OpenCL* on Intel® Programmable Acceleration Card with Intel® Arria® 10 GX FPGA Quick Start User Guide

Updated for Intel® Acceleration Stack for Intel® Xeon® CPU with FPGAs: 1.2
## Contents

1. About this Document ................................................................. 3  
   1.1. Conventions ................................................................. 3  
   1.2. Acceleration Glossary .................................................... 3  
   1.3. Acronyms ................................................................. 3  

2. Introduction ............................................................................. 5  
   2.1. Release Content ......................................................... 5  

3. Setting Up the Host Machine .................................................... 6  
   3.1. Installing the Release ..................................................... 6  
   3.2. Initializing the Intel Acceleration Stack for OpenCL ......................... 7  

4. Running Diagnostics ................................................................. 9  

5. OpenCL Support for Multi-Card Systems ........................................... 11  

6. Running Samples ...................................................................... 13  
   6.1. Running Hello World ................................................... 13  
   6.2. Running Vector Add ...................................................... 14  

7. Compiling OpenCL Kernels .......................................................... 16  
   7.1. Checking Timing Results .............................................. 16  

8. Running an OpenCL Design Example ............................................ 18  
   8.1. Downloading an OpenCL Design Example ................................. 18  
   8.2. Compiling the Kernel ................................................... 20  
   8.3. Compiling the Host ...................................................... 20  
   8.4. Running the Executable ................................................ 21  


A. Disabling Non-Uniform Memory Access (NUMA) and DMA Worker Threads to Optimize PCIe Bandwidth ................................................. 24
1. About this Document

1.1. Conventions

Table 1. Document Conventions

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#</code></td>
<td>Precedes a command that indicates the command is to be entered as root.</td>
</tr>
<tr>
<td><code>$</code></td>
<td>Indicates a command is to be entered as a user.</td>
</tr>
<tr>
<td>This font</td>
<td>Filenames, commands, and keywords are printed in this font. Long command lines are printed in this font. Although long command lines may wrap to the next line, the return is not part of the command; do not press enter.</td>
</tr>
<tr>
<td><code>&lt;variable_name&gt;</code></td>
<td>Indicates the placeholder text that appears between the angle brackets must be replaced with an appropriate value. Do not enter the angle brackets.</td>
</tr>
</tbody>
</table>

1.2. Acceleration Glossary

Table 2. Acceleration Stack for Intel® Xeon® CPU with FPGAs Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Acceleration Stack for Intel Xeon® CPU with FPGAs</td>
<td>Acceleration Stack</td>
<td>A collection of software, firmware, and tools that provides performance-optimized connectivity between an Intel FPGA and an Intel Xeon processor.</td>
</tr>
<tr>
<td>Intel FPGA Programmable Acceleration Card (Intel FPGA PAC)</td>
<td>Intel FPGA PAC</td>
<td>PCIe® FPGA accelerator card. Contains an FPGA Interface Manager (FIM) that pairs with an Intel Xeon processor over the PCIe bus.</td>
</tr>
</tbody>
</table>

1.3. Acronyms

Table 3. Acronyms

<table>
<thead>
<tr>
<th>Acronyms</th>
<th>Expansion</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFU</td>
<td>Accelerator Functional Unit</td>
<td>Hardware Accelerator implemented in FPGA logic which offloads a computational operation for an application from the CPU to improve performance.</td>
</tr>
<tr>
<td>AF</td>
<td>Accelerator Function</td>
<td>Compiled Hardware Accelerator image implemented in FPGA logic that accelerates an application.</td>
</tr>
<tr>
<td>Acronyms</td>
<td>Expansion</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
<td>A set of subroutine definitions, protocols, and tools for building software applications.</td>
</tr>
<tr>
<td>FIM</td>
<td>FPGA Interface Manager</td>
<td>The FPGA hardware containing the FPGA Interface Unit (FIU) and external interfaces for memory, networking, etc. The Accelerator Function (AF) interfaces with the FIM at run time.</td>
</tr>
<tr>
<td>OPAE</td>
<td>Open Programmable Acceleration Engine</td>
<td>The OPAE is a software framework for managing and accessing AFs.</td>
</tr>
</tbody>
</table>
2. Introduction

This user guide describes how to get started with the OpenCL* on the Intel PAC with Intel Arria® 10 GX FPGA 1.2 Release. The instructions use the precompiled OpenCL kernels included in this 1.2 Release. This user guide also includes a brief introduction to compiling OpenCL kernels.

