



APEX-CL SDK User Guide

UG-10322-00-05

Copyright

Copyright © 2014 CogniVue Corporation ("CogniVue") All rights reserved.

This document contains information which is proprietary to CogniVue and may be used for non-commercial purposes within your organization in support of CogniVue's products. No other use or transmission of all or any part of this document is permitted without written permission from CogniVue, and must include all copyright and other proprietary notices. Use or transmission of all or any part of this document in violation of any applicable Canadian or other legislation is hereby expressly prohibited.

User obtains no rights in the information or in any product, process, technology or trademark which it includes or describes, and is expressly prohibited from modifying the information or creating derivative works without the express written consent of CogniVue.

Disclaimer

CogniVue assumes no responsibility for the accuracy or completeness of the information presented which is subject to change without notice. In no event will CogniVue be liable for any direct, indirect, special, incidental or consequential damages, including lost profits, lost business or lost data, resulting from the use of or reliance upon the information, whether or not CogniVue has been advised of the possibility of such damages.

Mention of non-CogniVue products or services is for information purposes only and constitutes neither an endorsement nor a recommendation.

Revision History

Version	Details of Change	Author	Date	Status
01	Initial release	Christina Xu Lee, Ki-Ju	April 18 th , 2014	
02	Document format was changed	Stephane Francois	April 24 th , 2014	
03	1. The descriptions of applied algorithms in sample project were added 2. New and improved functions were added in Appendix A	Christina Xu Lee, Ki-Ju	Sept. 26 th , 2014	
04	Correction to Appendix A	Christina Xu	Oct. 21 st , 2014	
05	CL extension API update in Appendix A	Christina Xu	Oct. 31 st , 2014	

Table of Contents

1	Introduction	6
1.1	Acronyms	6
1.2	Scope	6
1.3	References	6
2	Using APEX OpenCL Development Environment	7
2.1	OpenCL Toolchain Overview	7
2.2	Metadata generation	9
2.2.1	Installation (MinGW and MSYS)	9
2.2.2	Makefile	9
2.2.3	<kernel_name>_md_config.cpp	10
2.3	OpenCL C kernel compilation	11
2.3.1	Project	11
2.3.2	Add a new kernel	11
2.3.2.1	APU_OCL.bcf file (linker file)	12
2.3.2.2	CL_KERNEL_DB (in apu_ocl_db.cpp)	13
2.3.3	Kernel build	14
2.4	OpenCL Host Compilation	14
3	Package Contents	17
3.1	Include	17
3.2	Libs	17
3.3	Tools	17
3.4	Example kernels and host applications	18
3.4.1	Algorithms	19
3.4.1.1	MemoryModel	19
3.4.1.2	Median Filter	20
3.4.1.3	Bitonic Sort	21
3.4.1.4	Floydwarshall	22
	Appendix A: Release Notes	23

Table of Figures

Figure 2-1: APEX-CL compilation toolchain	8
Figure 2-2: Adding new kernel directory	9
Figure 2-3 : Generating metadata for each kernel.....	10
Figure 2-4: Adding .cl kernel.....	12
Figure 2-5: Compile	14
Figure 2-6: Generate APU_OCL_LOAD.cpp	14
Figure 2-7: Open Project Manager Editor.....	15
Figure 2-8: Add your file.....	15
Figure 2-9: Newly added file	16
Figure 2-10: Host Builder	16
Figure 3-1. Median Filter.....	20
Figure 3-2. Bitonic Sort.....	21
Figure 3-3 A diagram of a graph with seven vertices and eight edges.....	22

Table of Tables

Table 1. Acronyms 6

Table 2: Adding new metadata symbols into linker file..... 13

Table 3: Kernel database structure..... 13

Table 4: CL_KRNL_DB_SIZE..... 13

Table 5: Include kernel metadata 14

1 Introduction

This document provides a basic description of the APEX OpenCL development environment and its components. It describes the basic architecture of the APEX OpenCL framework. It also provides the step-by-step guidance to programmers who want to develop OpenCL applications on APEX.

1.1 Acronyms

Terminology	Definition
APU	Array Processor Unit
CMEM	Computational Unit Memory
CU	Computational Unit
MD	Metadata
OpenCL	Open Computing Language

Table 1. Acronyms

1.2 Scope

This document is intended for OpenCL programmers. It assumes prior experience or a basic understanding of OpenCL standard, more specifically an understanding of chapters 1, 2, and 3 of [1].

1.3 References

[1] “The OpenCL Specification Version: 1.2 Document Revision: 19”, Khronos OpenCL working group.

2 Using APEX OpenCL Development Environment

Chapter 2 is organized into the following sections:

