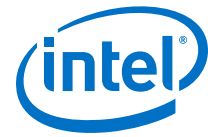
Table 2. Contents of the s10gx_ea_htile Directory

The following table lists the files in the *INTELFPGAOCCLSDKROOT/board/s10_ref/hardware/s10gx_ea_htile* directory, where *INTELFPGAOCCLSDKROOT* points to the location of the SDK installation.

File	Description
acl_ddr4_s10.qsys	Platform Designer system that, together with the .ip files in the ip/acl_ddr4_s10/ subdirectory, implements the mem component.
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 s10_ref 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 Archive File that contains root_partition.qdb and pr_base.id. This file is generated by the scripts/post_flow_pr.tcl file during base revision compile, and is used during import revision compilation. <i>root_partition.qdb</i> Intel Quartus Prime Database Export File the contains the precompiled netlist of the static regions of the design. <i>pr_base.id</i> 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.
board.qsys	Platform Designer system that implements the board interfaces (that is, the static region) of the OpenCL hardware system.
continued...	



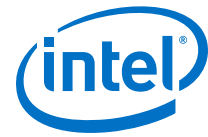
File	Description
board_spec.xml	XML file that provides the definition of the board hardware interfaces to the SDK.
device.tcl	Tcl file that is included in all revisions and contains all device-specific information (for example, device family, ordering part number (OPN), voltage settings, etc.)
flat.qsf	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, top.qsf, and top_synth.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.
import_compile.tcl	Tcl script for the SDK-user compilation flow (that is, import revision compilation).
max5_116.pof	Programming file for the MAX® V device on the Intel Stratix 10 GX FPGA Development Kit that sets the memory reference clock to 116 MHz by default at power-up. You must program the max5_116.pof file onto your s10gx_ea_htile board.
opencl_bsp_ip.qsf	Intel Quartus Prime Settings File that collects all the required .ip files in a unique location. During flat and base revision compilations, the board.qsys and acl_ddr4_s10.qsys Platform Designers files are added to the opencl_bsp_ip.qsf file.
quartus.ini	Contains any special Intel Quartus Prime software options that you need to compile OpenCL kernels for the s10_ref Reference Platform.
top.qpf	Intel Quartus Prime Project File for the OpenCL hardware system.
top.qsf	Intel Quartus Prime Settings File for the SDK-user compilation flow.
top.sdc	Synopsys Design Constraints File that contains board-specific timing constraints.
top.v	Top-level Verilog Design File for the OpenCL hardware system.
top_post.sdc	Platform Designer and Intel FPGA SDK for OpenCL IP-specific timing constraints.
top_synth.qsf	Intel Quartus Prime Settings File for the Intel Quartus Prime revision in which the OpenCL kernel system is synthesized.
ip/acl_ddr4_s10/<file_name>	Directory containing the .ip files that the Intel Quartus Prime Pro Edition software needs to parameterize the mem component. You must provide both the acl_ddr4_s10.qsys file and the corresponding .ip files in this directory to the Intel Quartus Prime Pro Edition software.
ip/board/<file_name>	Directory containing the .ip files that the Intel Quartus Prime Pro Edition software needs to parameterize the board instance. You must provide both the board.qsys file and the corresponding .ip files in this directory to the Intel Quartus Prime Pro Edition software.
ip/freeze_wrapper.v	Verilog Design File that implements the freeze logic.
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 <i>Message Signaled Interrupts</i> section for more information.
continued...	



File	Description
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 <i>Define the Contents of the fpga.bin File for the Intel Stratix 10 GX FPGA Development Kit Reference Platform</i> section.
scripts/helpers.tcl	Tcl script with helper functions used by qar_ip_files.tcl
scripts/post_flow_pr.tcl	Tcl script that implements the guaranteed timing closure flow, as described in the <i>Guaranteed Timing Closure of the Intel Stratix 10 GX FPGA Development Kit Reference Platform Design</i> section.
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.
scripts/qar_ip_files.tcl	Tcl script that packages up base.qdb and pr_base.id during base revision compile.
scripts/adjust_plls_sl0.tcl	PLL adjustment script for the kernel clock PLL to guarantee timing closure on the kernel clock, by setting it to the maximum allowed frequency.
scripts/kernel_system_update.tcl	Tcl script to update the kernel system.
root_partition.qdb	Database export of the base revision compile of the postfit netlist of the static board interface region.

Related Links

- [Guaranteed Timing Closure of the Intel Stratix 10 GX FPGA Development Kit Reference Platform Design](#) on page 39
- [Message Signaled Interrupt](#) on page 30
- [Define the Contents of the fpga.bin File for the Intel Stratix 10 GX FPGA Development Kit Reference Platform](#) on page 47



2 Developing Your Intel Stratix 10 Custom Platform

Use the tools available in Intel Stratix 10 GX FPGA Development Kit Reference Platform (s10_ref) and the Intel FPGA SDK for OpenCL Custom Platform Toolkit together to create your own Custom Platform.

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

- *Intel FPGA SDK for OpenCL Custom Platform User Guide*
- Contents of the SDK Custom Platform Toolkit
- *Intel FPGA SDK for OpenCL Intel Arria 10 GX FPGA Development Kit Reference Platform Porting Guide*
- Documentation for all the Intel FPGA IP in your Custom Platform
- *Intel FPGA SDK for OpenCL Getting Started Guide*
- *Intel FPGA SDK for OpenCL Programming Guide*

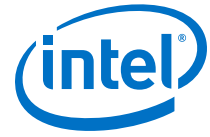
In addition, you must independently verify all IP on your computing card (for example, PCIe controllers and DDR4 external memory).

Related Links

- [Intel FPGA SDK for OpenCL Custom Platform Toolkit User Guide](#)
- [Intel FPGA SDK for OpenCL Intel Arria 10 GX FPGA Development Kit Reference Platform Porting Guide](#)
- [Intel FPGA SDK for OpenCL Intel Arria 10 SoC Development Kit Reference Platform Porting Guide](#)
- [Intel FPGA SDK for OpenCL Intel Cyclone V SoC Development Kit Reference Platform Porting Guide](#)
- [Intel FPGA SDK for OpenCL Getting Started Guide](#)
- [Intel FPGA SDK for OpenCL Programming Guide](#)

2.1 Initializing Your Intel Stratix 10 Custom Platform

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



1. Copy the `s10_ref` Reference Platform from the drop directory.
2. Paste the `s10_ref` directory into a directory that you own (that is, not a system directory) and then rename it (`<your_custom_platform>`).
3. Choose the `s10gx_ea_htile` board variant in the `<your_custom_platform>/hardware` directory to match the production silicon for the Intel Stratix 10 FPGA as the basis of your design.
4. Rename `s10gx_ea_htile` board variant to match the name of your FPGA board (`<your_custom_platform>/hardware/<board_name>`).
5. Modify the `<your_custom_platform>/board_env.xml` file so that the name and default fields match the changes you made in step 2 on page 11 and step 4 on page 11, respectively.
6. Modify the `my_board` name in the inside `<your_custom_platform>/hardware/<board_name>/board_spec.xml` file to match the change you made in step 2 on page 11.

```
> aoc -list-boards
Board list:
my_board
```

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

Related Links

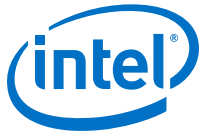
- [Setting the Intel FPGA SDK for OpenCL User Environment Variables for Windows](#)
- [Setting the Intel FPGA SDK for OpenCL User Environment Variables for Linux](#)
- [Describe the Intel Stratix 10 GX FPGA Development Kit Reference Platform to the Intel FPGA SDK for OpenCL on page 46](#)

2.2 Modifying the Intel Stratix 10 GX FPGA Development Kit Reference Platform Design

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

You can add a component in Platform Designer and connect it to the existing system, or add a Verilog file to the available system. After adding the custom components, connect those components in Platform Designer.

1. Instantiate your PCIe controller, as described in *Host-to-Intel Stratix 10 Communication over PCIe* section.
2. Instantiate any memory controllers and I/O channels. You can add the board interface hardware either as Platform Designer components in the `board.qsys` Platform Designer system or as HDL in the `top.v` file.



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

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

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

Related Links

[Host-to-Intel Stratix 10 FPGA Communication over PCIe](#) on page 22

2.3 Integrating Your Intel Stratix 10 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.

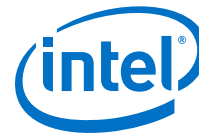
1. Update the `<your_custom_platform>/hardware/<board_name>/board_spec.xml` file. Ensure that there is at least one global memory interface, and all the global memory interfaces correspond to the exported interfaces from the `board.qsys` Platform Designer System File.
2. Set the environment variable `ACL_DEFAULT_FLOW` to `flat`.

Setting this environment variable instructs the SDK to compile the flat revision corresponding to `<your_custom_platform>/hardware/<board_name>/flat.qsf` file without the partitions or Logic Locks.

Tip: Intel recommends that you get a timing clean flat revision compiled before proceeding to the base revision compiles. You can also invoke the following command with the `-bsp-flow=<revision_type>` attribute to run different revisions of your project (for example, flat or base compiles).

```
aoc -bsp-flow=flat boardtest.cl -o=bin/boardtest.aocx
```

3. Set the environment variable `ACL_DEFAULT_FLOW` to `base`.
Setting this environment variable instructs the SDK to compile the base revision corresponding to the `<your_custom_platform>/hardware/<board_name>/base.qsf` file.
4. Perform the steps outlined in the `INTELFPGAOCCLSDKROOT/board/custom_platform_toolkit/tests/README.txt` file to compile the `INTELFPGAOCCLSDKROOT/board/custom_platform_toolkit/tests/boardtest/boardtest.cl` OpenCL kernel source file.



The environment variable `INTELFPGAOCSDKROOT` points to the location of the SDK installation.