OpenCL designs comprise two components, the kernel and the host. The kernel includes the accelerator code. The host runs on the host machine. The accelerator card plugs into the host machine.

Note: You must have root permission on the host machine to setup OpenCL.

Related Information

- Intel FPGA SDK for Open Computing Language (OpenCL) web-page
- Intel FPGA SDK for OpenCL Pro Edition Getting Started Guide
- OpenCL on the Intel PAC with Intel Arria 10 GX FPGA Quick Start User Guide

2.1. Release Content

The release available under $OPAE_PLATFORM_ROOT includes the files for the Intel PAC with Intel Arria 10 GX FPGA 1.2 Release. The release includes the following files for OpenCL located in the $OPAE_PLATFORM_ROOT/opencl folder:

- 1.2 OpenCL Board Support Package (BSP):
  - opencl_bsp
- OpenCL example designs tested with:
  - exm_opencl_hello_world_x64_linux.tgz
  - exm_opencl_vector_add_x64_linux.tgz
- Pre-compiled kernels <aocx>:
  - hello_world.aocx
  - vector_add.aocx

Related Information

Understanding the Extracted Intel PAC with Intel Arria 10 GX FPGA Release Package
3. Setting Up the Host Machine

**Prerequisites:** Before running OpenCL, you must follow the instructions from the *Getting Started* section of the *Intel Acceleration Stack Quick Start Guide for Intel Programmable Acceleration Card with Arria 10 GX FPGA*, referred to as *Quick Start Guide* throughout this document.

**Attention:**
- If you need the OpenCL compiler and tools to build and run OpenCL AFUs, download and install the Intel Acceleration Stack for Development. Installing the development software ensures that the OpenCL SDK is available under `/home/<username>/inteldevstack/` or a Custom Directory, `/<custom Directory>`. This user guide refers to this path as `/<dev Install Path>.
- If you only require the Intel FPGA SDK for the OpenCL deployment functionality, download and install the Intel Acceleration Stack for Runtime. Installing the runtime environment ensures that the OpenCL RTE is installed under `/home/<username>/intelrtestack/` or a Custom Directory, `/<custom Directory>`. This user guide refers to this path as `/<RTE Install Path>.
- Do not install the RTE and the DEV on the same host system. The DEV already contains the RTE.

**Related Information**
- *Intel Acceleration Stack Quick Start Guide for Intel Programmable Acceleration Card with Intel Arria 10 GX FPGA*
- *Getting Started*
- *Installing the Intel Acceleration Stack Development Package on the Host Machine*
- *Installing the Intel Acceleration Stack Runtime Package on the Host Machine*
- *Intel Acceleration Stack Quick Start Guide for Intel FPGA Programmable Acceleration Card D5005*
- *Installing the Runtime Package on the Host Machine*
- *Installing the Development Package on the Host Machine*

3.1. Installing the Release

Follow the installation instructions from the *Quick Start Guide* to set up the Intel PAC with Intel Arria 10 GX FPGA.

You can run the OPAE software in a non-virtualized environment with the Single Root I/O Virtualization (SR-IOV) disabled or in a virtualized environment with the SR-IOV enabled.
To run the OpenCL reference design in a virtualized environment that includes SR-IOV, complete the following additional steps:

1. Program the required OpenCL configuration from the host machine by typing the following command:
   ```
aocl program <device name> <filename>
   ```
   
   **Note:** The 1.2 Release does not allow partial reconfiguration in virtualized environment.

2. Enable virtualization using the instructions from section *Updating Settings Required for VFs* and section *Configuring the VF Port on the Host* of the *Quick Start Guide*.

