OpenCL* on Intel® Programmable Acceleration Card with Intel® Arria® 10 GX FPGA 快速入门用户指南

针对面向包含 FPGA 的 Intel® Xeon® CPU 的 Intel®加速堆栈进行的更新：**1.2**

本翻译版本仅供参考，如果本翻译版本与其英文版本存在差异，则以英文版本为准。某些翻译版本尚未更新对应到最新的英文版本，请参考英文版本以获取最新信息。

在线版本
发送反馈
UG-20166
ID: 683831
版本：2018.12.04

内容

1. 关于本文档

1.1. 约定

表 1. 文件约定

约定	描述
#	位于一个命令之前，表明作为 root 输入此命令。
$	表明作为用户(user)输入一个命令。
This font	文件名，命令和关键字都以此字体打印。长命令行以此字体打印。虽然长命令行可能会换行到下一行，但返回不是命令的一部分；不要按回车键。
<variable_name>	表示尖括号之间显示的占位符文本必须替换成相应的值。不要输入尖括号。

1.2. 加速术语表(Acceleration Glossary)

表 2. Acceleration Stack for Intel® Xeon® CPU with FPGAs 术语

术语	缩写	描述
Intel® Acceleration Stack for Intel Xeon® CPU with FPGAs	Acceleration Stack	一系列软件，固件和工具，提供 Intel FPGA 和 Intel Xeon 处理器之间性能优化的连接。
Intel Programmable Acceleration Card with Intel Arria® 10 GX FPGA	Intel PAC with Intel Arria 10 GX FPGA	具有 Intel Arria 10 FPGA 的 PCIe* 加速卡。可编程加速卡(Programmable Acceleration Card)缩写为 PAC。 包含一个 FPGA Interface Manager (FIM)，通过 PCIe 总线连接到 Intel Xeon 处理器。
Intel Xeon Scalable Platform with Integrated FPGA	Integrated FPGA Platform	Intel Xeon 和 FPGA 位于同一封装中的一种平台，使用 Intel Ultra Path Interconnect (UPI)共享一个连贯视图的存储器。

1.3. 略缩语

表 3. 略缩语

略缩语	全称	描述
AFU	Accelerator Functional Unit	在 FPGA 逻辑中实现的 Hardware Accelerator，从 CPU 卸载一个计算操作以提高性能。
AF	Accelerator Function	在 FPGA 逻辑中实现的已编译的 Hardware Accelerator 映像， 对应用程序进行加速。
API	Application Programming Interface	一组用于构建软件应用程序的子程序定义，协议和工具。
FIM	FPGA Interface Manager	FPGA 硬件，包含 FPGA Interface Unit (FIU)和用于存储器，网络等的外部接口。 Accelerator Function (AF)在运行时与 FIM 连接。
OPAE	Open Programmable Acceleration Engine	OPAE 是一个软件框架，用于管理和访问 AF。

2. 引言

本用户指南介绍了如何开始在 Intel PAC with Intel Arria 10 GX FPGA 1.2 Release 上使用 OpenCL* 。本指南使用包含在此 1.2 Release 中的预编译的 OpenCL 内核。本用户指南也包括对编译 OpenCL 内核的简介。

OpenCL 设计包含两个组件，内核(kernel)和主机(host)。kernel 包含加速器代码。host 在主机上运行。加速卡插入主机中。

注意: 在主机上必须有 root 权限才能对 OpenCL 进行设置。

相关链接

- 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 快速入门用户指南存档 (第 20 页)

2.1. 发布内容

$OPAE_PLATFORM_ROOT 下的发布包括 Intel PAC with Intel Arria 10 GX FPGA 1.2 Release 的文件。发布包括位于$OPAE_PLATFORM_ROOT/opencl 文件夹中的 OpenCL 文件：

- 1.2 OpenCL Board Support Package (BSP)：
  - `opencl_bsp`
- OpenCL 实例设计的测试文件：
  - `exm_opencl_hello_world_x64_linux.tgz`
  - `exm_opencl_vector_add_x64_linux.tgz`
- 预编译的内核(pre-compiled kernels) <aocx>：
  - `hello_world.aocx`
  - `vector_add.aocx`