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

You might be able to fix minor timing issues by simply compiling your kernel with a different seed.

Following is the XML code of an example `board_spec.xml` file:

```
<?xml version="1.0"?>
<board version="17.1" name="sl0gx_ea_htile">

  <compile name="top" project="top" revision="top" qsys_file="none"
generic_kernel="1">
    <generate cmd="quartus_sh -t scripts/pre_flow_pr.tcl"/>
    <synthesize cmd="quartus_cdb -t import_compile.tcl"/>
    <auto_migrate platform_type="sl0_ref" >
      <include fixes=""/>
    </auto_migrate>
  </compile>

  <compile name="base" project="top" revision="base" qsys_file="none"
generic_kernel="1">
    <generate cmd="quartus_sh -t scripts/pre_flow_pr.tcl base"/>
    <synthesize cmd="quartus_sh --flow compile top -c base"/>
    <auto_migrate platform_type="sl0_ref" >
      <include fixes=""/>
    </auto_migrate>
  </compile>

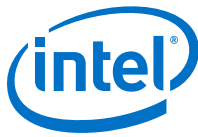
  <compile name="flat" project="top" revision="flat" qsys_file="none"
generic_kernel="1">
    <generate cmd="quartus_sh -t scripts/pre_flow_pr.tcl flat"/>
    <synthesize cmd="quartus_sh --flow compile top -c flat"/>
    <auto_migrate platform_type="sl0_ref" >
      <include fixes=""/>
    </auto_migrate>
  </compile>

  <device device_model="lsg280lu3f50elvgsl_dm.xml">
    <used_resources>
      <alms num="6566"/> <!-- ALMs used in final placement - ALMs used for
registers -->
      <ffs num="20030"/>
      <dsps num="0"/>
      <rams num="112"/>
    </used_resources>
  </device>

  <!-- DDR4-1866 -->
  <global_mem name="DDR" max_bandwidth="14928" 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>
```

Related Links

[Describe the Intel Stratix 10 GX FPGA Development Kit Reference Platform Hardware to the Intel FPGA SDK for OpenCL on page 46](#)

2.4 Setting up the Intel Stratix 10 Custom Platform Software Development Environment

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

- To compile the MMD layer for Windows, perform the following tasks:
 - a. Install the GNU make utility on your development machine.
 - b. Install a version of Microsoft Visual Studio that has the ability to compile 64-bit software (for example, Microsoft Visual Studio version 2010 Professional).
 - c. Set the development environment so that SDK users can invoke commands and utilities at the command prompt.
 - d. Modify the `<your_custom_platform_name>/source/Makefile.common` file so that `TOP_DEST_DIR` points to the top-level directory of your Custom Platform.
 - e. In the `Makefile.common` file or the development environment, set the `JUNGO_LICENSE` variable to your Jungo WinDriver license.
 - f. To check that you have set up the software development environment properly, invoke the `gmake` or `gmake clean` command.
- To compile the MMD layer for Linux, perform the following tasks:
 - a. Ensure that you use a Linux distribution that Intel supports (for example, GNU Compiler Collection (GCC) version 4.47).
 - b. Modify the `<your_custom_platform>/source/Makefile.common` file so that `TOP_DEST_DIR` points to the top-level directory of your Custom Platform.
- To check that you have set up the software environment properly, invoke the `make` or `make clean` command.

Related Links

[Jungo Connectivity Ltd. website](#)



2.5 Establishing Intel Stratix 10 Custom Platform Host Communication

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

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

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

2. Confirm that your operating system recognizes a PCIe device with your vendor and device IDs.
 - For Windows, open the **Device Manager** and verify that the correct device and IDs appear in the listed information.
 - For Linux, invoke the `lspci` command and verify that the correct device and IDs appear in the listed information.
3. Run the `aocl install <path_to_customplatform>` utility command to install the kernel driver on your machine.
4. For Windows, set the `PATH` environment variable. For Linux, set the `LD_LIBRARY_PATH` environment variable.

For more information about the settings for `PATH` and `LD_LIBRARY_PATH`, refer to *Setting the Intel FPGA SDK for OpenCL User Environment Variables* in the *Intel FPGA SDK for OpenCL Getting Started Guide*.
5. Modify the `version_id_test` function in your `<your_custom_platform>/source/host/mmd/acl_pcie_device.cpp` MMD source code file to exit after reading from the `version` ID register.
6. Run the `aocl diagnose` utility command and confirm that the `version` ID register reads back the ID successfully. You may set the environment variables `ACL_HAL_DEBUG` and `ACL_PCIE_DEBUG` to a value of 1 to visualize the result of the diagnostic test on your terminal.

Related Links

- [Integrating Your Intel Stratix 10 Custom Platform with the Intel FPGA SDK for OpenCL](#) on page 12
- [Setting the Intel FPGA SDK for OpenCL Environment Variables for Linux](#)
- [Setting the Intel FPGA SDK for OpenCL User Environment Variables for Windows](#)

2.6 Branding Your Intel Stratix 10 Custom Platform

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



1. In the software development environment available with the s10_ref Reference Platform, replace all references of "s10_ref" with the name of your Custom Platform.
2. Modify the PACKAGE_NAME and MMD_LIB_NAME fields in the `<your_custom_platform>/source/Makefile.common` file.
3. Modify the name, linklib, and mmlibs elements in `<your_custom_platform>/board_env.xml` file to your custom MMD library name.
4. In your Custom Platform, modify the following lines of code in the `hw_pcie_constants.h` file to include information of your Custom Platform:

```
#define ACL_BOARD_PKG_NAME "s10_ref"  
#define ACL_VENDOR_NAME "Intel Corporation"  
#define ACL_BOARD_NAME "Stratix 10 Reference Platform"
```

For Windows, the `hw_pcie_constants.h` file is in the `<your_custom_platform>\source_windows64\include` folder. For Linux, the `hw_pcie_constants.h` file is in the `<your_custom_platform>/linux64/driver` directory.

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

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

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

6. Update your Custom Platform's `board.qsys` Platform Designer system and the `hw_pcie_constants.h` file with the IDs defined in step 5 on page 16.
7. For Windows, update the DeviceList fields in the `<your_custom_platform>\windows64\driver\acl_boards_s10_ref.inf` file to match your PCIe ID values and then rename the file to `acl_board_<your_custom_platform>.inf`.

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

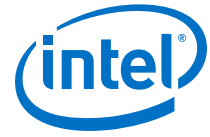
8. Run `make` in the `<your_custom_platform>/source` directory to generate the driver.

Related Links

[Device Identification Registers for Intel Stratix 10 PCIe Hard IP](#) on page 24

2.7 Changing the Device Part Number

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



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

- In the `device.tcl` file, change the device part number in the `set global assignment -name DEVICE 1SG280LU3F50E3VGS2` QSF assignment. The updated device number will appear in the `base.qsf`, `top.qsf`, and `top_synth.qsf` files.
- In the `board.qsys` and `acl_ddr4_s10.qsys` files, change all occurrences of `1SG280LU3F50E3VGS2`.

2.8 Connecting the Memory in the Intel Stratix 10 Custom Platform

Calibrate the external memory IP and controllers in your Custom Platform, and connect them to the host.

1. In your Custom Platform, instantiate your external memory IP based on the information in the *DDR4 as Global Memory for OpenCL Applications* section. Update the information pertaining to the `global_mem` element in the `<your_custom_platform>/hardware/<board_name>/board_spec.xml` file.
2. Remove the `boardtest` hardware configuration file that you created during the integration of your Custom Platform with the Intel FPGA SDK for OpenCL.
3. Recompile the `INTELFPGAOCCLSDKROOT/board/custom_platform_toolkit/tests/boardtest/boardtest.cl` kernel source file.

The environment variable `INTELFPGAOCCLSDKROOT` points to the location of the SDK installation.

4. Reprogram the FPGA with the new `boardtest` hardware configuration file and then reboot your machine.
5. Modify the `wait_for_uniphy` function in the `acl_pcie_device.cpp` MMD source code file to exit after checking the UniPHY status register. Rebuild the MMD software.

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

6. Run the `aocl diagnose` SDK utility and confirm that the host reads back both the version ID and the value 0 from the `uniphy_status` component. The utility should return the message `Uniphy are calibrated`.
7. Consider analyzing your design in the Signal Tap logic analyzer to confirm the successful calibration of all memory controllers.

Note: For more information on Signal Tap logic analyzer, download the Signal Tap II Logic Analyzer tutorial from the [University Program Tutorial](#) page.

Related Links

- [DDR4 as Global Memory for OpenCL Applications](#) on page 32
- [Integrating Your Intel Stratix 10 Custom Platform with the Intel FPGA SDK for OpenCL](#) on page 12
- [Signal Tap II with Verilog Designs](#)

2.9 Modifying the Kernel PLL Reference Clock

The Intel Stratix 10 GX FPGA Reference Platform uses an external 50 MHz clock as a reference for the I/O PLL. The I/O PLL relies on this reference clock to generate the internal `kernel_clk` clock, and the `kernel_clk2x` clock that runs at twice the frequency of `kernel_clk`. When porting the `s10_ref` Reference Platform to your own board using a different reference clock, update the `board.qsys` and `top.sdc` files with the new reference clock speed.

1. In the `<your_custom_platform>/hardware/<board_name>/board.qsys` file, update the `REF_CLK_RATE` parameter value on the `kernel_clk_gen` IP module.
2. In the `<your_custom_platform>/hardware/<board_name>/top.sdc` file, update the `create_clock` assignment for `kernel_pll_refclk`.
3. [Optional] In the `<your_custom_platform>/hardware/<board_name>/top.v` file, update the comment for the `kernel_pll_refclk` input port.

