AMD **Accelerated** Parallel Processing TECHNOLOGY

AMD APP SDK v2.9

Getting Started

1 How to Build a Sample

Ensure that the AMD APP SDK Samples package has been installed before proceeding.

The following sections describe the steps to build and execute a sample that's included in the APP SDK v2.9. The samples may be OpenCL, OpenCV-CL, C++ AMP, Bolt, or Aparapi samples.

To run pre-built Bolt and C++Amp samples, Microsoft Visual Studio 2012 must be installed on the host operating system.

C++ Amp samples can be built and executed only with Microsoft Visual Studio 2012.

Bolt OpenCL samples can be built and executed with Microsoft Visual Studio versions 2010 and 2012. Bolt C++ AMP samples can be built with Microsoft Visual Studio 2012.\

1.1 BUILDING SAMPLES USING CMAKE

1.1.1 Getting started with CMake

Starting with APP SDK 2.9, the build tool used in the APP SDK is CMake. CMake supports creating make files across different platforms and generating project files across different IDEs including Visual Studio and Eclipse. In order to use CMake with APP SDK, CMake binaries must be downloaded.

1. CMake binaries can be found at: http://www.cmake.org/cmake/resources/software.html.
2. The following page describes how to build a project for Windows and Linux using CMake: http://www.cmake.org/cmake/help/runningcmake.html.

Quick links to download:

1. For Windows: http://www.cmake.org/files/v2.8/cmake-2.8.12-win32-x86.exe.
2. For Linux: The "`sudo apt-get install cmake`" command will install the cmake binaries.

Note: The APP SDK 2.9 package for Windows includes Visual Studio 2010 and 2012 projects in addition to the CMake files. Developers can use the project files for building and for development.

1.1.2 Building an APP SDK sample using CMake

a.Windows: Generating Visual Studio project files –

Developers can create Visual Studio project files for the sample individually or for all of them together. To create the project files individually, CMake must be run by specifying the corresponding sample directory. For example, if you want to create the Visual studio 2012 solution file for the AtomicCounters OpenCL sample, then the corresponding command is:

For 32 bit:

```
cmake.exe -G "Visual Studio 11" <path to the AtomicCounters sample source>
```

For 64 bit:

```
cmake.exe -G "Visual Studio 11 Win 64" <path the to the AtomicCounters sample source>
```

It is recommended to create another directory under the sample and run CMake from there. CMake creates a few additional files and projects apart from the main sample project. Hence running CMake from a sub-directory will keep the project files separate from the sample source files. If run from a subdirectory, the command will change as follows:

```
cmake.exe .. -G "Visual Studio 11" <path to the sub-directory inside the AtomicCounters sample source folder>
```

To build all the OpenCL samples, run the following command

For 32 bit:

```
cmake -G "Visual Studio 11" <APPSDKSamplesInstallPath>/samples/opencl/
```

For 64 bit:

```
cmake -G "Visual Studio 11 Win 64" <APPSDKSamplesInstallPath>/samples/opencl/
```

Alternatively, `cmake-gui` can also be used to build the samples.

b.Linux: Generating Makefiles –

To generate the Makefiles, the same commands mentioned in the Windows section can be used except that the "Generator" name must be changed from the Visual Studio variant to the Unix variant.

For example, to build all the OpenCL samples, run the following command:

```
cmake -G "Unix Makefiles" <APPSDKSamplesInstallPath>/samples/opencl/
```

1.1.3 CMAKE APP SDK-SPECIFIC OPTIONS

1. `BUILD_OPENCL`: ON/OFF - Builds OpenCL samples. Default is ON.
2. `BUILD_OPENCV`: ON/OFF - Builds OpenCV samples .Default is ON.
3. `BUILD_AMP`: ON/OFF - Builds C++ AMP samples. Default is ON.
4. `BUILD_BOLT`: ON/OFF - Builds Bolt samples. Default is ON.

5. BUILD_TBB: ON/OFF - Builds all Bolt samples with the multicore option enabled. Default is OFF.

For example, to build all the samples except OpenCV samples, use -DBUILD_OPENCV=OFF as an argument to cmake or deselect the corresponding box in cmake-gui.

```
cmake -G "Unix Makefiles" _DBUILD_OPENCV=OFF <APPSDKSamplesInstallPath>
```

Note: The APP SDK samples have been tested with the following generators in CMake

1. Visual Studio 10
2. Visual Studio 10 Win64
3. Visual Studio 11
4. Visual Studio 11 Win64
5. NMake Makefiles
6. Unix Makefiles

1.2 OpenCL On Windows

Building With Visual Studio Solution Files –

The samples installation contains a a Microsoft Visual Studio 2010 solution file (OpenCLSamplesVS10.sln) and a Microsoft Visual Studio 2012 solution file (OpenCLSamplesVS12.sln). These files are located at $<*APPSDKSamplesInstallPath*>\samples\opencl\. The solution file contains the entire sample project. To build a sample:

1. Open the OpenCLSamplesVS10.sln file with Microsoft Visual Studio 2010 Professional Edition or the OpenCLSamplesVS12.sln file with Microsoft Visual Studio 2012 Professional Edition.
2. Select *Build* > *Build Solution* to build all solutions.
3. Select the project file in the Solutions Explorer.