相关链接

Understanding the Extracted Intel PAC with Intel Arria 10 GX FPGA Release Package

3. 设置主机(Setting Up the Host Machine)

先决条件：在运行 OpenCL 之前，必须按照 *Intel Acceleration Stack Quick Start Guide for Intel Programmable Acceleration Card with Arria 10 GX FPGA* 的 *Getting Started* 部分中的说明进行操作。

注意:

- 如果需要 OpenCL 编译器和工具来构建并运行 OpenCL AFU，那么要下载并安装 Intel Acceleration Stack for Development。安装开发软件确保了 OpenCL SDK 位于`/home/<username>/inteldevstack/`或者 Custom Directory，`/<custom Directory>`下。本用户指南将此路径指为`/<Dev Install Path>`。
- 如果只需要 Intel FPGA SDK 用于 OpenCL 部署功能，那么下载并安装 Intel Acceleration Stack for Runtime。安装运行时环境确保了 OpenCL RTE 安装在`/home/<username>/intelrtestack/`或者 Custom Directory，`/<custom Directory>`下。本用户指南将此路径指为`/<RTE Install Path>`。
- 请不要将 RTE 和 DEV 安装在同一主机系统上。DEV 已经包含了 RTE。

相关链接

- 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. 安装发行版(Installing the Release)

按照 *Quick Start Guide* 中的第 2 章到第 6 章中的指南对 Intel PAC with Intel Arria 10 GX FPGA 进行设置。

您可以在一个非虚拟化的环境中或者在一个 Single Root I/O Virtualization (SR-IOV)禁用的虚拟化环境中运行 OPAE 软件。

如果要在一个包含 SR-IOV 的虚拟化环境中运行 OpenCL 参考设计，那么需要完成以下额外的步骤:

1. 通过输入以下命令，从主机对所需的 OpenCL 配置进行编程:

```
$ aocl program <device name><filename>
```

注意: 1.2 Release 不支持虚拟化环境中的部分重配置。

2. 根据 *Quick Start Guide* 的 *Updating Settings Required for VFs* 和 *Configuring the VF Port on the Host* 部分中的说明来使能虚拟化。
3. 设置虚拟机中的 `CL_CONTEXT_COMPILER_MODE_INTELFPGA` 环境变量来禁止 OpenCL 主机运行时期间对 FPGA 配置或者重配置：

```
$ export CL_CONTEXT_COMPILER_MODE_INTELFPGA=3
```

4. 从虚拟机运行所需的应用程序。
5. 根据 *Quick Start Guide* 的 *Disconnecting the VF from the VM and Reconnecting to the PF* 部分中的说明来禁止虚拟化。

相关链接

- 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. 初始化 Intel Acceleration Stack for OpenCL

`init_env.sh` 脚本执行 Acceleration Stack for OpenCL 的所有初始化和设置。此脚本位于 `/<RTE install path>/`或者`/<DEV install path>/`中。

注意: 如果这是您第一次运行 `init_env.sh`，那么必须重新启动并重新运行此脚本，以使永久权限和系统参数生效。

注意: 每次重新启动主机或开始一个新的 shell 都要重新运行 `init_env.sh` 脚本。大多数设置是临时的。

此脚本完成以下任务：

- 导出以下环境变量：
  - `OPAE_PLATFORM_ROOT`：指向以提取的 Intel Acceleration Stack 发行版
  - `AOCL_BOARD_PACKAGE_ROOT`：指向未封装(unpacked)的 OpenCL BSP
  - `INTELFPGAOCLSKDROOT`：Intel FPGA SDK for OpenCL 安装目录
- 运行 OpenCL 初始化脚本(`init_opencl.sh`)来使能运行时环境或者开发环境(如果已安装)。
- 通过运行 `setup_permissions.sh` 来设置各种权限和系统参数
- 将位于`$INTELFPGAOCLSDKROOT/bin` 的 Intel SDK for OpenCL (`aocl`)实用程序添加到 PATH