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

2.10 Integrating an OpenCL Kernel in Your Intel Stratix 10 Custom Platform

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

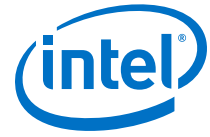
1. Perform the steps outlined in `INTELFPGAOCCLSDKROOT/board/custom_platform_toolkit/tests/README.txt` file to build the hardware configuration file from the `INTELFPGAOCCLSDKROOT/board/custom_platform_toolkit/tests/boardtest/boardtest.cl` kernel source file.

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

2. Program your FPGA device with the hardware configuration file you created in step 1 on page 18 and then reboot your machine.
3. Remove the early-exit modification in the `version_id_test` function in the `acl_pcie_device.cpp` file that you implemented when you established communication between the board and the host interface.

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

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



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

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

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

6. Set the environment variable `CL_CONTEXT_COMPILER_MODE_INTELFPGA` to a value of 3 and run the boardtest host application.

For more information on `CL_CONTEXT_COMPILER_MODE_INTELFPGA`, refer to *Troubleshooting Intel Stratix 10 GX FPGA Development Kit Reference Platform Porting Issues*.

Related Links

- [Establishing Intel Stratix 10 Custom Platform Host Communication](#) on page 15
- [Troubleshooting Intel Stratix 10 GX FPGA Development Kit Reference Platform Porting Issues](#) on page 21

2.11 Guaranteeing Timing Closure in the Intel Stratix 10 Custom Platform

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

1. Establish the floorplan of your design.

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

2. Compile several seeds of the `INTELFPGAOCCLSDKROOT/board/custom_platform_toolkit/tests/boardtest/boardtest.cl` file until you generate a design that closes timing cleanly.

To specify the seed number, include the `-seed=<N>` option in your `aoc` command.

3. Copy the `base.qar` file from the `INTELFPGAOCCLSDKROOT/board/s10_ref/hardware/s10gx_ea_htile` directory into your Custom Platform.
4. Use the `flat.qsf` file in the s10_ref Reference Platform as references to determine the type of information you must include in the `flat.qsf` file for your Custom Platform.

The `base.qsf`, `top.qsf`, and `top_synth.qsf` files automatically inherit all the settings in the `flat.qsf` file. However, if you need to modify Logic Lock Plus region, make the change only in the `base.qsf` file.

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

6. Remove the `ACL_DEFAULT_FLOW` environment variable that you added when integrating your Custom Platform with the Intel FPGA SDK for OpenCL.
7. Ensure that the environment variable `CL_CONTEXT_COMPILER_MODE_INTELFPGA` is not set.
8. Run the `boardtest_host` executable.

Related Links

- [Intel Stratix 10 FPGA System Design](#) on page 34
- [FPGA System Design](#)
- [Integrating Your Intel Stratix 10 Custom Platform with the Intel FPGA SDK for OpenCL](#) on page 12

2.11.1 Generating the base.qar Post-Fit Netlist for Your Intel Stratix 10 Custom Platform

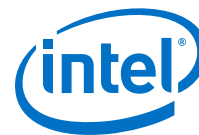
To implement the compilation flow, you must generate `base.qar` and `root_partition.qdb` Intel Quartus Prime Archive Files for your Intel Stratix 10 Custom Platform.

Following steps represent a general procedure for regenerating `base.qar` and `root_partition.qdb` files:

1. Port the system design and the `flat.qsf` file to your computing card.
2. Compile the `INTELFPGAOCSDKROOT/board/custom_platform_toolkit/tests/boardtest/boardtest.cl` kernel source file using the base revision. Fix any timing failures and recompile the kernel until timing is clean. You can add the `-bsp-flow=base` argument to the `aoc` command to generate `base.qar` and `root_partition.qdb` files during the kernel compilation.
INTELFPGAOCSDKROOT points to the location of the Intel FPGA SDK for OpenCL installation.
3. Copy the generated `base.qar` and `root_partition.qdb` files into your Custom Platform.
4. Using the default compilation flow, test `base.qar` and `root_partition.qdb` files across several OpenCL design examples and confirm that the following criteria are satisfied:
 - All compilations close timing.
 - The OpenCL design examples achieve satisfactory F_{max} .
 - The OpenCL design examples function on the accelerator board.

Related Links

- [Integrating Your Intel Stratix 10 Custom Platform with the Intel FPGA SDK for OpenCL](#) on page 12
- [Provide a Timing-Closed Post-Fit Netlist](#) on page 40

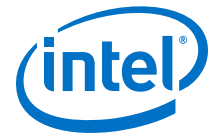


2.12 Troubleshooting Intel Stratix 10 GX FPGA Development Kit Reference Platform Porting Issues

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

Table 3. Intel FPGA SDK for OpenCL-Specific Environment Variables for Identifying Custom Platform Design Problems

Environment Variable	Description
<i>ACL_HAL_DEBUG</i>	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.
<i>ACL_PCIE_DEBUG</i>	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 <code>version ID</code> register was read correctly and the UniPHY IP cores are calibrated.
<i>ACL_PCIE_JTAG_CABLE</i>	Set this variable to override the default <code>quartus_pgm</code> argument that specifies the cable number. The default is cable 1. If there are multiple Intel FPGA Download Cable, you can specify a particular one here.
<i>ACL_PCIE_JTAG_DEVICE_INDEX</i>	Set this variable to override the default <code>quartus_pgm</code> argument that specifies the FPGA device index. By default, this variable has a value of 2. If the FPGA is not the first device in the JTAG chain, you can customize the value.
<i>ACL_PCIE_USE_JTAG_PROGRAMMING</i>	Set this variable to force the MMD to reprogram the FPGA using the JTAG cable.
<i>ACL_PCIE_DMA_USE_MSI</i>	Set this variable if you want to use MSI for DMA transfers on Windows.
<i>CL_CONTEXT_COMPILER_MODE_INTELFPGA</i>	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.



3 Intel Stratix 10 GX FPGA Development Kit Reference Platform Design Architecture

Intel created the Intel Stratix 10 GX FPGA Development Kit Reference Platform (s10_ref) based on various design considerations. Familiarize yourself with these design considerations. Having a thorough understanding of the design decision-making process might help in the design of your own Intel FPGA SDK for OpenCL Custom Platform.

[Host-to-Intel Stratix 10 FPGA Communication over PCIe](#) on page 22

[DDR4 as Global Memory for OpenCL Applications](#) on page 32

[Host Connection to OpenCL Kernels](#) on page 34

[Intel Stratix 10 FPGA System Design](#) on page 34

[Guaranteed Timing Closure of the Intel Stratix 10 GX FPGA Development Kit Reference Platform Design](#) on page 39

[Intel Quartus Prime Compilation Flow and Scripts](#) on page 42

[Addition of Timing Constraints](#) on page 45

[Connection of the Intel Stratix 10 GX FPGA Development Kit Reference Platform to the Intel FPGA SDK for OpenCL](#) on page 45

[Intel Stratix 10 FPGA Programming Flow](#) on page 47

[Host-to-Device MMD Software Implementation](#) on page 48

[Implementation of Intel FPGA SDK for OpenCL Utilities](#) on page 48

[Intel Stratix 10 FPGA Development Kit Reference Platform Scripts](#) on page 50

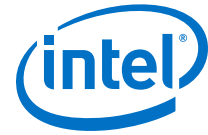
[Considerations in Intel Stratix 10 GX FPGA Development Kit Reference Platform Implementation](#) on page 51

3.1 Host-to-Intel Stratix 10 FPGA Communication over PCIe

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

3.1.1 Instantiation of Intel Stratix 10 PCIe Hard IP with Direct Memory Access

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



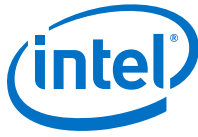
Dependencies

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

Table 4. Highlights of Intel Stratix 10 PCIe Hard IP Parameter Settings

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

Parameter(s)	Setting
System Settings	
Application interface type	Avalon-MM with DMA This Avalon Memory-Mapped (Avalon-MM) interface instantiates the embedded DMA of the PCIe hard IP core.
Hard IP mode	Gen2x8, Interface: 256-bit, 125 MHz Number of Lanes: x8 Lane Rate: Gen2 (5.0 Gbps) <i>Note:</i> This is not the fastest configuration, but it was chosen to obtain reasonable timing closure rates even when running flat compile flow on large designs.
Rx Buffer credit allocation	Low <i>Note:</i> This setting is derived experimentally.
Intel Stratix 10 Avalon-MM Settings	
Export MSI/MSI-X conduit interfaces	Enabled Export the MSI interface in order to connect the interrupt sent from the kernel interface to the MSI.
Instantiate Internal Descriptor Controller	Enabled Instantiates the descriptor controller in the Avalon-MM DMA bridge. Use the 128-entry descriptor controller that the PCIe hard IP core provides. Disabled for a10gx_hostch board variant The descriptor controller is implemented in the <code>ip/host_channel</code> subdirectory.
Address width of accessible PCIe memory space	64 bits This value is machine dependent. To avoid truncation of the MSI memory address, 64-bit machines should allot 64 bits to access the PCIe address space.
Base Address Register (BAR) Settings	
Base Address Registers (BARs)	This design uses two BARs. For BAR 0, set Type to 64-bit prefetchable memory . The Size parameter setting is disabled because the Instantiate Internal Descriptor Controller parameter is enabled in the Avalon-MM system settings. BAR 0 is only used to access the DMA Descriptor Controller, as described in the <i>Intel Stratix 10 Avalon-MM DMA for PCI Express</i> section of the <i>Intel Stratix 10 Avalon-MM DMA Interface for PCIe Solutions User Guide</i> . For Bar 4, set Type to 64-bit prefetchable memory , and set Size to 256 KBytes - 18 bits . BAR 4 is used to connect PCIe to the OpenCL kernel systems and other board modules.