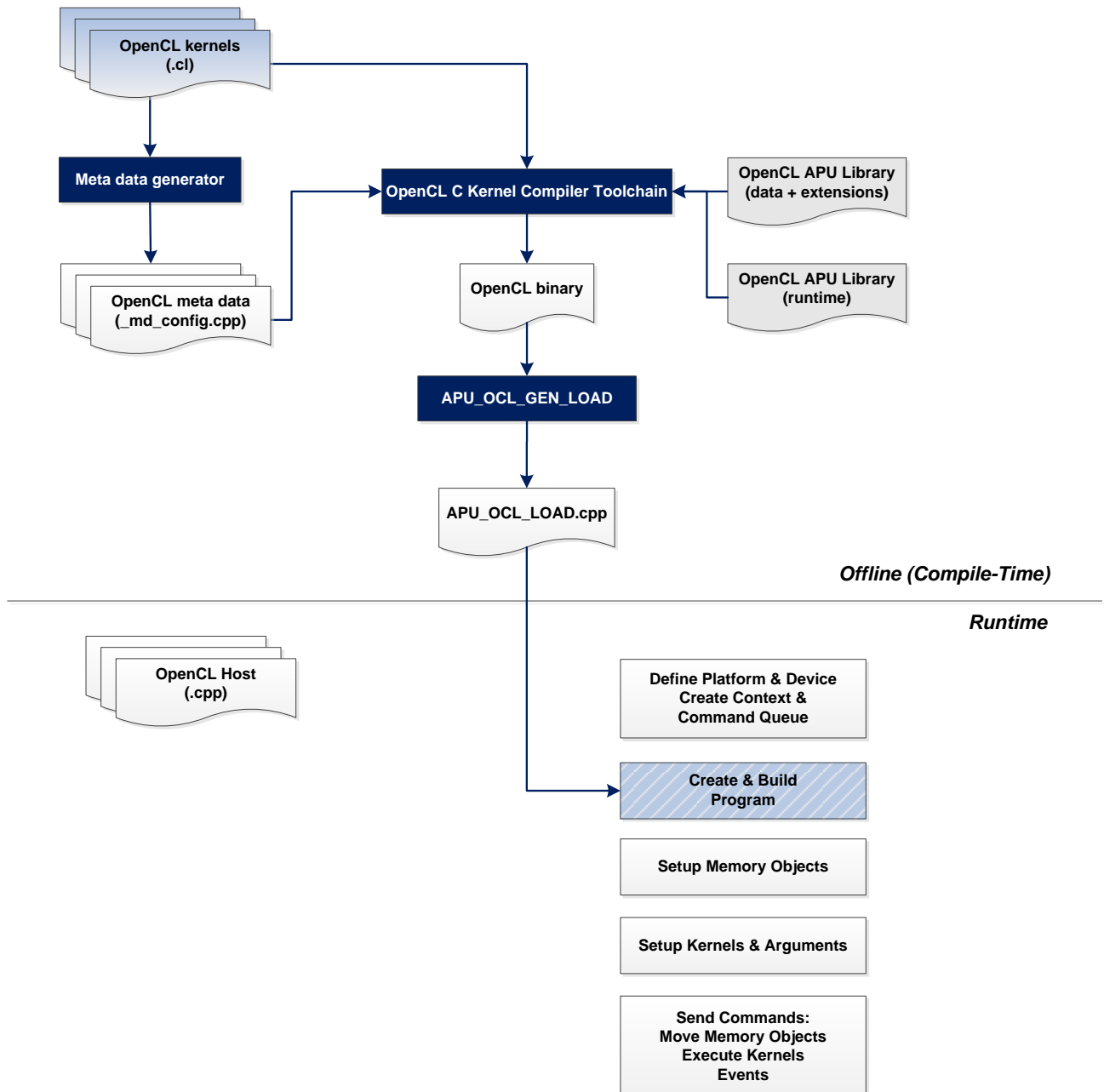
- 1) High-level overview of APEX-CL compilation and development flow (see 2.1)
- 2) Kernel metadata generation for offline compilation (see 2.2)
- 3) Kernel compilation (section 2.3)
- 4) OpenCL host program build (see 2.4)

2.1 OpenCL Toolchain Overview

Figure 2-1 (below) depicts the general compilation path of applications using OpenCL.

The OpenCL C kernel compiler is responsible for cl kernel source code parsing and compilation. It also links in the built-in OpenCL functions required by the source.

The OpenCL device runtime automatically determines the number of processing elements present in the core and distributes the OpenCL work between them.

OpenCL / APEX-CL***User OpenCL Application*****Figure 2-1: APEX-CL compilation toolchain**

2.2 Metadata generation

Kernel metadata is required for offline compilation, and a kernel metadata generator is provided as part of the package.

Required tools, their installation instructions, and the steps to add a new kernel are described below.

2.2.1 Installation (MinGW and MSYS)

1. Download: [mingw-get-setup.exe](#)
2. Install “mingw-get” by executing “mingw-get-setup.exe”
 - a. Click “Install”
 - b. Uncheck “... Also install support for the graphical user interface.”
 - c. Click “Continue”
3. Open a command prompt in the directory where “mingw-get” was installed, (e.g C:\Tools\MinGW\bin)
4. Execute these commands from the prompt:
 - a. `mingw-get install "gcc<4.8" "g++<4.8" "mingw32-make=3.82.*" msys`
 - i. (Multiple errors will appear in the output but it is safe to ignore them.)
 - b. `mingw-get install "mpc=0.8.1-1"`
5. Add the following directories to the beginning of your “[PATH](#)” environment variable:
 - a. MinGW “bin” directory, (e.g. C:\Tools\MinGW\bin).
 - b. MSYS “bin” directory, (e.g. C:\Tools\MinGW\msys\1.0\bin).

[Note] The paths for MinGW and MSYS need to be at the beginning of the “PATH” variable.