3. Set the `CL_CONTEXT_COMPILER_MODE_INTELFPGA` environment variable in the virtual machine to disable FPGA configuration or reconfiguration during OpenCL host runtime:
   ```
   export CL_CONTEXT_COMPILER_MODE_INTELFPGA=3
   ```

4. Run the required application from the virtual machine.

5. Disable virtualization using the instruction from section *Disconnecting the VF from the VM and Reconnecting to the PF* of the *Quick Start Guide*.

**Related Information**

- Intel Acceleration Stack Quick Start Guide for Intel Programmable Acceleration Card with Intel Arria 10 GX FPGA
- Updating Settings Required for VFs
- Configuring the VF Port on the Host
- Disconnecting the VF from the VM and Reconnecting to the PF

### 3.2. Initializing the Intel Acceleration Stack for OpenCL

The `init_env.sh` script performs all the initialization and setup for the Acceleration Stack for OpenCL. The script is available in either `/<RTE install path>/` or `/<DEV install path>/`.

**Note:** If this is your first time running `init_env.sh`, you must restart and rerun the script for permanent permissions and system parameter settings to take effect.

**Note:** Each time you restart the host or start a new shell, rerun the `init_env.sh` script. Most settings are temporary.

The script completes the following tasks:
• Exports the following environment variables:
  — OPAE_PLATFORM_ROOT: Points to the extracted Intel Acceleration Stack release
  — AOCL_BOARD_PACKAGE_ROOT: Points to the unpacked OpenCL BSP
  — INTELFPGAOCLSDKROOT: The Intel FPGA SDK for OpenCL installation directory
• Runs the OpenCL initialization script to enable the runtime environment or the development environment (if installed) by running init_opencl.sh
• Sets various permissions and system parameters by running setup_permissions.sh
• Adds the Intel SDK for OpenCL (aocl) utility located at $INTELFPGAOCLSDKROOT/bin to your PATH
4. Running Diagnostics

Before running diagnostics, load an OpenCL kernel to the board. The following instructions use the hello_world kernel or you may also use your own.

1. Load hello_world OpenCL kernel:

   ```
   $ aocl program acl0 $OPAE_PLATFORM_ROOT/opencl/hello_world.aocx
   ```

   Sample program output:

   ```
aocl program: Running program from $OPAE_PLATFORM_ROOT/opencl/opencl_bsp /linux64/libexec
Program succeed.
```  

2. Run the simple diagnostic utility:

   ```
   $ aocl diagnose
   ```

   Sample diagnostic output:

   ```
   ------------------------------------------------
   Device Name: acl0
   Package Pat: $OPAE_PLATFORM_ROOT/opencl/opencl_bsp
   Vendor: Intel Corp
   Phys Dev Name  Status   Information
   pac_a10_f200000  Passed   PAC Arria 10 Platform (pac_a10_f200000)
   PCIe 04:00.0
   FPGA temperature = 46 degrees C.
   DIAGNOSTIC_PASSED
   ------------------------------------------------
   ```

3. Run the advanced diagnostic:

   ```
   $ aocl diagnose acl0
   ```

   Sample advanced diagnostic output:

   ```
aocl diagnose: Running diagnose from $OPAE_PLATFORM_ROOT/opencl /opencl_bsp/linux64/libexec
Using platform: Intel(R) FPGA SDK for OpenCL(TM)
Using Device with name: pac_a10 : PAC Arria 10 Platform (pac_a10_f200000)
Using Device from vendor: Intel Corp clGetDeviceInfo
CL_DEVICE_GLOBAL_MEM_SIZE = 8589934592
clGetDeviceInfo CL_DEVICE_MAX_MEM_ALLOC_SIZE = 8588886016
Memory consumed for internal use = 1048576
Actual maximum buffer size 8588886016 bytes
Writing 8191 MB to global memory...
Allocated 1073741824 Bytes host buffer for large transfers
Write speed: 5447.76 MB/s [5100.38 -> 5710.86]
Reading and verifying 8191 MB from global memory ...
```
Read speed: 6319.11 MB/s [5829.62 -> 6815.82]
Successfully wrote and readback 8191 MB buffer