Building With Visual Studio Solution Files Using the Intel Compiler (icl) –

The samples installation contains a a Microsoft Visual Studio 2010 solution file (OpenCLSamplesVS10.sln) and a Microsoft Visual Studio 2012 solution file (OpenCLSamplesVS12.sln). These files are located at $<*APPSDKSamplesInstallPath*>\samples\opencl\. The solution file contains the entire sample project. To build a sample:

1. Open the OpenCLSamplesVS10.sln file with Microsoft Visual Studio 2010 Professional Edition or the OpenCLSamplesVS12.sln file with Microsoft Visual Studio 2012 Professional Edition.
2. Right-click on a project file, and select *Properties*.
3. Under Configuration Properties | General, change the Platform Toolset to Intel C++ Compiler, and Click OK.
4. Right-click on the project file, and select *Build* to build the sample.

Building with DirectX –

To build DirectX samples in the package:

1. Ensure that the Microsoft DirectX SDK (June 2010) is installed.
2. Open a Visual Studio command prompt.
3. Run `C:\Program Files (x86)\Microsoft DirectX SDK (June 2010)\Utilities\bin\dx_setenv.cmd`. A successful execution of `dx_setenv.cmd` outputs the following message:

```
Dx x64 target environment is now enabled.

Dx x64 host environment is now enabled.
```

This allows the DirectX samples in the package to be built.

1.3 C++ AMP On Windows

Building With Visual Studio Solution Files –

The samples installation contains a Microsoft Visual Studio 2012 solution file (`C++AmpSamplesVS12.sln`). This file is located at `$<APPSDKSamplesInstallPath>\samples\C++AMP\`.

Note: C++ AMP is supported only by Microsoft Visual Studio 2012. Also, C++ AMP samples do not work on Linux.

The solution file contains the entire sample project. To build a sample:

1. Open the `C++AmpSamplesVS12.sln` file with Microsoft Visual Studio 2012 Professional Edition.
2. Select *Build* > *Build Solution* to build all solutions.
3. Select the project file in the Solutions Explorer.
4. Right-click on the project file, and select *Build* to build a particular sample.

1.4 Bolt On Windows

Installing of Bolt 1.1 AND TBB Libraries –

1. Ensure that your system contains the following prerequisites for Bolt:
   - Windows 7/8
   - Microsoft Visual Studio version 2010 or higher (Microsoft Visual Studio 2012 for APP SDK Bolt C++ AMP samples)
   - CMake version 2.8.10 and higher (required only if one needs to build Bolt libraries from source)
   - TBB library (For Multi-core CPU path, BOLT is tested with 4.2)
2. Download the TBB libraries from http://threadingbuildingblocks.org/download - 4.2 and install the binaries.

3. Set the environmental variable `TBB_ROOT` to the root directory of the installed TBB binaries.
4. Append the `PATH` environment variable with the directory containing all the TBB .dll files. For example, for a 64-bit machine with VS 2012, this path will be: `%TBB_ROOT%\bin\intel64\vc11\`.
5. Download and install the Bolt 1.1 prebuilt binaries from http://developer.amd.com/tools-and-sdks/heterogeneous-computing/amd-accelerated-parallel-processing-app-sdk/bolt-c-template-library/.
   Bolt can be also be built from the `github` sources.

4. Set the `BOLTLIB_DIR` environmental variable to the root directory of the Bolt binaries.

Building With Visual Studio Solution Files –

The samples installation contains a Microsoft Visual Studio 2010 solution file (`BoltSamplesVS10.sln`) and Microsoft Visual Studio 2012 solution file (`BoltSamplesVS12.sln`). This file is located at `$<APPSDKSamplesInstallPath>\samples\Bolt\`.

Note: Bolt samples can be built with only Microsoft Visual Studio 2010 and Microsoft Visual Studio 2012 on Windows.

The solution file contains the entire sample project. To build a sample:

1. Open the `BoltSamplesVS10.sln` or `BoltSamplesVS12.sln` file with the appropriate Microsoft Visual Studio Professional Edition.
2. Select *Build* > *Build Solution* to build all solutions.
3. Select the project file in the Solutions Explorer.
4. Right-click on the project file, and select *Build* to build a particular sample.