2.2.2 Makefile

- Add the new kernel into the Makefile as illustrated below:

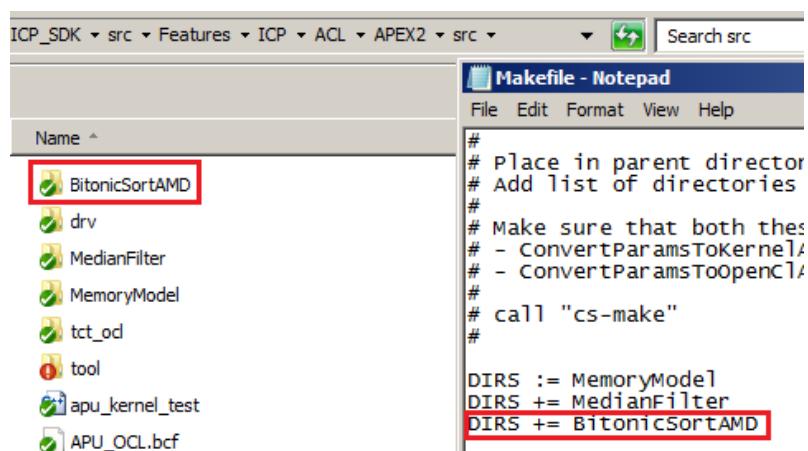
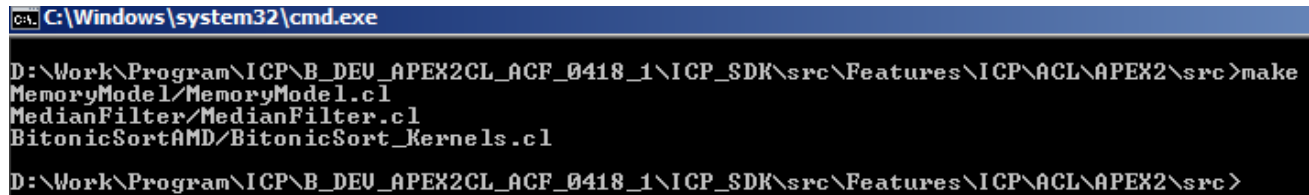


Figure 2-2: Adding new kernel directory

- Run command prompt and call 'make' at:
<ROOT> \ICP_SDK\src\Features\ICP\ACL\APEX2\src

<kernel_name>_md_config.cpp would be generated locally.



```

C:\Windows\system32\cmd.exe
D:\Work\Program\ICP\B_DEU_APEX2CL_ACF_0418_1\ICP_SDK\src\Features\ICP\ACL\APEX2\src>make
MemoryModel/MemoryModel.cl
MedianFilter/MedianFilter.cl
BitonicSortAMD/BitonicSort_Kernels.cl
D:\Work\Program\ICP\B_DEU_APEX2CL_ACF_0418_1\ICP_SDK\src\Features\ICP\ACL\APEX2\src>

```

Figure 2-3 : Generating metadata for each kernel

2.2.3 <kernel_name>_md_config.cpp

The generated file includes the following:

- Kernel wrapper function declaration:

```

int <kernel_name>__3 (
    int nxt,
    vuint _vec_wx, vuint _vec_wy, size_t wz,
    const _cl_runtime * rt,
    void chess_storage(DMb)*, void chess_storage(VMb)*);

```

- Kernel metadata struct:
CL_KRNL_MD <kernel_name>_md

This data structure includes all the basic information about an OpenCL kernel:

- Number of arguments
- Address qualifier type
- Pointer access type
- Function pointer to kernel wrapper function
- Function pointer to kernel set argument function
- void <kernel_name>__3__set(void chess_storage(DMb)* ctx, cl_argument* args)
This function is used to set up the arguments for kernel execution.

2.3 OpenCL C kernel compilation

2.3.1 Project

- Run ChessDE
- Open a project (path: <ROOT>\ICP_SDK\src\Features\ICP\ACL\APEX2\src\apu_ocl.prx)

2.3.2 Add a new kernel

(Prerequisite: kernel metadata generation, see 2.2)

- Select Project > Add source files, and choose the kernel.