Transferring 262144 KBs in 512 512 KB blocks ... 3295.09 MB/s
Transferring 262144 KBs in 256 1024 KB blocks ... 3465.62 MB/s
Transferring 262144 KBs in 128 2048 KB blocks ... 4173.86 MB/s
Transferring 262144 KBs in 64 4096 KB blocks ... 5069.94 MB/s
Transferring 262144 KBs in 32 8192 KB blocks ... 5084.80 MB/s
Transferring 262144 KBs in 16 16384 KB blocks ... 5538.76 MB/s
Transferring 262144 KBs in 8 32768 KB blocks ... 6165.23 MB/s
Transferring 262144 KBs in 4 65536 KB blocks ... 6536.86 MB/s
Transferring 262144 KBs in 2 131072 KB blocks ... 6320.60 MB/s
Transferring 262144 KBs in 1 262144 KB blocks ... 6619.78 MB/s

As a reference:
PCIe Gen1 peak speed: 250MB/s/lane
PCIe Gen2 peak speed: 500MB/s/lane
PCIe Gen3 peak speed: 985MB/s/lane

Writing 262144 KBs with block size (in bytes) below:

<table>
<thead>
<tr>
<th>Block_Size</th>
<th>Avg</th>
<th>Max</th>
<th>Min</th>
<th>End-End (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>524288</td>
<td>2509.11</td>
<td>3295.09</td>
<td>1693.93</td>
<td>2018.67</td>
</tr>
<tr>
<td>1048576</td>
<td>2543.70</td>
<td>3087.25</td>
<td>1656.82</td>
<td>2279.26</td>
</tr>
<tr>
<td>2097152</td>
<td>3634.87</td>
<td>4173.86</td>
<td>2265.05</td>
<td>3410.79</td>
</tr>
<tr>
<td>4194304</td>
<td>4548.67</td>
<td>5069.94</td>
<td>3939.32</td>
<td>4362.32</td>
</tr>
<tr>
<td>8388608</td>
<td>4813.88</td>
<td>5266.92</td>
<td>3946.97</td>
<td>4821.61</td>
</tr>
<tr>
<td>16777216</td>
<td>5266.92</td>
<td>5770.48</td>
<td>4266.47</td>
<td>5206.11</td>
</tr>
<tr>
<td>33554432</td>
<td>5226.23</td>
<td>6665.99</td>
<td>4792.34</td>
<td>5219.55</td>
</tr>
<tr>
<td>67108864</td>
<td>4818.27</td>
<td>5226.23</td>
<td>3681.99</td>
<td>4952.34</td>
</tr>
<tr>
<td>134217728</td>
<td>4367.72</td>
<td>4546.45</td>
<td>4124.93</td>
<td>4366.66</td>
</tr>
<tr>
<td>268435456</td>
<td>4546.45</td>
<td>4546.45</td>
<td>4546.45</td>
<td>4546.45</td>
</tr>
</tbody>
</table>

Reading 262144 KBs with block size (in bytes) below:

<table>
<thead>
<tr>
<th>Block_Size</th>
<th>Avg</th>
<th>Max</th>
<th>Min</th>
<th>End-End (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>524288</td>
<td>2487.06</td>
<td>3038.19</td>
<td>1757.40</td>
<td>2015.28</td>
</tr>
<tr>
<td>1048576</td>
<td>2534.70</td>
<td>3087.25</td>
<td>1656.82</td>
<td>2279.26</td>
</tr>
<tr>
<td>2097152</td>
<td>3634.87</td>
<td>4173.86</td>
<td>2265.05</td>
<td>3410.79</td>
</tr>
<tr>
<td>4194304</td>
<td>4548.67</td>
<td>5069.94</td>
<td>3939.32</td>
<td>4362.32</td>
</tr>
<tr>
<td>8388608</td>
<td>4813.88</td>
<td>5266.92</td>
<td>3946.97</td>
<td>4821.61</td>
</tr>
<tr>
<td>16777216</td>
<td>5266.92</td>
<td>5770.48</td>
<td>4266.47</td>
<td>5206.11</td>
</tr>
<tr>
<td>33554432</td>
<td>5226.23</td>
<td>6665.99</td>
<td>4792.34</td>
<td>5219.55</td>
</tr>
<tr>
<td>67108864</td>
<td>4818.27</td>
<td>5226.23</td>
<td>3681.99</td>
<td>4952.34</td>
</tr>
<tr>
<td>134217728</td>
<td>4367.72</td>
<td>4546.45</td>
<td>4124.93</td>
<td>4366.66</td>
</tr>
<tr>
<td>268435456</td>
<td>4546.45</td>
<td>4546.45</td>
<td>4546.45</td>
<td>4546.45</td>
</tr>
</tbody>
</table>