Related Links

- [Avalon-MM Settings](#)
- [Intel Stratix 10 Avalon-MM DMA for PCI Express Design Example](#)

3.1.2 Device Identification Registers for Intel Stratix 10 PCIe Hard IP

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

Table 5. Device Hardware-Related PCIe ID Registers

ID Register Name	ID Provider	Description	Parameter Name in PCIe IP Core
Vendor ID	PCI-SIG®	Identifies the FPGA manufacturer. Always set this register to 0x1172 , which is the Intel vendor ID.	vendor_id_hw_tcl
Device ID	Intel	Describes the PCIe configuration on the FPGA according to Intel's internal guideline. Set the device ID to the device code of the FPGA on your accelerator board. For the Intel Stratix 10 GX FPGA Development Kit Reference Platform, set the Device ID register to 0x2494 , which signifies Gen 3 speed, 8 lanes, Intel Stratix 10 device family, and Avalon-MM interface, respectively. Refer to Table 6 on page 25 for more information.	device_id_hw_tcl
Revision ID		When setting this ID, ensure that it matches the following revision IDs: <ul style="list-style-type: none">• For Windows, the revision ID specified for the DeviceList field in the <code><your_custom_platform>\windows64\driver\acl_boards_<your_custom_platform>.inf</code> file.• For Linux, the revision ID specified for the ACL_PCI_REVISION variable in the <code><your_custom_platform>/linux64/driver/hw_pcie_constants.h</code> file.	—
Class Code	Intel	The Intel FPGA SDK for OpenCL utility checks the base class value to verify whether the board is an OpenCL device. Do not modify the class code settings. <ul style="list-style-type: none">• Base class: 0x12 for processing accelerator• Sub class: 0x00• Programming interface: 0x01	—
Subsystem Vendor ID	Board vendor	Identifies the manufacturer of the accelerator board. Set this register to the vendor ID of manufacturer of your accelerator board. For the s10_ref Reference Platform, the subsystem vendor ID is 0x1172 . If you are a board vendor, set this register to your vendor ID.	subsystem_vendor_id_hw_tcl
Subsystem Device ID	Board vendor	Identifies the accelerator board. The SDK uses this ID to identify the board because the software might perform differently on different boards. If you create a Custom Platform that supports multiple boards, use this ID to distinguish	subsystem_device_id_hw_tcl
continued...			



ID Register Name	ID Provider	Description	Parameter Name in PCIe IP Core
		<p>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.</p> <p><i>Important:</i> Make this ID unique to your Custom Platform. For example, for the s10_ref Reference Platform, the ID is 0x5170.</p>	

You can find these PCIe ID definitions in the PCIe controller instantiated in the `INTELFPGAOCCLSDKROOTboard/s10_ref/hardware/s10gx_ea_htile/board.qsys` Platform Designer System File. These IDs are necessary in the driver and the SDK's programming flow. The kernel driver uses the **Vendor ID**, **Subsystem Vendor ID** and the **Subsystem Device ID** to identify the boards it supports. The SDK's programming flow checks the **Device ID** to ensure that it programs a device with a .aocx Intel FPGA SDK for OpenCL Offline Compiler executable file targeting that specific device.

Table 6. Intel FPGA SDK for OpenCL's Numbering Convention for PCIe Hard IP Device ID

Location in ID	Definition
15:14	RESERVED
13:12	Speed <ul style="list-style-type: none"> 0 — Gen 1 1 — Gen 2 2 — Gen 3 3 — Gen 4
11	RESERVED
10:8	Number of lanes <ul style="list-style-type: none"> 0 — 1 lane 1 — 2 lanes 3 — 4 lanes 4 — 8 lanes 5 — 16 lanes 6 — 32 lanes
7:4	Device family <ul style="list-style-type: none"> 0 — Altera Stratix IV GX 1 — Altera Arria II GX 2 — Stratix II GX 3 — Arria GX 4 — Cyclone IV GX 5 — External 6 — Stratix V 7 — Arria V 8 — Cyclone V 9 — Arria 10 10 — Stratix 10
3	1 — Soft IP (SIP)
continued...	



Location in ID	Definition
	This ID indicates that the PCIe protocol stack is implemented in soft logic. If unspecified, the IP is considered a hard IP.
2:0	Platform Designer PCIe interface type <ul style="list-style-type: none">0 — 64 bits1 — 128 bits2 — 256 bits3 — Desc/Data (that is, Avalon-Streaming (Avalon-ST) interface)4 — Avalon-MM interface

3.1.3 Instantiation of the version_id Component

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

The version ID for the s10_ref Reference Platform is 0xA0C7C1E4.

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

- The PCIe can access the FPGA fabric successfully
- The address map matches the map in the MMD software

Update the VERSION_ID parameter in the version_id component to a new value with every slave addition or removal from the PCIe BAR 4 bus, or whenever the address map changes.

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

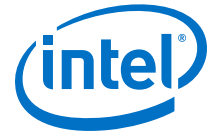
After you build the PCIe component in your hardware design, you need a software layer to communicate with the board via PCIe. To enable communication between the board and the host interface, define the hardware constants for the software in header files.

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

- For Windows systems, the header files are in the *INTELFPGAOCSDKROOT*\board\s10_ref\source\include folder, where *INTELFPGAOCSDKROOT* is the path to the SDK installation.
- For Linux systems, the header files are in the *INTELFPGAOCSDKROOT*/board/s10_ref/linux64/driver directory.

Table 7. Intel Stratix 10 GX FPGA Development Kit Reference Platform Header Files

Header File Name	Description
hw_pcie_constants.h	Header file that defines most of the hardware constants for the board design.
continued...	



Header File Name	Description
	<p>This file includes constants such as the IDs described in <i>PCIe Device Identification Registers</i>, 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.</p> <p>Update the information in this file whenever you change the board design.</p>
hw_pcie_dma.h	<p>Header file that defines DMA-related hardware constants.</p> <ul style="list-style-type: none"> ACL_PCIE_DMA_ONCHIP_RD_FIFO_BASE refers to the Platform Designer address of rd_dts_slave on the PCIe IP's dma_rd_master. ACL_PCIE_DMA_ONCHIP_WR_FIFO_BASE refers to the Platform Designer address of wr_dts_slave on the PCIe IP's dma_rd_master. <p>Update these addresses whenever you change the board design. Refer to the <i>Direct Memory Access</i> section for more information.</p> <ul style="list-style-type: none"> ACL_PCIE_DMA_TABLE_SIZE refers to the DMA descriptor FIFO depth connected to the DMA. When using the internal descriptor controller, refer to the <i>DMA Descriptor Controller Registers</i> section in the <i>Intel Stratix 10 Avalon-MM DMA Interface for PCIe Solutions User Guide</i> for the required size. ACL_PCIE_DMA_PAGES_LOCKED specifies the maximum pages you can lock. You may modify this constant to improve performance. ACL_PCIE_DMA_NON_ALIGNED_TRANS_LOG specifies the starting and ending power-of-two values that non-aligned DMA transfers should have. You may modify this constant to improve performance.

Related Links

- [Direct Memory Access](#) on page 28
- [Device Identification Registers for Intel Stratix 10 PCIe Hard IP](#) on page 24
- [DMA Descriptor Controller Registers](#)

3.1.5 PCIe Kernel Driver for the Intel Stratix 10 GX FPGA Development Kit Reference Platform

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

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

The s10_ref Reference Platform

- For Windows systems, the driver is in the `<path_to_s10pciedk>\windows64\driver` folder.

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

- For Linux, an open-source MMD-compatible kernel driver is in the `<path_to_s10pciedk>/linux64/driver` directory. The table below highlights some of the files that are available in this directory.

**Table 8. Highlights of the Intel Stratix 10 GX FPGA Development Kit Reference Platform's Linux PCIe Kernel Driver Directory**

File	Description
pcie_linux_driver_exports.h	Header file that defines the special commands that the kernel driver supports. The installed kernel driver works as a character device. The basic operations to the driver are <code>open()</code> , <code>close()</code> , <code>read()</code> , and <code>write()</code> . To execute a complicated command, create a variable as an <code>acl_cmd</code> struct type, specify the command with the proper parameters, and then send the command through a <code>read()</code> or <code>write()</code> operation. This header file defines the interface of the kernel driver, which the MMD layer uses to communicate with the device.
aclpci.c	File that implements the Linux kernel driver's basic structures and functions, such as the <code>init</code> , <code>remove</code> , and <code>probe</code> functions, as well as hardware design-specific functions that handle interrupts. For more information on the interrupt handler, refer to the <i>Message Signaled Interrupts</i> section.
aclpci_fileio.c	File that implements the kernel driver's file I/O operations. The kernel driver that is available with the s10_ref Reference Platform supports four file I/O operations: <code>open()</code> , <code>close()</code> , <code>read()</code> , and <code>write()</code> . Implementing these file I/O operations allows the OpenCL user program to access the kernel driver through the file I/O system calls (that is, <code>open</code> , <code>read</code> , <code>write</code> , or <code>close</code>).
aclpci_cmd.c	File that implements the specific commands defined in the <code>pcie_linux_driver_exports.h</code> file. These special commands include <code>SAVE_PCI_CONTROL_REGS</code> , <code>LOAD_PCI_CONTROL_REGS</code> , and <code>GET_PCI_SLOT_INFO</code> .
aclpci_dma.c	File that implements DMA-related routines in the kernel driver. Refer to the <i>Direct Memory Access</i> section for more information.
aclpci_queue.c	File that implements a queue structure for use in the kernel driver to simplify programming.

Related Links

- [aocl install](#) on page 48
- [Message Signaled Interrupt](#) on page 30
- [Direct Memory Access](#) on page 28
- [Jungo Connectivity Ltd. website](#)

3.1.6 Direct Memory Access

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

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



Hardware Considerations

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

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

The `rd_dcm_master` and `wr_dcm_master` ports must connect to the `txs` port. At the end of the DMA transfer, the DMA controller writes the MSI data and the `done` status into the user memory via the `txs` slave. The `txs` slave is part of the PCIe hard IP in `board.qsys`.