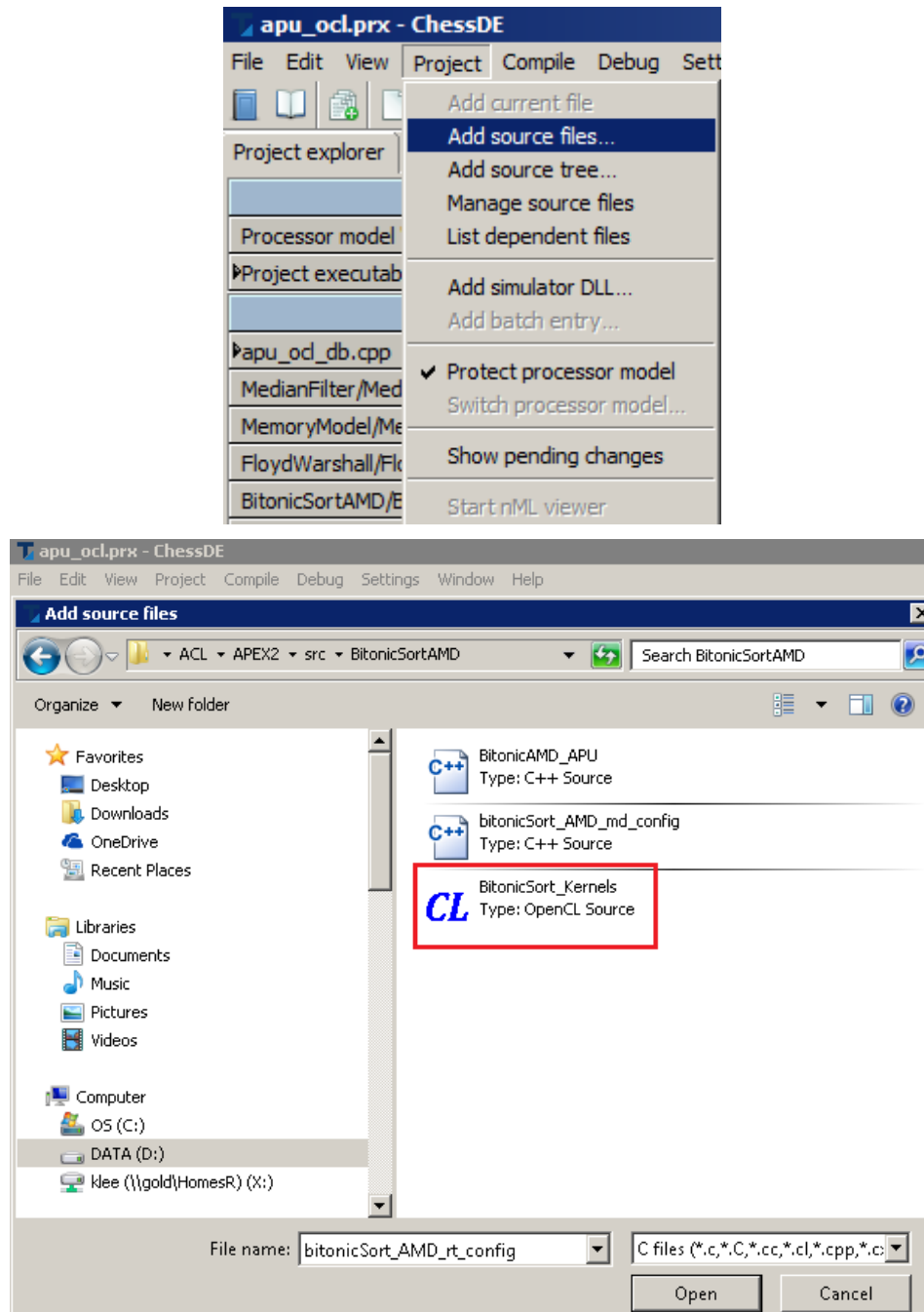


Figure 2-4: Adding .cl kernel

2.3.2.1 APU_OCL.bcf file (linker file)

Add your kernel metadata symbol into APU_OCL.bcf as follows:

Note that the kernel metadata symbols have to be in the same order as they are added into the kernel database as described in section 2.3.2.2

```

_symbol CL_KRNL_DB_SIZE      0x10D00
_symbol CL_KERNEL_DB         0x10D04
_symbol MemoryModel_md      _after CL_KERNEL_DB
_symbol MedianFilterBitonic_md _after MemoryModel_md
_symbol bitonicSort_AMD_md  _after MedianFilterBitonic_md

```

Table 2: Adding new metadata symbols into linker file**2.3.2.2 CL_KERNEL_DB (in apu_ocl_db.cpp)**

The kernel database is required for host execution to retrieve information for each kernel that is included in the load.

The kernel developer is responsible for adding the new kernel entry into the kernel database, and the order of kernels should match with the kernel order in the *.bcf file.

For each kernel, the first component is the name of kernel function. The second is the pointer to <kernel_name>_md data struct.

```

const CL_KRNL_DB_ENTRY CL_KERNEL_DB[16] =
{
    { "MemoryModel",      (int32_t)&MemoryModel_md      },
    { "MedianFilterBitonic", (int32_t)&MedianFilterBitonic_md },
    { "bitonicSort_AMD",   (int32_t)&bitonicSort_AMD_md   },
    { 0,                  0                               },
};

```

Table 3: Kernel database structure

- CL_KRNL_DB_SIZE[0] is used to indicate the number of kernels registered in the database. It is the kernel developer's responsibility to ensure the size matches with the number of registered kernels.

```

// This number must match with the number of registered kernels
const int32_t CL_KRNL_DB_SIZE[1] = {3};

```

Table 4: CL_KRNL_DB_SIZE

- Include the generated metadata file in the header file of a kernel.

```
#ifndef _BITONICSORT_AMD_H_
#define _BITONICSORT_AMD_H_

#include "bitonicSort_AMD_md_config.cpp"

#endif // _BITONICSORT_AMD_H_
```

Table 5: Include kernel metadata

2.3.3 Kernel build

- Select Compile > Make (or Rebuild) or click the Make button to build your kernel.

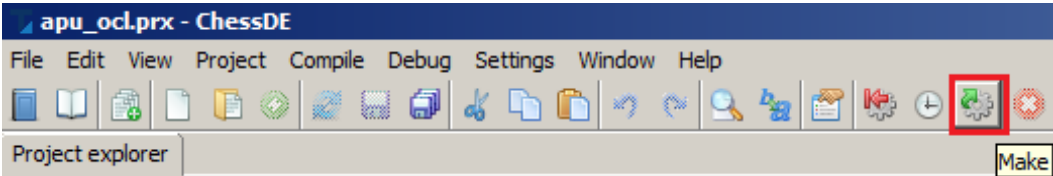


Figure 2-5: Compile

- If compilation completes successfully, open APU_OCL_GEN_LOAD.bat and modify the path of chess_env.bat so it is consistent with your environment.
- Double-click APU_OCL_GEN_LOAD.bat to run the batch file. This will generate an “APU_OCL_LOAD.cpp” file that is ready to be included for host compilation.