3.3. 配置 OpenCL Driver

1. 您必须通过运行以下命令来对 Acceleration Stack OpenCL driver 进行配置：

```
sudo -E aocl install
```

2. 当被问及是否安装 BSP 时，回答 Y (yes)。

4. 运行诊断程序(Running Diagnostics)

在运行诊断程序之前，请加载一个 OpenCL 内核(kernel)到板级。以下说明使用 `hello_world` 内核(kernel)，或者您也可以使用自己的内核(kernel)。

1. 加载 `hello_world` OpenCL 内核(kernel)：

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

示例程序输出：

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

2. 运行简单诊断实用程序：

```
$ aocl diagnose
```

诊断程序输出示例：

```

-----------------------------------------------
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. 运行高级诊断程序：

```
$ aocl diagnose acl0
```

高级诊断程序输出示例：

```
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. 对多卡系统的 OpenCL 支持

在运行 OpenCL 应用程序之前，要对包含 BSP 逻辑的 Accelerator Function (AF)的 PAC 卡进行编程。使用 `aocl` 程序命令将一个 `aocx` 文件加载到 PAC 卡。只需要对每个 PAC 卡的 AF 编程一次。初始编程后，可以使用 OpenCL API (使用 `aocx` 程序命令)将不同的应用程序加载到 PAC 卡中。

注意: 对于使用一个 PAC 卡的系统， Intel 建议将 hugepage 的数量分配成 20。如果系统中有多个 PAC 卡，那么每个卡必须分配 20 个 hugepage。例如，一个使用四个 PAC 卡的系统需要总共 80 个 hugepage。

输入以下命令将 hugepag 的数量设为 80：

```
$ sudo sh -c "echo 80 > /sys/kernel/mm/hugepages/hugepages-2048kB \
/nr_hugepages"
```

运行 `aocl diagnose` 命令以确定系统包含的 FPGA 数量。例如，在使用两个 PAC 卡的系统上运行 `aocl diagnose` 命令可能会显示类似于以下内容的输出：

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. 以下命令对 Step 1 中列出的第一个卡进行编程：

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

```
aocl program: Running program from $OPAE_PLATFORM_ROOT/opencl \
/opencl_bsp

Program succeed.
```

3. 以下命令对 Step 1 中列出的第二个卡进行编程：

   ```
   $ aocl program acl1 $OPAE_PLATFORM_ROOT/opencl/
   hello_world.aocx
   ```

```
aocl program: Running program from $OPAE_PLATFORM_ROOT/opencl \
/opencl_bsp

Program succeed.
```

4. 对 FPGA 编程后，`aocl diagnose` 命令提供相关信息：

   ```
   $ 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
------------------------------------------------------------
```

注意: 您可以使用以下命令在多卡系统中的任何特定器件上运行高级诊断程序：

```
$ aocl diagnose <device name>
```

6. 运行示例(Running Samples)

本节介绍如何使用预编译的 OpenCL 内核(kernel)对所提供示例编译和运行主机代码(host code)。

6.1. 运行 Hello World

1. 提取(extract) `hello_world` 示例:

```
$ 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. 构建示例(build example):

```
$ export ALTERAOCLSDKROOT=$INTELFPGAOCLSDKROOT

$ cd hello_world

$ make
```

3. 将 aocx 复制到示例 bin 文件夹:

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

4. 运行示例:

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

样例输出:

```
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. 运行矢量加(Vector Add)

1. 提取示例(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. 构建示例(build example):

```
$ export ALTERAOCLSDKROOT=$INTELFPGAOCLSDKROOT

$ cd vector_add

$ make
```

3. 将预编译的 OpenCL 内核(kernel)复制到 bin 文件夹:

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

4. 运行示例:

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

样例输出:

```
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. 编译 OpenCL 内核(Compiling OpenCL Kernels)

1. 使用下面其中一个命令设置用户环境变量：

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

2. 确保使用正确的 BSP 对环境进行设置，请使用以下命令：

```
aoc --list-boards

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

3. 使用类似于以下的命令将 OpenCL Kernel 编译成 aocx：

```
$ 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
```