All modules that use DMA must connect to the `dma_rd_master` and `dma_wr_master` ports. For DDR4 memory connection, Intel recommends implementing an additional pipeline to connect the two 256-bit PCIe DMA ports to the 512-bit memory slave. For more information, refer to the *DDR4 Connection to PCIe Host* section.

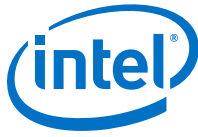
Software Considerations

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

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

Related Links

- [Definitions of Intel Stratix 10 FPGA Development Kit Reference Platform Hardware Constraints in Software Headers Files](#) on page 26
- [Intel Stratix 10 DMA Avalon-MM DMA Interface to the Application Layer](#)
- [DMA Descriptor Controller Registers](#)
- [Implementing a DMA Transfer](#) on page 30
- [DDR4 Connection to PCIe Host](#) on page 33



3.1.6.1 Implementing a DMA Transfer

Implement a DMA transfer in the MMD on Windows (`INTELFPGAOCSDKROOT\board\s10_ref\source\host\mmd\acl_pcie_dma_windows.cpp`) or in the kernel driver on Linux (`INTELFPGAOCSDKROOT/board/s10_ref/linux64/driver/aclpci_dma`).

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

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

To implement a DMA transfer:

1. Verify that the previous DMA transfer sent all the requested bytes of data.
2. Map the virtual memories that are requested for DMA transfer to physical addresses.

Note: The amount of virtual memory that can be mapped at a time is system dependent. Large DMA transfers will require multiple mapping or unmapping operations. For a higher bandwidth, map the virtual memory ahead in a separate thread that is in parallel to the transfer.

3. Set up the DMA descriptor table on local memory.
4. Write the location of the DMA descriptor table, which is in user memory, to the DMA control registers (that is, RC Read Status and Descriptor Base and RC Write Status and Descriptor Base).
5. Write the Platform Designer address of descriptor FIFOs to the DMA control registers (that is EP Read Descriptor FIFO Base and EP Write Status and Descriptor FIFO Base).
6. Write the start signal to the `RD_DMA_LAST_PTR` and `WR_DMA_LAST_PTR` DMA control registers.
7. After the current DMA transfer finishes, repeat the procedure to implement the next DMA transfer.

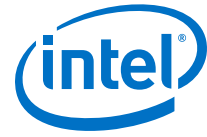
Related Links

[Direct Memory Access](#) on page 28

3.1.7 Message Signaled Interrupt

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

Two different modules generate the signal for the MSI line. The DMA controller in the PCIe hard IP core generates the DMA's MSI. The PCI Express interrupt request (IRQ) module (that is, the `INTELFPGAOCSDKROOT/board/s10_ref/hardware/s10gx_ea_htile/ip/irq_controller` directory) generates the kernel interface's MSI.



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

Hardware Considerations

In `INTELFPGAOCCLSDKROOT/board/s10_ref/hardware/s10gx_ea_htile/board.qsys`, the DMA MSI is connected internally; however, you must connect the kernel interface interrupt manually. For the kernel interface interrupt, the PCI Express IRQ module is instantiated as `pcie_irq_0` in `board.qsys`. The kernel interface interrupts connections are as follows:

- The `kernel_irq_to_host` port from the OpenCL Kernel Interface (`kernel_interface`) connects to the interrupt receiver, which allows the OpenCL kernels to signal the PCI Express IRQ module to send an MSI.
- The PCIe hard IP's `msi_intf` port connects to the `MSI_Interface` port in the PCI Express IRQ module. The kernel interface interrupt receives the MSI address and the data necessary to generate the interrupt via `msi_intf`.
- The `IRQ_Gen_Master` port on the PCI Express IRQ module, which is used to write the MSI, connects to the `txs` port on the PCIe hard IP.
- The `IRQ_Read_Slave` and `IRQ_Mask_Slave` ports connect to the `pipe_stage_host_ctrl` module on Bar 4. After receiving an MSI, the user driver can read the `IRQ_Read_Slave` port to check the status of the kernel interface interrupt, and read the `IRQ_Mask_Slave` port to mask the interrupt.

Software Considerations

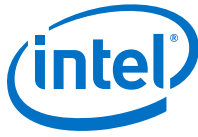
The interrupt service routine in the Linux driver checks which module generates the interrupt. For the DMA's MSI, the driver reads the DMA descriptor table's status bit in local memory, as specified in the *Read DMA Example* section of the *Intel Stratix 10 Avalon-MM DMA Interface for PCIe Solutions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

The interrupt service routine involves the following tasks:

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

Related Links

- [Handling PCIe Interrupts](#)
- [Read DMA Example](#)



3.1.8 Cable Autodetect

The Intel Stratix 10 GX FPGA Development Kit Reference Platform automatically tries to detect the cable by default when programming the FPGA via the Intel FPGA Download Cable.

You can set the `ACL_PCIE_JTAG_CABLE` or `ACL_PCIE_JTAG_DEVICE_INDEX` environment variables to disable the auto-detect feature and use values that you define.

Cable autodetect is useful when you have multiple devices connected to a single host.

The memory-mapped device (MMD) uses in-system sources and probes to identify the cable connected to the target board. You must instantiate the `cade_id` register block and connect it to Bar 4 with the correct address map. You must also instantiate `board_in_system_sources_probes_cade_id`, which is an in-system sources and probe component, and connect it to `cade_id` register.

The MMD must be updated to take in the relevant changes. Add the `scripts/find_jtag_cable.tcl` script to be added to your custom platform.

When the FPGA is being programmed via the Intel FPGA Download Cable, the MMD invokes `quartus_stp` to execute the `find_jtag_cable.tcl` script. The script identifies the cable and index number which is then used to program the FPGA through the `quartus_pgm` command.

3.2 DDR4 as Global Memory for OpenCL Applications

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

In the current version of the `s10_ref` Reference Platform, all Platform Designer components related to the DDR4 global memory are now part of the `INTELFPGAOCSDKROOT/board/s10_ref/hardware/s10gx_ea_htile/acl_ddr4_s10.qsys` Platform Designer subsystem within `board.qsys`. In addition, the location of the clock domain crossings has changed to increase the number of blocks operating in the slower PCIe domain. With this modified structure, you can add multiple memories with different clock domains to the system.

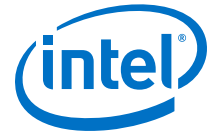
If you have a Custom Platform that is ported from a previous version of the `s10_ref` Reference Platform, you have the option to modify your Custom Platform as described above. This modification is not mandatory.

Dependencies

DDR4 external memory interfaces

For more information on the DDR4 external memory interface IP, refer to the *DDR3 Board Design Guidelines* and *DDR4 Board Design Guidelines* sections in *Intel Stratix 10 External Memory Interfaces IP User Guide*.

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



Related Links

- [DDR3 Board Design Guidelines](#)
- [DDR4 Board Design Guidelines](#)

3.2.1 DDR4 IP Instantiation

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

Table 9. DDR4 SDRAM Controller IP Configuration Settings

IP Parameter	Configuration Setting
Timing Parameters	As per the computing card's data specifications.
Avalon Width Power of 2	Currently, OpenCL does not support non-power-of-2 bus widths. As a result, the s10_ref Reference Platform uses the option that forces the DDR4 controller to power of 2. Use the additional pins of this x72 core for error checking between the memory controller and the physical module.
Byte Enable Support	Enabled Byte enable support is necessary in the core because the Intel FPGA SDK for OpenCL requires byte-level granularity to all memories.
Performance	Enabling the reordering of DDR4 memory accesses and a deeper command queue look-ahead depth might provide increased bandwidth for some OpenCL kernels. For a target application, adjust these and other parameters as necessary. <i>Note:</i> Increasing the command queue look-ahead depth allows the DDR4 memory controller to reorder more memory accesses to increase efficiency, which improves overall memory throughput.
Debug	Disabled for production.

3.2.2 DDR4 Connection to PCIe Host

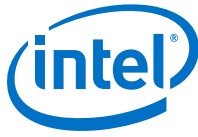
Connect all global memory systems in the Intel Stratix 10 GX FPGA Development Kit Reference Platform to the host via the OpenCL Memory Bank Divider component.

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

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

Important: Instruct the host to verify the successful calibration of the memory controller.

The `INTELFPGAOCCLSDKROOT/board/s10_ref/hardware/s10gx_ea_htile/board.qsys` Platform Designer system uses a custom UniPHY Status to AVS IP component to aggregate different UniPHY status conduits into a single Avalon slave port named `s`. This slave port connects to the `pipe_stage_host_ctrl` component so that the PCIe host can access it.



Related Links

[OpenCL Memory Bank Divider](#)

3.2.3 DDR4 Connection to the OpenCL Kernel

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

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

3.3 Host Connection to OpenCL Kernels

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

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

3.4 Intel Stratix 10 FPGA System Design

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

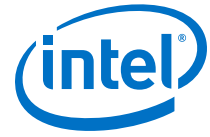
Examples of design complexities:

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

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

3.4.1 Clocks

Several clock domains affect the Platform Designer hardware system of the Intel Stratix 10 GX FPGA Development Kit Reference Platform.



These clock domains include:

- 125 MHz PCIe clock
- 233 MHz DDR4 clock
- 50 MHz general clock (`config_clk`)
- Kernel clock that can have any clock frequency

With the exception of the kernel clock, the `s10_ref` Reference Platform is responsible for the timing closure of these clocks. However, because the board design must clock cross all interfaces in the kernel clock domain, the board design also has logic in the kernel clock domain. It is crucial that this logic is minimal and achieves an F_{\max} higher than typical kernel performance.