	APEX2	16/04/2014 10:35 AM	File folder	
	host	16/04/2014 10:35 AM	File folder	
	host_test	16/04/2014 10:35 AM	File folder	
	APU_OCL	16/04/2014 10:35 AM	Makefile	7 KB
	APU_OCL_APEX	16/04/2014 10:35 AM	Makefile	1 KB
	APU_OCL_GEN_LOAD	16/04/2014 10:35 AM	Windows Batch File	1 KB
	APU_OCL_LOAD	16/04/2014 2:47 PM	C++ Source	109 KB
	APU_OCL_Test	16/04/2014 10:35 AM	Makefile	1 KB

Figure 2-6: Generate APU_OCL_LOAD.cpp

2.4 OpenCL Host Compilation

The following steps describe the process of adding, compiling, and executing the OpenCL examples within the “ApexCLSample” application provided by CogniVue:

- Add your own OpenCL example source code and header file in:
<ROOT>\Applications\Cognivue\ApexCLSamples\src
<ROOT>\Applications\Cognivue\ApexCLSamples\inc
- Copy and paste 'APU_OCL_LOAD.cpp', which was generated by offline compilation in
<ROOT>\Applications\Cognivue\ApexCLSamples\src
- Add the source code into the project by editing the project file directly:

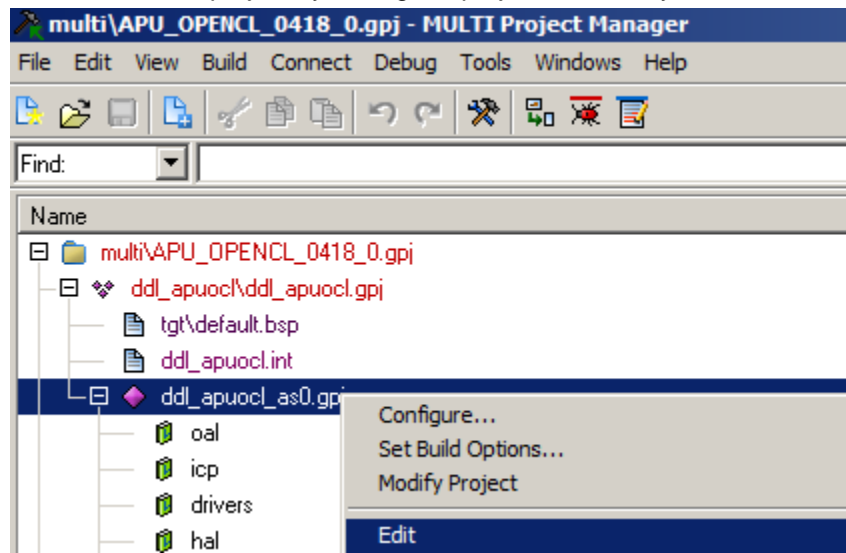


Figure 2-7: Open Project Manager Editor

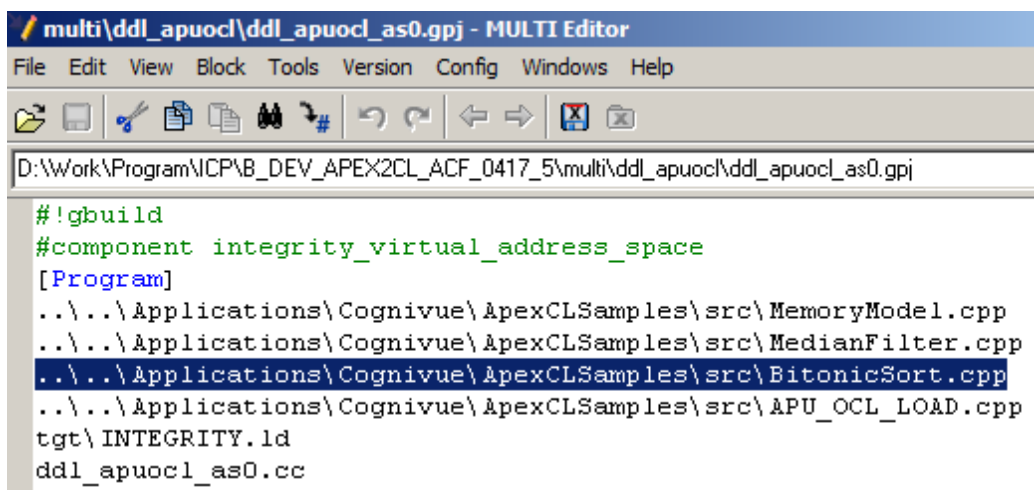


Figure 2-8: Add your file

The added file should appear in the project:

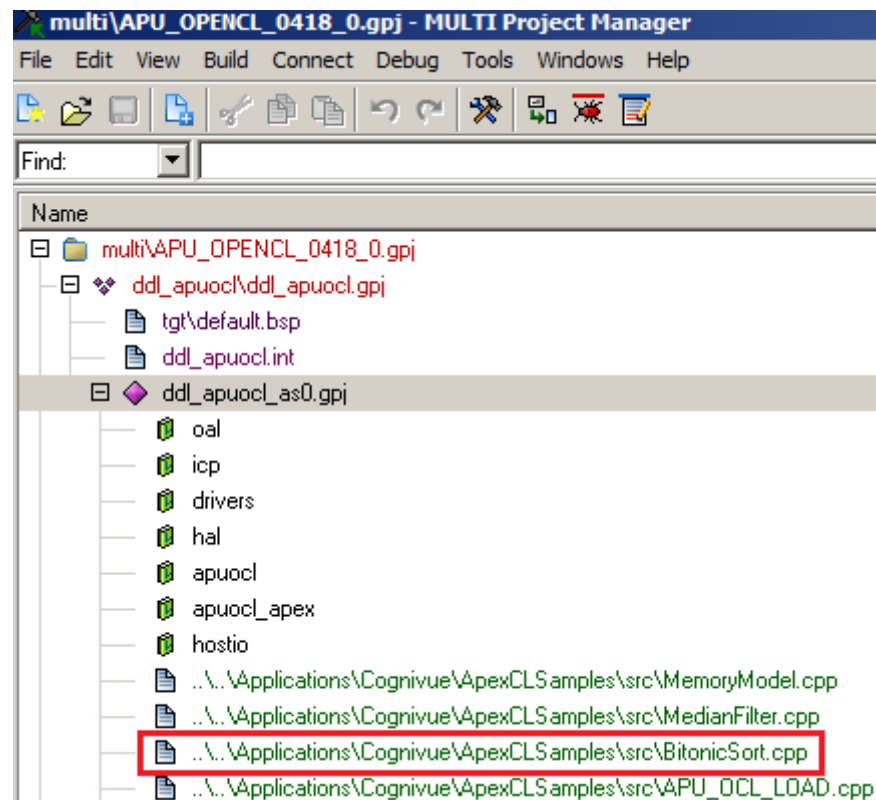


Figure 2-9: Newly added file

- Build > Build Top Project

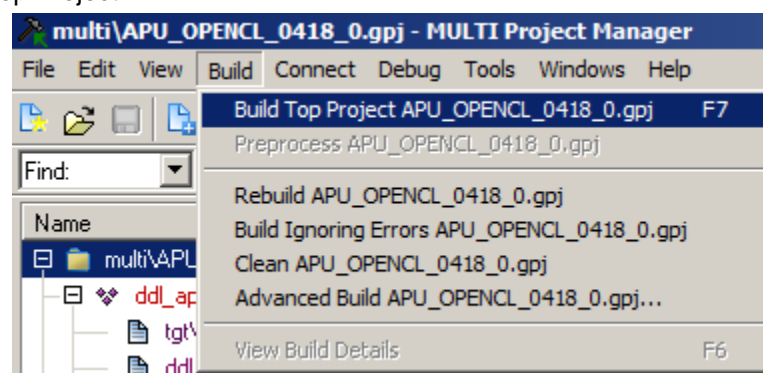


Figure 2-10: Host Builder

3 Package Contents

The APEX-CL SDK package contains the SDK libraries, documentation, tools, developer files and samples.

3.1 Include

Headers for the OpenCL runtime APIs:

`\ICP_SDK\src\Features\ICP\ACL\host\include\CL`

3.2 Libs

Offline device runtime library:

`<ROOT>\ICP_SDK\src\Features\ICP\ACL\APEX2\src\lib\ libapu_ocl_device.a`

Host runtime library:

`<ROOT>\multi\ libapuocl\ libapuocl.gpj`

`<ROOT>\multi\ libapuocl_apex\ libapuocl_apex.gpj`

`<ROOT>\multi\ libdrivers\ libdrivers.gpj`

`<ROOT>\multi\ libhal\ libhal.gpj`

`<ROOT>\multi\ libicp\ libicp.gpj`

`<ROOT>\multi\ liboal\ liboal.gpj`

Example project:

`<ROOT>\multi\dd_apuocl\ dd_apuocl.gpj`

3.3 Tools

Besides the tools that come with this SDK, the following tool(s) is also required:

- APU OpenCL C kernel compiler

3.4 Example kernels and host applications

- Example kernels
 - <ROOT>\ICP_SDK\src\Features\ICP\ACL\APEX2\src\MemoryModel\MemoryModel.cl
 - <ROOT>\ICP_SDK\src\Features\ICP\ACL\APEX2\src\MedianFilter\MedianFilter.cl
 - <ROOT>\ICP_SDK\src\Features\ICP\ACL\APEX2\src\MedianFilter\MedianFilter_Async.cl
 - <ROOT>\ICP_SDK\src\Features\ICP\ACL\APEX2\src\MedianFilter\MedianFilter_Async_2d.cl
 - <ROOT>\ICP_SDK\src\Features\ICP\ACL\APEX2\src\BitonicSortAMD\BitonicSortAMD.cl
 - <ROOT>\ICP_SDK\src\Features\ICP\ACL\APEX2\src\FloydWarshall\FloydWarshall_Kernels.cl
- Host applications
 - <ROOT>\Applications\Cognivue\ApexCLSamples\src\MemoryModel.cpp
 - <ROOT>\Applications\Cognivue\ApexCLSamples\src\MedianFilter.cpp
 - <ROOT>\Applications\Cognivue\ApexCLSamples\src\BitonicSort.cpp
 - <ROOT>\Applications\Cognivue\ApexCLSamples\src\FloydWarshall.cpp

3.4.1 Algorithms

3.4.1.1 MemoryModel

This is a simple sample used to teach developers the concept and use of the four distinct memory regions in OpenCL: Global Memory, Local Memory, Constant Memory, and Private Memory.

The Memory Model kernel sums consecutive elements in a data array and multiplies the sum by a "mask" value. Formally, the kernel is given by

$$m \sum_{k=0}^N a[(b+k) \bmod M]$$

Where ***a*** is the input array of size ***M***, ***b*** is an offset in the array, ***N*** is the number of terms in the summation, and ***m*** is the mask value. Note that the summation wraps around to the beginning of the array if the array index ***b+k*** exceeds ***M***.

3.4.1.2 Median Filter

The median filter processes each pixel in the image and compares it to its neighbors to determine whether this pixel can represent the window entries. It replaces the central pixel value with the median of the pixel values in the window.