Write top speed = 5662.74 MB/s
Read top speed = 6691.78 MB/s
Throughput = 6177.26 MB/s

DIAGNOSTIC_PASSED
5. OpenCL Support for Multi-Card Systems

Before running an OpenCL application, program the PAC with an Accelerator Function (AF) that includes the BSP logic. Use the `aocl program` command to load an `aocx` file to the PAC. It is only necessary to program the AF one time per PAC. After the initial programming, you can use the OpenCL API to load different applications to the PAC using the `aocx` program command.

Note: For a system with one PAC, Intel recommends that you allocate the number of hugepages to 20. If your system has multiple PACs, you must allocate 20 hugepages per card. For example, a system with four PAC requires a total of 80 hugepages.

To set the hugepages to 80, enter the following command:
```
$ sudo sh -c "echo 80 > /sys/kernel/mm/hugepages/hugepages-2048kB /nr_hugepages"
```

Run the `aocl diagnose` command to determine how many FPGAs the system includes. For example, running the `aocl diagnose` command on a system with two PAC might show output similar to the following:

1. $ aocl diagnose

```
------------------------------------------------
Device Name: acl0
Package Pat: $OPAE_PLATFORM_ROOT/opencl/opencl_bsp
Vendor: Intel Corp
Phys Dev Name    Status           Information
pac_a10_f100001  Uninitialized    OpenCL BSP not loaded. Must load BSP using command:
                                           'aocl program <device_name> <aocx_file>'
                                           before running OpenCL programs using this device
DIAGNOSTIC_PASSED
------------------------------------------------

Device Name: acl1
Package Pat: $OPAE_PLATFORM_ROOT/opencl/opencl_bsp
Vendor: Intel Corp
Phys Dev Name    Status           Information
pac_a10_f100000  Uninitialized    OpenCL BSP not loaded. Must load BSP using command:
                                           'aocl program <device_name> <aocx_file>'
                                           before running OpenCL programs using
```

Intel Corporation. All rights reserved. Agilex, Altera, Arria, Cyclone, Empirion, Intel, the Intel logo, 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.
2. The following command programs the first card listed in Step 1:
   $ aocl program acl0 $OPAE_PLATFORM_ROOT/opencl/hello_world.aocx
   
aocl program: Running program from $OPAE_PLATFORM_ROOT/opencl \opencl_bsp
   Program succeed.

3. The following command programs the second card listed in Step 1:
   $ aocl program acl1 $OPAE_PLATFORM_ROOT/opencl/hello_world.aocx
   
aocl program: Running program from $OPAE_PLATFORM_ROOT/opencl \opencl_bsp
   Program succeed.

4. After programming the FPGAs, the `aocl diagnose` command provides information about them:
   $ aocl diagnose

   ------------------------------------------------
   Device Name: acl0
   Package Pat: $OPAE_PLATFORM_ROOT/opencl/opencl_bsp
   Vendor: Intel Corp
   Phys Dev Name   Status   Information
   pac_a10_f100001 Passed   PAC Arria 10 Platform (pac_a10_f100001)
                             PCIe 04:00.0
                             FPGA temperature = 56 degrees C.
   DIAGNOSTIC_PASSED
   ------------------------------------------------

   Device Name: acl1
   Package Pat: $OPAE_PLATFORM_ROOT/opencl/opencl_bsp
   Vendor: Intel Corp
   Phys Dev Name   Status   Information
   pac_a10_f100000 Passed   PAC Arria 10 Platform (pac_a10_f100000)
                             PCIe 04:00.0
                             FPGA temperature = 48 degrees C.
   DIAGNOSTIC_PASSED
   ------------------------------------------------

   **Note:** You can run the advanced diagnostic on any specific device in your multi-card system using the following command:

   $ aocl diagnose <device name>
6. Running Samples

This section describes how to compile and run the host code for the provided samples using the precompiled OpenCL kernels.