Related Links

[Guaranteed Timing Closure of the Intel Stratix 10 GX FPGA Development Kit Reference Platform Design](#) on page 39

3.4.2 Resets

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

These reset drivers include:

- The `por_reset_counter` in the `INTELFPGAOCSDKROOT/board/s10_ref/hardware/s10gx_ea_htile/board.qsys` Platform Designer system implements the power-on-reset. The power-on-reset resets all the hardware on the device by issuing a reset for a number of cycles after the FPGA completes configuration.
- The PCIe bus issues a `perst` reset that resets all hardware on the device.
- The OpenCL Kernel Interface component issues the `kernel_reset` that resets all logic in the kernel clock domain.

The power-on-reset and the `perst` reset are combined into a single `global_reset`; therefore, there are only two reset sources in the system (that is, `global_reset` and `kernel_reset`). However, these resets are explicitly synchronized across the various clock domains, resulting in several reset interfaces.

Important Considerations Regarding Resets

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

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

- Resets and clocks must work together to propagate reset to all logic.

Resetting a circuit in a given clock domain involves asserting the reset over a number of clock cycles. However, your design may apply resets to the PLLs that generate the clocks for a given clock domain. This means a clock domain can hold in reset without receiving the clock edge that is necessary for synchronous resets. In addition, a clock holding in reset might prevent the propagation of a reset signal because it is synchronized to and from that clock domain. Avoid such situations by ensuring that your design satisfies the following criteria:

 - Generate the `global_reset` signal off the free-running `config_clk`.
 - The `ddr4_calibrate` IP resets the External Memory Interface controller separately.
- Apply resets to both reset interfaces of a clock-crossing bridge or FIFO component.

FIFO content corruption might occur if only part of a clock-crossing bridge or a dual-clock FIFO component is reset. These components typically provide a reset input for each clock domain; therefore, reset both interfaces or none at all. For example, in the `s10_ref` Reference Platform, `kernel_reset` resets all the kernel clock-crossing bridges between DDR on both the `m0_reset` and `s0_reset` interfaces.

3.4.3 Floorplan

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

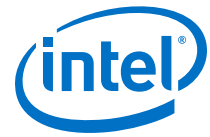
Dependencies

- Chip Planner
- Logic Lock Plus regions

Intel performed the following tasks iteratively to derive the floorplan of the `s10_ref` Reference Platform:

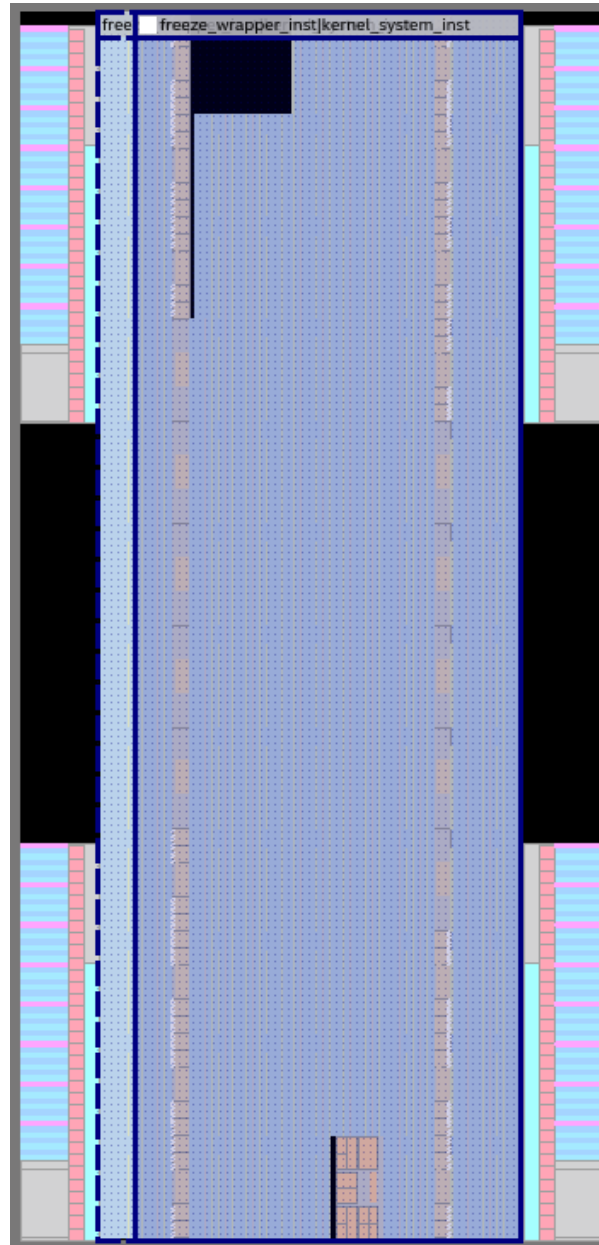
1. Compile a design without any region or floorplanning constraints.

Intel recommends that you compile the design with several seeds.
2. Examine the placement of the IP cores (for example, PCIe, DDR4, Avalon interconnect pipeline stages and adapters) for candidate locations, as determined by the Intel Quartus Prime Pro Edition software's Fitter. In particular, Intel recommends examining the seeds that meet or almost meet the timing constraints.



For the s10_ref Reference Platform, the PCIe I/O is located in the lower left corner of the Intel Stratix 10 FPGA. The DDR4 I/O is located on the top part of the left I/O column of the device. Because the placements of the PCIe and DDR4 IP components tend to be close to the locations of their respective I/Os, you can apply Logic Lock Plus regions to constrain the IP components to those candidate regions.

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



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

You must create a dedicated Logic Lock Plus region for the OpenCL kernel system. Furthermore, do not place kernel logic in the board's Logic Lock Plus regions (that is, static region). As shown in [Figure 2](#) on page 37, the logic for the `boardtest.cl` OpenCL kernel, that is, the scatter area, can be placed anywhere except within the seven Logic Lock Plus regions.

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

- The size of a Logic Lock Plus region should be just large enough to contain the board logic and to meet timing constraints of the board clocks. Oversized Logic Lock Plus regions consume FPGA resources unnecessarily.
- Avoid creating tightly-packed Logic Lock Plus regions that cause very high logic utilization and high routing congestion.

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

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

Refer to the *Pipelining* section for more information.

Related Links

- [Pipelining](#) on page 38
- [Designing with Logic Lock Regions](#)

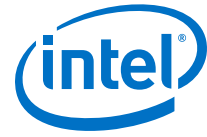
3.4.4 Global Routing

FPGAs have dedicated clock trees that distribute high fan-out signals to various sections of the devices. In the FPGA system that the Intel Stratix 10 FPGA Development Kit Reference Platform targets, global routing can distribute high fan-out signals regionally or globally. Regional distribution applies across any quadrant of the device. Global distribution applies across the entire device.

There is no restriction on the placement location of the OpenCL kernel on the device. As a result, the kernel clocks and kernel reset must distribute high fan-out signals globally.

3.4.5 Pipelining

You must manually insert pipelines throughout the FPGA system.



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

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

Examples of Pipeline Implementation

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

The DMA at the bottom of the FPGA must connect to the DDR4 memory at the top of the FPGA. To achieve timing closure of the board interface logic at a DDR4 clock speed of 233 MHz, additional pipeline stages between the OpenCL Memory Bank Divider component and the DDR4 controller IP are necessary. In the Intel Stratix 10 GX FPGA Development Kit Reference Platform's board.qsys Platform Designer system, the pipeline stages are named `pipe_stage_ddr4a_dimm_*`.

The middle pipeline stage, `pipe_stage_ddr4a_dimm`, combines both the direct kernel DDR4 accesses and the accesses through the OpenCL Memory Bank Divider. The multistage pipeline approach ensures that the kernel entry point to the pipeline is geared towards neither the OpenCL Memory Bank Divider, which is close to the PCIe IP core, nor the DDR4 IP core, which is at the very top of the FPGA.

3.4.6 DDR4 Calibration

The Intel Stratix 10 GX FPGA Development Kit Reference Platform includes special mechanisms to ensure the functional stability of the Intel Stratix 10 silicon. For example, the DDR4 memory might not calibrate successfully after FPGA reconfiguration. The driver within the `s10_ref` Reference Platform can detect a failed calibration via the Uniphy Status to AVS IP, and retrigger calibration through the `ddr4_calibrate` IP block.

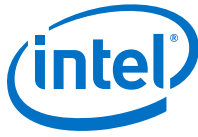
3.5 Guaranteed Timing Closure of the Intel Stratix 10 GX FPGA Development Kit Reference Platform Design

One of the key features of the Intel FPGA SDK for OpenCL is that it abstracts away hardware details, such as timing closure, for software developers. Both the SDK and the Custom Platform contribute to the implementation of the SDK's guaranteed timing closure feature.

The SDK provides the IP to generate the kernel clock, and a post-flow script that ensures this clock is configured with a safe operating frequency confirmed by timing analysis. The Custom Platform developer imports a post-fit netlist that has already achieved timing closure on all non-kernel clocks.

3.5.1 Supply the Kernel Clock

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



The **REF_CLK_RATE** parameter specifies the frequency of the reference clock that connects to the kernel PLL (`pll_refclk`). For the `s10_ref` Reference Platform, the **REF_CLK_RATE** frequency is 50 MHz.

The **KERNEL_TARGET_CLOCK_RATE** parameter specifies the frequency that the Intel Quartus Prime Pro Edition software attempts to achieve during compilation. The board hardware contains some logic that the kernel clock clocks. At a minimum, the board hardware includes the clock crossing hardware. To prevent this logic from limiting the Fmax achievable by a kernel, the **KERNEL_TARGET_CLOCK_RATE** must be higher than the frequency that a simple kernel can achieve on your device. For the Intel Stratix 10 GX FPGA Development Kit that the `s10_ref` Reference Platform targets, the **KERNEL_TARGET_CLOCK_RATE** is 1 GHz.