To define the median of a window, sort the entries of the window numerically. For windows with an odd number of entries, the median is the value of the middle entry. For windows with an even number of entries, several options are possible.

The following figure illustrates a sample calculation of the median value for a pixel neighborhood:

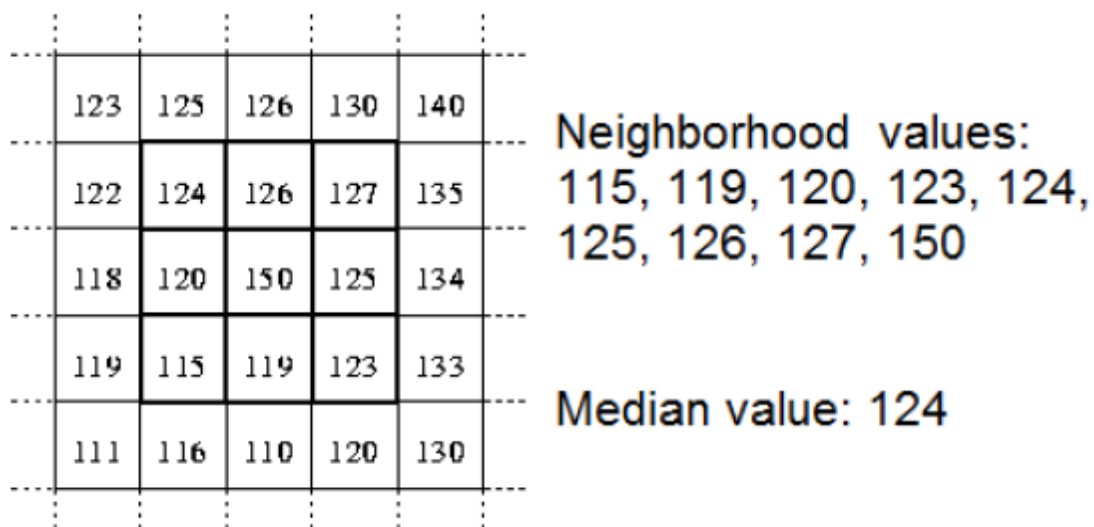


Figure 3-1. Median Filter

This example illustrates a 3x3 square window. As the central pixel value of 150 does not represent the surrounding values well, it is replaced with the median value of 124. Please note that larger windows produce greater smoothing.

The advantage of the median filtering is that unrepresentative pixels in a window cannot have significant effect on the median value. Since the median value must be an actual value of one of the window entries, the median filter does not create new unrealistic pixel values when the filter processes an edge region. Thus, median filtering permits to preserve sharp edges.

3.4.1.3 Bitonic Sort

The bitonic sort is developed on the basis of the 0-1-principle. The 0-1-principle states that a comparator network that sorts every sequence of 0's and 1's is a sorting network. If we see the proof of the 0-1-principle, it is pointed out that the failure in sorting the zero-one sequence in network is contradiction. It is a "proof by contradiction".

Because the explicit sequence '0' and '1' could be compared and sorted. Finally 0-1-principle makes a conclusion "if all 0-1-sequences are sorted by N, then all arbitrary sequences are sorted by N", N is sorting network.

For the complete of sorting input sequence "n" using Bitonic sort, the bitonic sort requires that the number of comparator stages is equal to " $\log(n) \cdot (\log(n)+1) / 2$ " and the Each stage of the sorting network consists of $n/2$ comparators. On the whole, these are comparators.

$$\Theta(n \cdot \log(n)^2)$$

That means the bitonic sort is working in an amount of time proportional to

$$n \cdot \log(n)^2$$

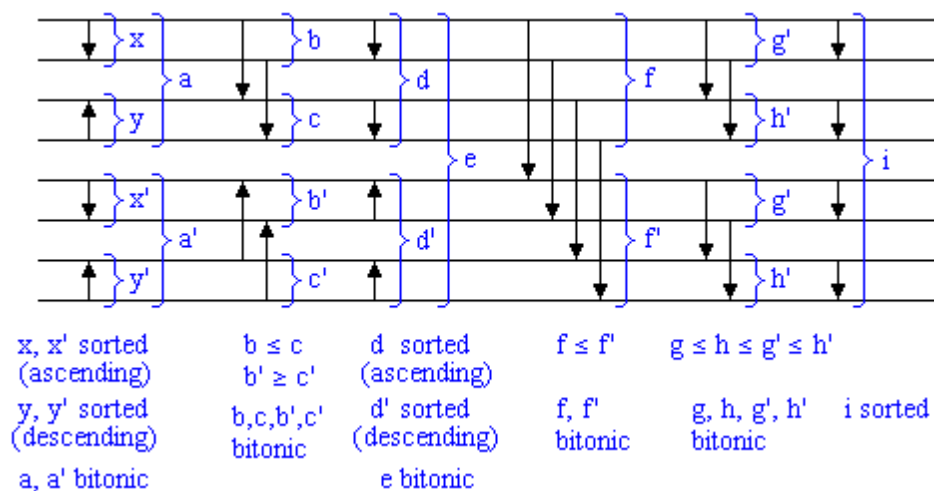


Figure 3-2. Bitonic Sort

3.4.1.4 Floydwarshall

The Floyd-Warshall algorithm computes the shortest path between each pair of nodes in a graph.

