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**

Subscribe

Send Feedback

UG-20166 | 2018.12.04

Latest document on the web: PDF | HTML

Contents

1. About this Document

1.1. Conventions

Table 1. Document Conventions

Convention	Description
#	Precedes a command that indicates the command is to be entered as root.
$	Indicates a command is to be entered as a user.
`This font`	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.
<variable_name>	Indicates the placeholder text that appears between the angle brackets must be replaced with an appropriate value. Do not enter the angle brackets.

1.2. Acceleration Glossary

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

Term	Abbreviation	Description
Intel® Acceleration Stack for Intel Xeon® CPU with FPGAs	Acceleration Stack	A collection of software, firmware, and tools that provides performance-optimized connectivity between an Intel FPGA and an Intel Xeon processor.
Intel Programmable Acceleration Card with Intel Arria® 10 GX FPGA	Intel PAC with Intel Arria 10 GX FPGA	PCIe* accelerator card with an Intel Arria 10 FPGA. Programmable Acceleration Card is abbreviated PAC. Contains a FPGA Interface Manager (FIM) that connects to an Intel Xeon processor over PCIe bus.
Intel Xeon Scalable Platform with Integrated FPGA	Integrated FPGA Platform	A platform with the Intel Xeon and FPGA in a single package and sharing a coherent view of memory using the Intel Ultra Path Interconnect (UPI).

1.3. Acronyms

Table 3. Acronyms

Acronyms	Expansion	Description
AFU	Accelerator Functional Unit	Hardware Accelerator implemented in FPGA logic which offloads a computational operation for an application from the CPU to improve performance.
AF	Accelerator Function	Compiled Hardware Accelerator image implemented in FPGA logic that accelerates an application.
API	Application Programming Interface	A set of subroutine definitions, protocols, and tools for building software applications.
FIM	FPGA Interface Manager	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.
OPAE	Open Programmable Acceleration Engine	The OPAE is a software framework for managing and accessing AFs.

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 Archives on page 22

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: 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*, before running OpenCL.

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

3.1. Installing the Release

Follow the instructions from Chapter 2 to Chapter 6 of 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 or in a virtualized environment with Single Root I/O Virtualization (SR-IOV) disabled.

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
  - `INTELFPGAOCLSKDROOT`: 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

3.3. Configuring the OpenCL Driver

1. You must configure the the Acceleration Stack OpenCL driver by running on of the following command:

```
sudo -E aocl install
```

2. When you are asked to install the BSP, answer Y for yes.

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:

Block_Size Avg    Max    Min    End-End (MB/s)
  524288 2509.11 3295.09 1693.93 2018.67
 1048576 2543.70 3087.25 1656.82 2279.26
 2097152 3634.87 4173.86 2265.05 3410.79
 4194304 4548.67 5069.94 3939.32 4362.32
 8388608 4813.88 5084.80 4089.09 4722.04
16777216 5266.92 5446.97 4821.61 5206.11
33554432 4818.27 5226.23 3681.99 4792.34
67108864 4964.35 5662.74 4123.11 4952.34
134217728 4367.72 4640.88 4124.93 4366.66
268435456 4546.45 4546.45 4546.45 4546.45

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

Block_Size Avg    Max    Min    End-End (MB/s)
  524288 2487.06 3038.19 1757.40 2015.28
 1048576 2934.13 3465.62 2241.64 2613.45
 2097152 3485.74 3673.13 2820.99 3296.42
 4194304 3406.50 3629.74 3040.80 3300.23
 8388608 4474.60 4589.06 4241.70 4378.74
16777216 5289.71 5538.76 5081.67 5219.55
33554432 6014.68 6165.23 5686.37 5976.21
67108864 6440.31 6536.86 6365.68 6421.60
134217728 6106.75 6320.60 5906.89 6098.65
268435456 6691.78 6691.78 6691.78 6691.78

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

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

DIAGNOSTIC_PASSED
---------------------------------------------------------
```

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:

```
$ export ALTERAOCLSDKROOT=$INTELFPGAOCLSDKROOT

$ 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_f200000)
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
CL_DEVICE_GLOBAL_MEM_SIZE                = 8589934592
CL_DEVICE_IMAGE_SUPPORT                  = true
CL_DEVICE_LOCAL_MEM_SIZE                 = 16384
CL_DEVICE_MAX_CLOCK_FREQUENCY            = 1000
CL_DEVICE_MAX_COMPUTE_UNITS              = 1
CL_DEVICE_MAX_CONSTANT_ARGS              = 8
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE       = 2147483648
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS       = 3
CL_DEVICE_MEM_BASE_ADDR_ALIGN            = 8192
CL_DEVICE_MIN_DATA_TYPE_ALIGN_SIZE       = 1024
CL_DEVICE_PREFERRED_VECTOR_WIDTH_CHAR    = 4
CL_DEVICE_PREFERRED_VECTOR_WIDTH_SHORT   = 2
CL_DEVICE_PREFERRED_VECTOR_WIDTH_INT     = 1
CL_DEVICE_PREFERRED_VECTOR_WIDTH_LONG    = 1
CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT   = 1
CL_DEVICE_PREFERRED_VECTOR_WIDTH_DOUBLE  = 0
Command queue out of order?              = false
Command queue profiling enabled?         = true
Using AOCX: hello_world.aocx
Reprogramming device [0] with handle 1

Kernel initialization is complete.
Launching the kernel...

Thread #2: Hello from Altera's OpenCL Compiler!

Kernel execution is complete.
```

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:

```
$ export ALTERAOCLSDKROOT=$INTELFPGAOCLSDKROOT

$ cd vector_add

$ make
```

3. Copy precompiled OpenCL kernel to bin folder:

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

4. 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
Launching for device 0 (1000000 elements)
```

```
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:

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

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

```
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:

```
$ cd $OPAE_PLATFROM_ROOT/opencl/exm_opencl_vector_add_x64_linux/vector_add

$ aoc 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:

```
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

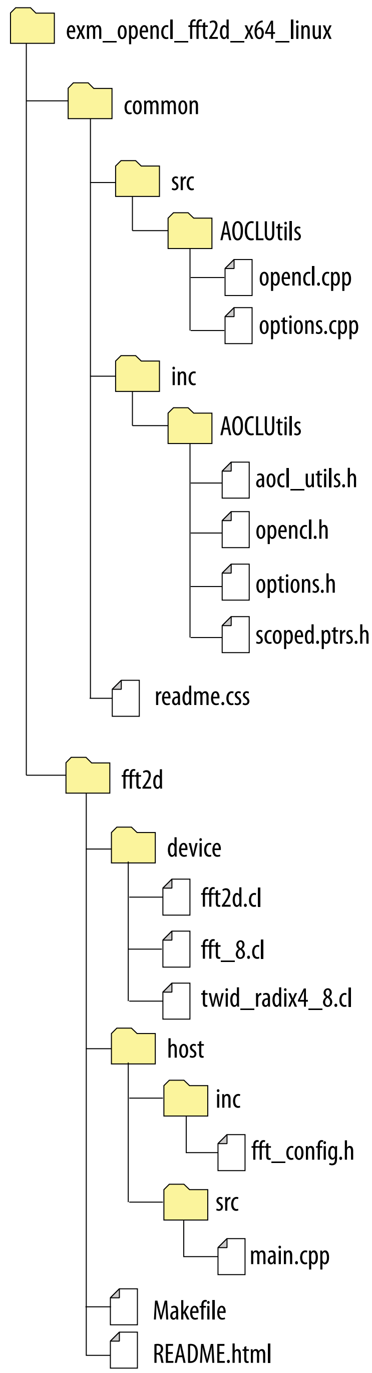
8.1. Downloading an OpenCL Design Example

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 zxvf 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 (`installation_directory/fft2d/host`)
- Kernel code which is executed on an FPGA (`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:

```
$ 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.

```
$ which aoc
```

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

```
$ cd ff2d

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

Related Information

Compiling OpenCL Kernels on page 17
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:

   Note: You must install GCC compiler version 4.8.5 or above.

```
$ which gcc
```

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.

```
$ 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 Gflpos)
    Signal to noise ration 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 ration 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 ration on output sample: 136.689050 PASSED
```

9. OpenCL on the Intel PAC with Intel Arria 10 GX FPGA Quick Start User Guide Archives

Intel Acceleration Stack Version	User Guide (PDF)
1.1	OpenCL on the Intel PAC with Intel Arria 10 GX FPGA Quick Start User Guide
1.0	OpenCL on the Intel PAC with Intel Arria 10 GX FPGA Quick Start User Guide

10. Document Revision History for OpenCL on Intel Programmable Acceleration Card with Intel Arria 10 GX FPGA Quick Start User Guide

Document Version	Intel Acceleration Stack Version	Changes
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
  ```