3.5.2 Guarantee Kernel Clock Timing

The Intel Quartus Prime database interface executable (`quartus_cdb`) runs a script after every Intel Quartus Prime Pro Edition software compilation as a post-flow script. In the Intel Stratix 10 GX FPGA Development Kit Reference Platform, the OpenCL Kernel Clock Generator component works together with the post-flow script to guarantee kernel clock timing.

In the import revision compilation, the compilation script `import_compile.tcl` invokes the `INTELFPGAOCSDKROOT/board/s10_ref/hardware/s10gx_ea_htile/scripts/post_flow.tcl` Tcl script in the `s10_ref` Reference Platform after every Intel Quartus Prime Pro Edition software compilation using `quartus_cdb`.

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

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

3.5.3 Provide a Timing-Closed Post-Fit Netlist

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

Dependencies

Intel Quartus Prime Pro Edition compiler

Intel Quartus Prime software provides several mechanisms for preserving the placement and routing of some previously compiled logic and importing this logic into a new compilation. For Intel Stratix 10 devices, the previously compiled logic is imported into the compilation flow.

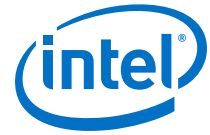


Figure 3. Custom Platform Development Flow and Hand-Off between Board Developer and SDK End User

The board developer is responsible for porting the s10_ref Reference Platform to their own board, closing timing, and locking down the static part of the board.

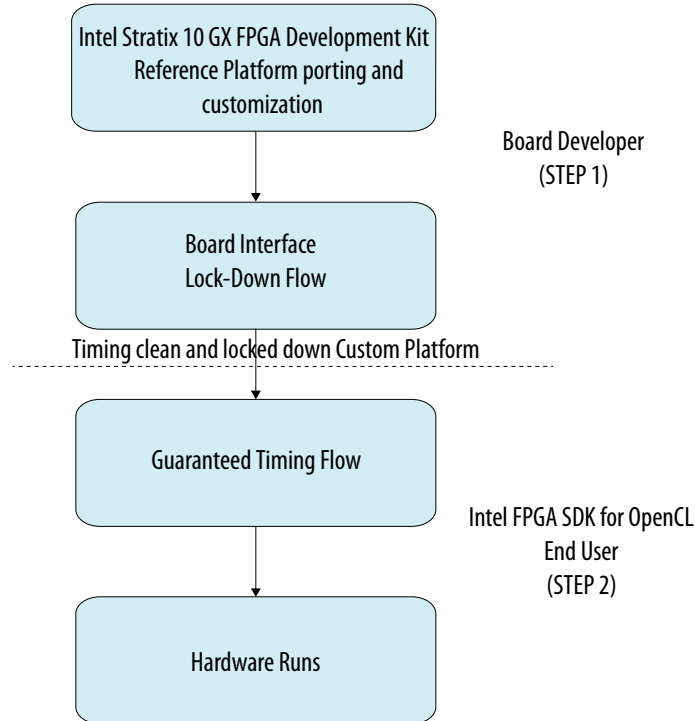
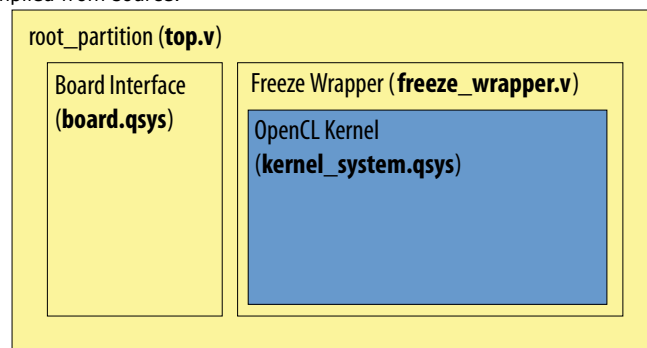


Figure 4. Structure of the Hierarchy for the OpenCL Hardware System on the Intel Stratix 10 Device

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



The Intel Quartus Prime Pro Edition compilation flow can preserve the placement and routing of the board interface partition via the exported Intel Quartus Prime Archive File. `base.qar` and `root_partition.qdb` files contain all the database files for the base compilation of `root_partition`. The `s10_ref` Reference Platform is configured with the project revisions and partitioning that are necessary to implement the compilation flow. By default, the SDK invokes the Intel Quartus Prime Pro Edition



software on the top revision. This revision is configured to import and restore the `base.qdb` file, which has been precompiled and exported from a base revision compilation.

When developing your Custom Platform from the `s10_ref` Reference Platform, it is essential to maintain the `flat.qsf`, `base.qsf`, `top.qsf`, and `top_synth.qsf` Intel Quartus Prime Settings Files.

The `s10_ref` Reference Platform includes two additional partitions: the `Top` partition and the `kernel` partition. The `Top` partition contains all logic, and the `kernel` partition contains the logic.

Related Links

[Generating the base.qar Post-Fit Netlist for Your Intel Stratix 10 Custom Platform](#) on page 20

3.6 Intel Quartus Prime Compilation Flow and Scripts

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

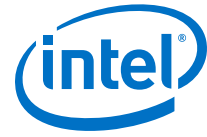
Invoke the Intel Quartus Prime compilation flow by calling the following `quartus_sh` executables:

- The board developer runs the `quartus_sh --flow compile top -c base` command to execute the base revision compilation. This compilation closes timing, locks down the static region, and generates the `base.qdb` file.
- The user of the Intel Stratix 10 FPGA Development Kit Reference Platform or a Custom Platform runs the `quartus_sh -t import_compile.tcl` command to execute the import revision compilation. This compilation generates programming files that are guaranteed to be timing closed.

3.6.1 Intel Quartus Prime Compilation Flow for Board Developers

The `quartus_sh --flow compile top -c base` command executes the Intel Quartus Prime compilation flow that generates a `base.sof` full-chip JTAG programming file within the `.aocx` file.

The script performs the necessary tasks to ensure that the import revision compilations are timing-closed.



Running the `quartus_sh --flow compile top -c base` command executes the following tasks:

- Runs `quartus_syn` to execute the Analysis and Synthesis stage of the Intel Quartus Prime compilation flow.
- Runs `quartus_fit` to execute the Place and Route stage of the Intel Quartus Prime compilation flow.
- Runs `quartus_sta` to execute the Static Timing Analysis stage of the Intel Quartus Prime compilation flow.
- Runs the `INTELFPGAOCSDKROOT/board/sl0_ref/hardware/sl0gx_ea_hfile/scripts/post_flow_pr.tcl` file.

The `post_flow_pr.tcl` script determines the maximum frequency at which the OpenCL kernel can run and generates the corresponding PLL settings. The script then reruns static timing analysis. The script also exports the compilation database of the base revision compilation results as a forward-compatible Partition Database File (`.qdb`). Refer to the *QDB File Generation* section for more information.

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

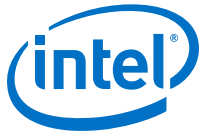
The `.aocx` file that the base revision compilation flow generates only contains the `.sof` full-chip programming file. The Intel FPGA SDK for OpenCL `program` utility automatically uses JTAG programming when it programs with a `.aocx` file from the base revision compilation. Only the import revision compilation flow, executed by the SDK user, generates a `.aocx` file.

Related Links

- [Platform Designer System Generation](#) on page 45
- [QDB File Generation](#) on page 45

3.6.2 Intel Quartus Prime Compilation Flow for Custom Platform Users

The `import_compile.tcl` script executes the Intel Quartus Prime compilation flow that generates a `top.sof` full-chip JTAG programming file within the `.aocx` file.



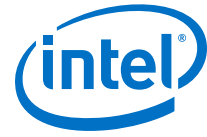
The `import_compile.tcl` script executes the following tasks:

- Runs the `INTELFPGAOCCLSDKROOT/board/s10_ref/hardware/s10gx_ea_htile/scripts/pre_flow_pr.tcl` file. The `pre_flow_pr.tcl` script generates the `board.qsys` and the `kernel_system.qsys` Platform Designer System Files.
Refer to the *Platform Designer System Generation* section for more information.
- Imports the base revision compilation results as a `.qdb` file.
Refer to the *QDB File Generation* section for more information.
- Runs `quartus_fit` and `quartus_asm` to verify that the `.qdb` file is forward compatible.
- Runs `quartus_syn` to execute the Analysis and Synthesis stage of the Intel Quartus Prime compilation flow for the kernel partition only.
- Runs `quartus_fit` to execute the Place and Route stage of the Intel Quartus Prime compilation flow for the entire design.
- Runs `quartus_sta` to execute the static timing analysis stage of the Intel Quartus Prime compilation flow.
- Runs the `INTELFPGAOCCLSDKROOT/board/s10_ref/hardware/s10gx_ea_htile/scripts/post_flow_pr.tcl` file. The `post_flow_pr.tcl` script determines the maximum frequency at which the OpenCL kernel can run and generates the corresponding PLL settings. The script then reruns the static timing analysis.
- Runs `quartus_asm` to generate the full-chip programming files for the base revision.
- Runs `quartus_asm` to generate the full-chip programming files for the import revision.
- Generates the `fpga.bin` file, which contains the `top.sof` full-chip programming file.

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

Related Links

- [Platform Designer System Generation](#) on page 45
- [QDB File Generation](#) on page 45



3.6.3 Platform Designer System Generation

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

The `board.qsys` Platform Designer system represents the bulk of the static region. The `pre_flow_pr.tcl` script generates both Platform Designer systems on the fly before the beginning of the Intel Quartus Prime compilation flow in both the base and import revision compilations.

3.6.4 QDB File Generation

The `base.qdb` Intel Quartus Prime Compilation Database File contains all the necessary compilation database information for importing a timing-closed and placed-and-routed netlist of the static region.