相关链接

Setting the Intel FPGA SDK for OpenCL User Environment Variables

7.1. 检查时序结果

Intel 建议您在编译 aocx 文件后检查时序故障(timing failure)。

检查编译目录中是否存在以下报告文件：

```
afu_fit.failing_clocks.rpt
```

```
afu_fit.failing_paths.rpt
```

例如，编译 vector_add.cl 后，找到$OPAE_PLATFORM_ROOT/opencl/exm_opencl_vector_add_x64_linux/vector_add/device/vector_add 目录。如果存在时序违规，那么此目录将包含失败报告文件。失败报告文件指示时序不正确(not clean)，无法保证功能正确性。

如果 OpenCL 内核编译导致了时序违规，那么 Intel 建议使用不同的 seed (aoc <*kernel.cl*>--seed <*integer*>)重试编译。

例如，

```
aoc vector_add.cl --seed 2
aoc vector_add.cl --seed 3
aoc vector_add.cl --seed 63
```

8. 运行一个 OpenCL 设计实例

本节介绍如何在包含 Intel Arria 10 GX FPGA 的 Intel PAC 上运行 OpenCL 设计实例。此设计实例演示了 FFT (2D)设计。

注意: 您必须按照 *Setting up the Host Machine* 部分中的说明对主机进行设置。

相关链接

设置主机(Setting Up the Host Machine) (第 6 页)

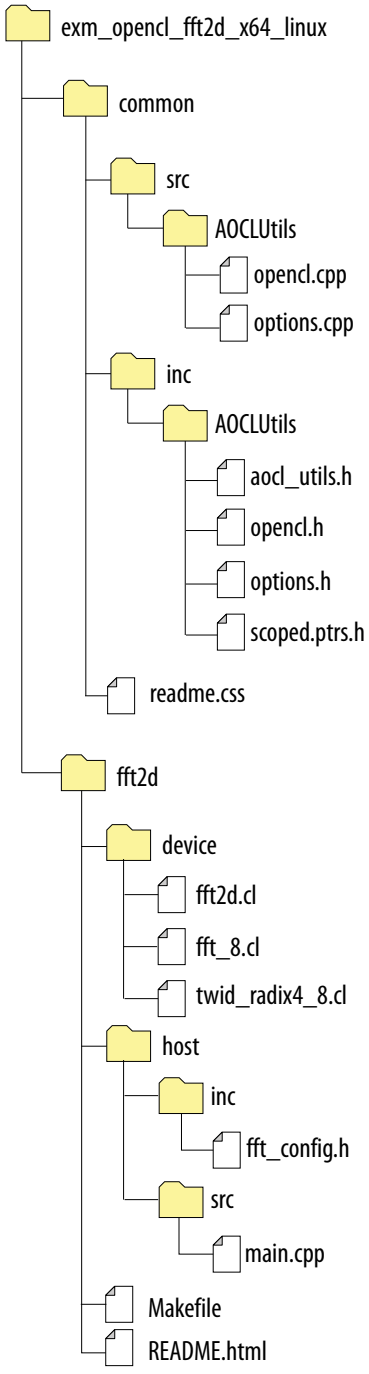
8.1. 下载 OpenCL 设计实例

按照以下步骤下载 FFT (2D)设计实例。

1. 去到 Intel FPGA SDK for OpenCL 页面。
2. 点击 **Design Examples** 选项卡，在 High-Performance Computing Platform Examples 下点击 **FFT (2D)**，这样会去到 **OpenCL 2D Fast Fourier Transform Design Example** 页面。
3. 在 Downloads 下，点击 ***<version>* x64 Linux package (.tar.gz)**，将 tar 文件 `exm_opencl_fft2d_x64_linux.tar.gz` 下载到所选的目录中。
4. 输入以下命令来解压缩 `.tar.gz` 文件：

```
$ tar zxvf exm_opencl_fft2d_x64_linux.tar.gz
```

此命令创建一个名为 `exm_opencl_fft2d_x64_linux` 的目录，其中包括以下文件：



此 OpenCL 设计实例包含两个部分：

- 在主机上执行的主机代码(host code)(*installation_directory*/fft2d/host)
- 在 FPGA 上执行的内核代码(kernel code)(*installation_directory*/fft2d/device)