6.1. Running Hello World

1. Extract `hello_world` example:
   
   ```
   cd $OPAE_PLATFORM_ROOT/opencl
   mkdir exm_opencl_hello_world_x64_linux
   cd exm_opencl_hello_world_x64_linux
   tar xf ../exm_opencl_hello_world_x64_linux.tgz
   ```

2. Build example:

   ```
   cd hello_world
   make
   ```

3. Copy `aocx` to example bin folder:

   ```
   cp $OPAE_PLATFORM_ROOT/opencl/hello_world.aocx ./bin/
   ```

4. Run example:

   ```
   ./bin/host
   ```

Example sample output:

```
Querying platform for info:

--------------------------
CL_PLATFORM_NAME               = Intel(R) FPGA SDK for OpenCL(TM)
CL_PLATFORM_VENDOR             = Intel(R) Corporation
CL_PLATFORM_VERSION            = OpenCL 1.0 Intel(R) FPGA SDK for
                                 OpenCL(TM), Version 17.1.1

Querying device for info:

--------------------------
CL_DEVICE_NAME                  = pac_a10 : PAC Arria 10 Platform
(pac_a10_f2000000)
CL_DEVICE_VENDOR                = Intel Corp
CL_DEVICE_VENDOR_ID             = 4466
CL_DEVICE_VERSION               = OpenCL 1.0 Intel(R) FPGA SDK for
                                 OpenCL(TM), Version 17.1.1
CL_DRIVER_VERSION               = 17.1.1
CL_DEVICE_ADDRESS_BITS          = 64
CL_DEVICE_AVAILABLE             = true
CL_DEVICE_ENDIAN_LITTLE         = true
CL_DEVICE_GLOBAL_MEM_CACHE_SIZE = 32768
CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE = 0
```
6.2. Running Vector Add

1. Extract example:
   ```
   cd $OPAE_PLATFORM_ROOT/opencl
   mkdir exm_opencl_vector_add_x64_linux
   cd exm_opencl_vector_add_x64_linux
   tar xzvf ../exm_opencl_vector_add_x64_linux.tgz
   ```

2. Build example:
   ```
   cd vector_add
   make
   ```

3. Copy precompiled OpenCL kernel to bin folder:
   ```
   cp $OPAE_PLATFORM_ROOT/opencl/vector_add.aocx ./bin
   ```

4. Program the aocx file:
   ```
   aocl program acl0 ./bin/vector_add.aocx
   ```

5. Run example:
   ```
   ./bin/host
   ```

Example sample output:

```
Initializing OpenCL
Platform: Intel(R) FPGA SDK for OpenCL(TM)
Using 1 device(s)
pac_a10 : PAC Arria 10 Platform (pac_a10_f200000)
Using AOCX: vector_add.aocx
Reprogramming device [0] with handle 1
Launch for device 0 (1000000 elements)
Thread #2: Hello from Altera's OpenCL Compiler!
Kernel execution is complete.
```
Time: 8.046 ms
Kernel time (device 0): 3.711 ms
Verification: PASS
7. Compiling OpenCL Kernels

1. Set the user environment variable using one of the following commands:

```bash
source <DEV Install Path>/init_env.sh
source <DEV Install Path>/intelFPGA_pro/hld/init_opencl.sh
export ALTERAOCLSDKROOT=$INTELFPGAOCLSDKROOT
```

2. Ensure that the environment is setup with correct BSP using the following command:

```bash
aoc --list-boards
```

**Output**
```
Board list:
pac_a10
Board Package: /home/username/inteldevstack/opencl_bsp
```

3. Compile an OpenCL Kernel to an aocx using commands similar to the following:

```bash
cd $OPAE_PLATFORM_ROOT/opencl/exm_opencl_vector_add_x64_linux/vector_add
daoc device/vector_add.cl -o bin/vector_add.aocx -board pac_a10
```

**Related Information**

Setting the Intel FPGA SDK for OpenCL User Environment Variables

7.1. Checking Timing Results

Intel recommends that you check for timing failures after compilation of the aocx file.