It is a dynamic programming approach that iteratively refines the adjacency matrix of the graph in question until each entry in the matrix reflects the shortest path between the corresponding nodes. The main idea of the algorithm is as follows: Given the shortest path between node V_i and V_j using $V_1 \dots V_k$ as intermediate nodes, find out the shortest path between V_i and V_j using $V_1 \dots V_{k+1}$ as intermediate nodes. This idea can be recursively formulated as:

$\text{ShortestPath}(i, j, k) = \min(\text{ShortestPath}(i, j, k-1) , \text{ShortestPath}(i, k, k-1) + \text{ShortestPath}(k, j, k-1))$

$\text{ShortestPath}(i, j, 0) = \text{EdgeCost}(i, j)$

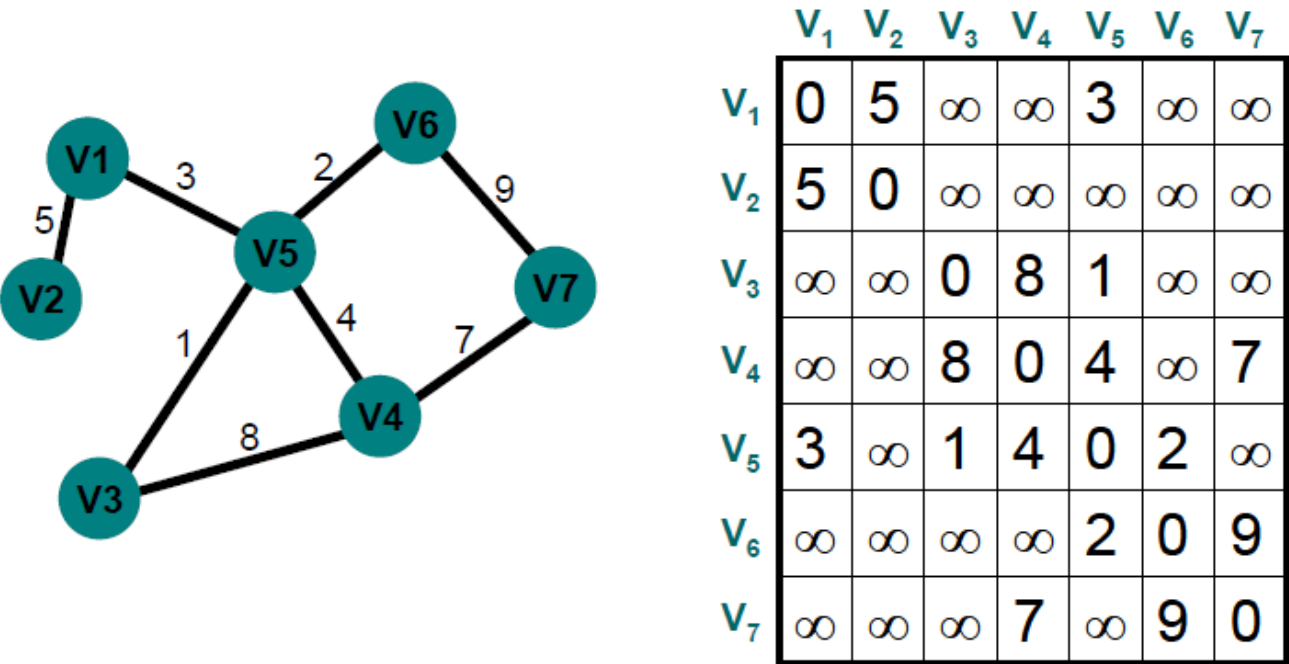


Figure 3-3 A diagram of a graph with seven vertices and eight edges

Appendix A: Release Notes

.1 New and Improved Functions

.1.1 `async_work_group_copy` extension for 2D or 3D range execution:

```
/**
 * @brief Perform an async gather of elements of size num_gentypes_x * num_gentypes_y from src to dst.
 * @dst          destination local buffer
 * @dst_stride    destination buffer stride
 * @src          source buffer
 * @src_stride    source buffer stride
 * @num_gentypes_x number of elements to copy in x dimension
 * @num_gentypes_y number of lines to copy in y dimension
 * @event        event object that can be used with wait_group_events call
 */
event_t async_work_group_copy_2d(    __local gentype* dst, size_t dst_stride,
                                     const __global gentype* src, size_t src_stride,
                                     size_t num_gentypes_x, size_t num_gentypes_y,
                                     event_t event);
```

Note:

This API now supports all OpenCL scalar data types. The generic type name `gentype` indicates the built-in data types `char`, `uchar`, `short`, `ushort`, `int`, `uint`, as the type for the arguments unless otherwise stated.

Sample test kernel for this API is:

<ROOT>\ICP_SDK\src\Features\ICP\ACL\APEX2\src\MedianFilter\MedianFilter_Async_2d.cl

And the sample application for the kernel testing is:

<ROOT>\multi\dd_apuocl\dd_apuocl.gpj

.1.2 ACF native kernel support:

The OpenCL task parallel programming model defines a model in which a single instance of a kernel is executed independent of any index space.

Native kernels are task kernels which execute as a single work item. It allows functions compiled with a traditional compiler rather than the OpenCL compiler flow to be executed within the OpenCL task graph.

Therefore, to queue a native ACF graph within the same command queue as standard OpenCL kernels and operate on the same memory buffer object, the native ACF graph needs to adjust its chunk size dynamically to support different global buffer sizes (note that static chunk might be possible up to a maximum limit, however the performance will not be optimal). It improves the user friendliness and efficiency.

.2 Known Issues

This section describes the known issues that you might run into during kernel compilation stage:

- Regular function call inside of kernel function: need to be inlined;
- Non-supported features in the OpenCL C kernel:
 - Float data type
 - attributes
 - pragmas