分别编译这两个部分。使用 C/C++ compiler 来编译主机代码，使用 Intel OpenCL compiler 来编译内核代码。

8.2. 编译内核(Compiling the Kernel)

按照以下步骤编译内核(kernel)：

1. 使用下面其中一个命令设置环境变量：

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

2. 输入以下命令检查路径中 OpenCL compiler 的可用性。

```
$ which aoc
```

3. 输入以下命令来编译内核，这可能需要 4 到 7 个小时才能完成编译。

```
$ cd ff2d

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

相关链接

编译 OpenCL 内核(Compiling OpenCL Kernels) (第 15 页)
提供关于如何在 Linux 安装 Intel FPGA SDK for 17.1.1 的详细信息。

8.3. 编译主机(Compiling the Host)

1. 要编译主机，需确保通过输入以下命令在路径中安装了 GCC (C/C++) compiler：

   注意: 您必须安装 GCC compiler 4.8.5 或更高版本。

```
$ which gcc
```

2. 如果路径中有 GCC，那么输入以下命令来编译主机代码。此命令在 `./bin` 目录下创建名为 host 的可执行文件。

```
$ make
```

这将在 `./bin` 目录下创建名为 `host` 的可执行文件。

8.4. 运行可执行文件(Running the Executable)

在运行主机(host)前，必须要有已编译的 OpenCL kernel 和 host。

1. 要在硬件上运行主机程序，需输入以下命令来运行 bin 目录下名为 `host` 的可执行文件：

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

输出显示类似如下：

```
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 快速入门用户指南存档

Intel Acceleration Stack Version	用户指南(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. OpenCL on Intel Programmable Acceleration Card with Intel Arria 10 GX FPGA 快速入门用户指南的文档修订历史

文档版本	Intel Acceleration Stack Version	修订内容
2018.12.04	1.2 (受 Intel Quartus® Prime Pro Edition 17.1.1 支持)	通过在 `init_env.sh` 脚本中包括更多命令和使用 *`<RTE Install Path>`* 和 *`<DEV Install Path>`* 作为安装路径来简化安装过程。
2018.11.02	1.1 (受 Intel Quartus Prime Pro Edition 17.1.1 支持)	在 *Setting up the Host Machine* 章节中添加了 *Configuring the OpenCL Driver* 主题。
2018.08.06	1.1 (受 Intel Quartus Prime Pro Edition 17.1.1 支持)	首次发布。

A. 禁用非统一存储器访问(NUMA)和 DMA 工作线程以优化 PCIe 带宽

OpenCL 使用 DMA 将数据从主机传输到 FPGA 器件。默认情况下，有一个 DMA 工作线程(DMA worker thread)，此 DMA 工作线程执行传输，在传输完成时触发一个回调函数。 DMA 工作线程支持主机程序在 DMA 传输过程中继续工作，从而倾向于提高整体性能。

默认情况下，runtime 使用 `numactl` 将 DMA 工作线程保持在与主 OpenCL 线程相同的 NUMA 节点上。这降低了与传输数据到不同节点上的工作线程相关联的性能开销。

尽管默认设置旨在提供最佳性能，但在某些系统上，用户可能希望禁用工作线程来提高 PCIe 带宽或 NUMA 关联，从而实现调度线程中 OS 更多自由度。

要对调整存储器传输性能进行试验， OpenCL MMD 提供了两个环境变量：

- `DISABLE_NUMA_AFFINITY_ENV`：禁止 CPU 关联的设置。这使 Operating System (OS)能够在任何内核上调度 DMA 线程。通过输入以下命令来使能此环境变量：

  ```
  $ export DISABLE_NUMA_AFFINITY_ENV=yes
  ```

- `DISABLE_DMA_WORK_THREAD_ENV`：完全禁止 DMA 工作线程。这将大数据传输转换成主机代码中的模块化操作(blocking operation)。通过输入以下命令来使能此环境变量：

  ```
  $ export DISABLE_DMA_WORK_THREAD_ENV=yes
  ```