Check the compilation directory for the presence of the following report files:

```bash
afu_fit.failing_clocks.rpt
afu_fit.failing_paths.rpt
```

For example, after compiling vector_add.cl, locate the $OPAE_PLATFORM_ROOT/opencl/exm_opencl_vector_add_x64_linux/vector_add/device/vector_add directory. If there is a timing violation, this directory contains the failing report files. The failing report files indicate that the timing is not clean and the functional correctness cannot be guaranteed.

If OpenCL kernel compilation results in timing violations, Intel recommends to retry compilation with a different seed (aoc <kernel.cl>--seed <integer>).
For example,

```
  aoc vector_add.cl --seed 2
  aoc vector_add.cl --seed 3
  aoc vector_add.cl --seed 63
```
8. Running an OpenCL Design Example

This section describes how to run an OpenCL design example on Intel PAC with Intel Arria 10 GX FPGA. The design example demonstrates the FFT (2D) design.

**Note:** You must set up the host machine using the instructions in the Setting up the Host Machine section.

**Related Information**
Setting Up the Host Machine on page 6

8.1. Downloading an OpenCL Design Example

The following instructions are for downloading the FFT (2D) design example.

1. Navigate to the Intel FPGA SDK for OpenCL page.
2. Click the **Design Examples** tab, and under High-Performance Computing Platform Examples, click the **FFT (2D)**. This navigates to OpenCL 2D Fast Fourier Transform Design Example page.
3. Under the Downloads, click the `<version> x64 Linux package (.tar.gz)` and download the tar file `exm_opencl_fft2d_x64_linux.tar.gz` to your chosen directory.
4. Uncompress the `.tar.gz` file by typing the following command:

   ```
   $ tar zxf exm_opencl_fft2d_x64_linux.tar.gz
   ```

   This command creates a directory named `exm_opencl_fft2d_x64_linux` which contains the following files:
This OpenCL design example has two parts:

- Host code which is executed on the host machine
  \((\text{installation
directory}/fft2d/host)\)

- Kernel code which is executed on an FPGA
  \((\text{installation
directory}/fft2d/device)\)
Compile these two portions separately. Compile the host code using any C/C++ compiler and the kernel code using the Intel OpenCL compiler.

### 8.2. Compiling the Kernel

Follow these instructions to compile the kernel.

1. Set the environment variables as by using one of the following commands:

   ```bash
   $ source <DEV Install Path>/init_env.sh
   $ source <RTE Install Path>/init_env.sh
   ```

2. Type the following command to check the availability of the OpenCL compiler in the path.

   ```bash
   $ which aoc
   ```

3. Compile the kernel by typing the following command. This may take 4 to 7 hours to complete the compilation.

   ```bash
   $ cd ff2d
   $ aoc -v -board pac_a10 device/ff2d.cl -o bin/ff2d.aocx
   ```

**Related Information**

- [Compiling OpenCL Kernels](#) on page 16
  For more information about how to install the Intel FPGA SDK for 17.1.1 for Linux.