The `INTELFPGAOCCLSDKROOT/board/s10_ref/hardware/s10gx_ea_htile/scripts/post_flow_pr.tcl` script creates the `root_partition.qdb` file. The `.tcl` file invokes the `export_design` command to export the entire base revision compilation database to the `base.qar` file that also contains the `root_partition.qdb` and `pr_base.id` files. For your Custom Platform, you do not need to add the `root_partition.qdb` and `pr_base.id` files to the board directory (that is, `INTELFPGAOCCLSDKROOT/board/<custom_platform>/hardware/<board_name>`) separately.

3.7 Addition of Timing Constraints

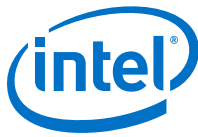
A Custom Platform must apply the correct timing constraints to the Intel Quartus Prime project. In the Intel Stratix 10 FPGA Development Kit Reference Platform, the `top.sdc` file contains all timing constraints applicable before IP instantiation in Platform Designer. The `top_post.sdc` file contains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

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

One noteworthy constraint in the `s10_ref` Reference Platform is the multicycle constraint for the kernel reset in the `top_post.sdc` file. Using global routing saves routing resources and provides more balanced skew. However, the delay across the global route might cause recovery timing issues that limit kernel clock speed. Therefore, it is necessary to include a multicycle path on the global reset signal.

3.8 Connection of the Intel Stratix 10 GX FPGA Development Kit Reference Platform to the Intel FPGA SDK for OpenCL

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



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

3.8.1 Describe the Intel Stratix 10 GX FPGA Development Kit Reference Platform to the Intel FPGA SDK for OpenCL

The `INTELFPGAOCSDKROOT/board/s10_ref/board_env.xml` file describes the Intel Stratix 10 GX FPGA Development Kit Reference Platform to the Intel FPGA SDK for OpenCL. Details of each field in the `board_env.xml` file are available in the *Creating the board_env.xml File* section of the *Intel FPGA SDK for OpenCL Custom Platform Toolkit User Guide*.

In the `s10_ref` Reference Platform, Intel uses the `bin` folder for Windows dynamic link libraries (DLLs), the `lib` directory for delivering libraries, and the `libexec` directory for delivering the SDK utility executables. This directory structure allows the `PATH` environment variable to point to the location of the DLLs (that is, `bin`) in isolation of the SDK utility executables.

Related Links

[Creating the board_env.xml File](#)

3.8.2 Describe the Intel Stratix 10 GX FPGA Development Kit Reference Platform Hardware to the Intel FPGA SDK for OpenCL

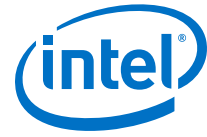
The Intel Stratix 10 GX FPGA Development Kit Reference Platform includes an `INTELFPGAOCSDKROOT/board/s10_ref/hardware/s10gx_ea_htile/board_spec.xml` file that describes the hardware to the Intel FPGA SDK for OpenCL.

Device

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

```
+-----+
-+
; Fitter Partition
Statistics                                     ;
+-----+
+-----+
; Statistic           ; 1           ; freeze_wrapper_inst|
kernel_system_inst   ;
+-----+
+-----+
; ALMs needed [=A-B+C] ; 0 / 427200 (0%) ; 0 / 385220
(0%)                  ;
```

The value of `used_resources` equals the total number of ALMs in `1` minus the total number of ALMs in `freeze_wrapper_inst|kernel_system_inst`. In the example above, `used_resources = 427200 - 385220 = 41980` ALMs.



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

Global Memory

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

- `DDR`

Because DDR memory serves as the default memory for the board that the `s10_ref` Reference Platform targets, its `address` attribute begins at zero. Its `config_addr` is `0x018` to match the `memory` conduit used to connect to the corresponding OpenCL Memory Bank Divider for DDR.

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

Interfaces

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

3.9 Intel Stratix 10 FPGA Programming Flow

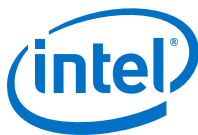
There are three ways to program the Intel Stratix 10 FPGA for the Intel Stratix 10 GX FPGA Development Kit Reference Platform: Flash and `quartus_pgm`

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

- To replace both the FPGA periphery and the core while maintaining the programmed state after power cycling, use Flash programming.
- To replace both the FPGA periphery and the core, use the Intel Quartus Prime Programmer command-line executable (`quartus_pgm`) to program the device via cables such as the Intel FPGA Download Cable (formerly USB-Blaster).

3.9.1 Define the Contents of the `fpga.bin` File for the Intel Stratix 10 GX FPGA Development Kit Reference Platform

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

**Table 10. Contents of the Intel Stratix 10 GX FPGA Development Reference Platform's fpga.bin File**

Field	Description
.acl.sof	The full programming bitstream for the compiled design. This section appears in the fpga.bin files generated from both the base revision and the import revision compilations.

3.10 Host-to-Device MMD Software Implementation

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

The source codes of an MMD library that demonstrates good performance are available in the `INTELFPGAOCCLSDKROOT/board/sl0_ref/source/host/mmd` directory. Refer to the *Host-to-Device MMD Software Implementation* section in the *Stratix V Network Reference Platform Porting Guide* for more information.

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

Related Links

- [Host-to-Device MMD Software Implementation](#)
- [MMD API Descriptions](#)

3.11 Implementation of Intel FPGA SDK for OpenCL Utilities

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

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

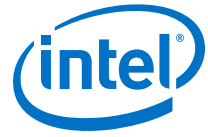
Related Links

[Providing Intel FPGA SDK for OpenCL Utilities Support](#)

3.11.1 aocl install

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

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.

3.11.2 aocl uninstall

The `uninstall <path_to_customplatform>` utility in the Intel Stratix 10 GX FPGA Development Kit Reference Platform removes the current host computer drivers used for communicating with the board.

Windows

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

Linux

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

3.11.3 aocl program

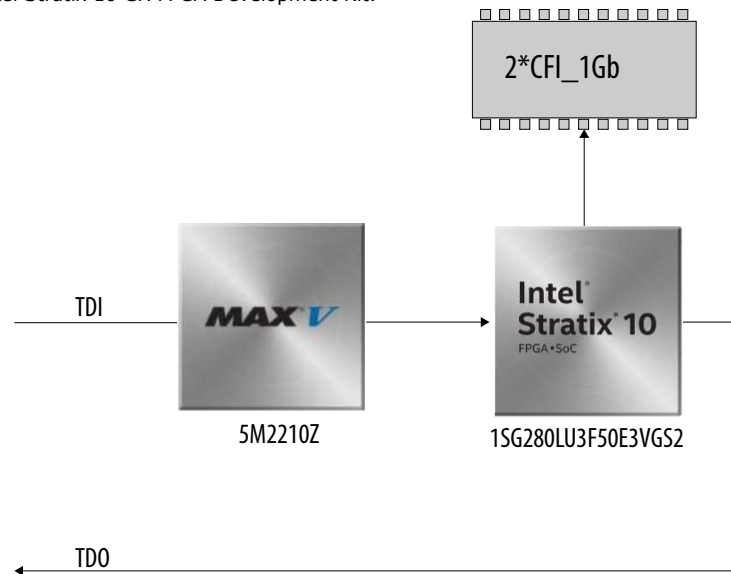
The `program` utility in the Intel Stratix 10 GX FPGA Development Kit Reference Platform programs the board with the specified `.aocx` file. Calling the `aocl_mmd_reprogram()` MMD API function implements the `program` utility.

3.11.4 aocl flash

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

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

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



3.11.5 aocl diagnose

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

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

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`).

3.11.6 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.

3.12 Intel Stratix 10 FPGA Development Kit Reference Platform Scripts

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

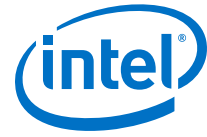


Table 11. Tcl Scripts within the INTELFGAOCLSDKROOT/board/s10_ref/hardware/s10gx_ea_htile/scripts Directory

Script	Description
create_fpga_bin_pr.tcl	Creates the ELF binary file, fpga.bin, from the .sof file, the .rbf file, and the pr_base.id file.
post_flow_pr.tcl	This script runs after every Intel Quartus Prime Pro Edition software compilation. It facilitates the guaranteed timing flow by setting the kernel clock PLL, generating a small report in the acl_quartus_report.txt file, and rerunning STA with the modified kernel clock settings.
pre_flow_pr.tcl	This script generates the RTL of the top-level board.qsys Platform Designer system for the static region.
adjust_plls_s10.tcl	PLL adjustment script for the kernel clock PLL to guarantee timing closure on the kernel clock, by setting it to the maximum allowed frequency.
qar_ip_files.tcl	Tcl script that packages up base.qdb and pr_base.id during base revision compile.
helpers.tcl	Tcl script with helper functions used by qar_ip_files.tcl
device.tcl	Tcl file that is included in all revisions and contains all device-specific information (for example, device family, ordering part number (OPN), voltage settings, etc.)
scripts/kernel_system_update.tcl	Tcl script to update the kernel system.

Related Links

[Contents of the Intel Stratix 10 GX FPGA Development Kit Reference Platform](#) on page 7

3.13 Considerations in Intel Stratix 10 GX FPGA Development Kit Reference Platform Implementation

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

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

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

- Intel Quartus Prime compilation is only ever performed after the Intel FPGA SDK for OpenCL Offline Compiler embeds an OpenCL kernel inside the system.
- Perform Intel Quartus Prime compilation after you install the Intel FPGA SDK for OpenCL and set the INTELFGAOCLSDKROOT environment variable to point to the SDK installation.
- 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.
- The PATH or LD_LIBRARY_PATH environment variable must point to the MMD library in the Custom Platform.



4 Document Revision History

Table 12. Document Revision History of the Intel FPGA SDK for OpenCL Intel Stratix 10 GX FPGA Development Kit Reference Platform Porting Guide

Date	Version	Changes
November 2017	17.1	Initial release.

Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Cyclone, Enpirion, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

*Other names and brands may be claimed as the property of others.

**ISO
9001:2008
Registered**