### 8.3. Compiling the Host

1. To compile the host, make sure that you have installed GCC (C/C++) compiler in the path by typing the following command:

   ```bash
   $ which gcc
   
   **Note:** You must install GCC compiler version 4.8.5 or above.

2. If GCC is available in the path, type the following command to compile the host code. This command creates the executable named host under the ./bin directory.

   ```bash
   $ make
   ```

This creates the executable named `host` under the ./bin directory.
8.4. Running the Executable

Before you run the host, you must need compiled OpenCL kernel and host.

1. To run the host program on the hardware, run the executable named `host` under `bin` directory by typing the following command:

   ```
   $ ./bin/host
   ```

   The output displays similar to below:

   ```
   Using AOCX:fft2d.aocx
   Reprogramming device [0] with handle 1
   Launching FFT transform (ordered data layout)
   Kernel initialization is complete.
   Processing time= 3.0598ms
   Throughput= 0.3427 Gpoints/sec (34.2690 Gflops)
   Signal to noise ratio on output sample: 137.231003 → PASSED
   
   Launching inverse FFT transform (ordered data layout)
   Kernel initialization is complete.
   Processing time= 3.0628ms
   Throughput= 0.3424 Gpoints/sec (34.2364 Gflops)
   Signal to noise ratio on output sample: 136.860797 → PASSED
   
   Launching FFT transform (alternative data layout)
   Kernel initialization is complete.
   Processing time= 2.2904ms
   Throughput= 0.4578 Gpoints/sec (45.7821 Gflops)
   Signal to noise ratio on output sample: 137.435876 → PASSED
   
   Launching inverse FFT transform (alternative data layout)
   Kernel initialization is complete.
   Processing time= 2.3253ms
   Throughput= 0.4509 Gpoints/sec (45.0934 Gflops)
   Signal to noise ratio on output sample: 136.689050 → PASSED
   ```

<table>
<thead>
<tr>
<th>Intel Acceleration Stack Version</th>
<th>User Guide (PDF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>OpenCL on the Intel PAC with Intel Arria 10 GX FPGA Quick Start User Guide</td>
</tr>
<tr>
<td>1.0</td>
<td>OpenCL on the Intel PAC with Intel Arria 10 GX FPGA Quick Start User Guide</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Intel Acceleration Stack Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 2019.06.28       | 1.2 (supported with Intel Quartus® Prime Pro Edition 17.1.1) | • Added a step to program the aocx file in section *Running Vector Add*.  
• Modified command to set the user environment variable in section *Compiling OpenCL Kernels*. |
| 2018.12.04       | 1.2 (supported with Intel Quartus Prime Pro Edition 17.1.1) | Simplified the installation process by including more commands in the `init_env.sh` script and using `<RTE Install Path>` and `<DEV Install Path>` for the installation paths when appropriate. |
| 2018.11.02       | 1.1 (supported with Intel Quartus Prime Pro Edition 17.1.1) | Added *Configuring the OpenCL Driver* topic to the *Setting up the Host Machine* chapter. |
| 2018.08.06       | 1.1 (supported with Intel Quartus Prime Pro Edition 17.1.1) | Initial release. |
A. Disabling Non-Uniform Memory Access (NUMA) and DMA Worker Threads to Optimize PCIe Bandwidth

OpenCL transfers data from the host to the FPGA device using the DMA. By default, there is a DMA worker thread that performs the transaction and triggers a callback function when the transaction completes. The DMA worker thread tends to improve the overall performance by allowing the host program to continue working while the DMA transfer is in progress.

By default, the runtime uses `numactl` to keep the DMA worker thread on the same NUMA node as the main OpenCL thread. This reduces performance overhead associated with transferring data to a worker thread on a different node.

Although the defaults are intended to provide best performance, on some systems, users may want to disable the worker thread to improve PCIe bandwidth, or the NUMA affinity to allow the OS more freedom in scheduling threads.

To experiment with tuning the memory transfer performance, the OpenCL MMD provides two environment variables:

- **DISABLE_NUMA_AFFINITY_ENV**: Disables the settings of the CPU affinity. This allows the Operating System (OS) to schedule the DMA thread on any core. You can enable this environment variable by typing the following command:
  
  ```
  export DISABLE_NUMA_AFFINITY_ENV=yes
  ```

- **DISABLE_DMA_WORK_THREAD_ENV**: Disables the DMA worker thread entirely. This converts large data transfers into a blocking operation in the host code. You can enable this environment variable by typing the following command:
  
  ```
  export DISABLE_DMA_WORK_THREAD_ENV=yes
  ```