1.5 OpenCV on Windows and Linux

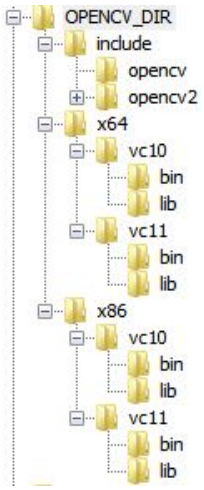
APP SDK v2.9 OpenCV samples require OpenCV 2.4.4 to be installed as a prerequisite. The prebuilts of OpenCV 2.4.4 do not have the OpenCV-CL libraries in them. To compile and run the OpenCV samples in APP SDK v2.9, you must build from the source files of OpenCV 2.4.4, enabling the OpenCL flag in CMake.

Before building and running OpenCV-CL samples, perform the following actions:

1. Build the OpenCV library from the source files.
   The OpenCV sources for the 2.4.4 build are available at https://github.com/Itseez/opencv/tree/2.4.4. To have OpenCL support for OpenCV, the `opencv_ocl` library must be built. During configuring with CMake, select the `WITH_OPENCL` option and provide the path of the OpenCL library (`libOpenCL.so` in Linux and `OpenCL.lib` in Windows). The following links from opencv.org are useful:
   - Linux: http://docs.opencv.org/doc/tutorials/introduction/linux_install/linux_install.html
   - Windows: http://docs.opencv.org/doc/tutorials/introduction/windows_install/windows_install.html
2. Set the correct directory structure.
   On Windows, the directory structure of the complied OpenCV binaries created in the

preceding step must be restructured as per the OpenCV prebuilt directory structure. This step ensures that, when future OpenCV releases include the OpenCV-CL components in the prebuilts, developers do not need to change the path of the OpenCV library references in their projects.
Restructure the OpenCV binaries created from sources as per the prebuilt directory structure for Windows as shown in the following figure. Place the created `include` directly under the root OpenCV directory and copy the `bin` and `lib` files (both debug and release versions) into the following locations:

- x86/vc10, if your binaries and libraries are built with Microsoft Visual Studio 10 and the target is x86.
- x86/vc11, if your binaries and libraries are built with Microsoft Visual Studio 12 and the target is x86.
- x64/vc10 if your binaries and libraries are built with Microsoft Visual Studio 10 and the target is x64.
- x64/vc11 if your binaries and libraries are built with Microsoft Visual Studio 12 and the target is x64.



Alternatively, you may skip restructuring the directory as per the OpenCV pre-builts, in which case, the Visual Studio property sheets of the OpenCV-CL projects must be updated to point to the correct paths.

For Linux, retain the default directory structure created from sources.

3. Set the environment variables:
   - Create and set the environmental variable, `OPENCV_DIR`, to the root directory containing the OpenCV include and lib files created in the preceding step.
   - Create and set the environmental variable, `OCVCL_VER`, to the OpenCV version used for APP SDK 2.9 release, that is, set `OCVCL_VER = 244`.
   - For Windows, set `PATH` to the directory containing all the OpenCV .dll files. For example, for a 64-bit machine with VS 2012, this path will be `%OPENCV_DIR%\x64\vc11\bin`. If you have built OpenCV with `AMD BLAS` and `FFT`, then set those paths also

- For Linux, export `LD_LIBRARY_PATH` to the directory containing all the OpenCV shared object files.

For answers to frequently asked questions on OpenCV, see the AMD APP SDK FAQ.

1.5.1 Download and Installation of OpenNI libraries

The GestureRecognition APP SDK OpenCV-CL sample makes use of OpenNI libraries to extract video frames. OpenNI framework is an open source SDK used for the development of 3D sensing middleware libraries and applications. OpenNI SDK can be downloaded from the following links:

1. Windows 32-bit: http://www.openni.org/openni-sdk/?download=http://www.openni.org/wp-content/uploads/2013/07/OpenNI-Windows-x86-2.2.zip.
2. Windows 64-bit: http://www.openni.org/openni-sdk/?download=http://www.openni.org/wp-content/uploads/2013/07/OpenNI-Windows-x64-2.2.zip.
3. Linux 32-bit: http://www.openni.org/openni-sdk/?download=http://www.openni.org/wp-content/uploads/2013/07/OpenNI-Linux-x86-2.2.tar.zip.
4. Linux 64-bit: http://www.openni.org/openni-sdk/?download=http://www.openni.org/wp-content/uploads/2013/07/OpenNI-Linux-x64-2.2.tar.zip.

Environment variables to be set for OpenNI:

1. Windows
    i. For 32-bit builds, add `OPENNI2_REDIST` to `PATH`.
    ii. For 64-bit builds, add `OPENN2_REDIST64` to `PATH`.

Note: Include and library paths will be added by the OpenNI Windows installer .If those paths are missing from the system, then the user must set the environment variables mentioned in the Linux section.

2. Linux
    i. For 32-bit platforms:
    Export `OPENNI2_INCLUDE` to `<<OPENNI-INSTALL_PATH>>/Include`
    Export `OPENNI2_LIB` to `<<OPENNI-INSTALL_PATH>>/Redist`
    Add `OPENNI2_LIB` to `LD_LIBRARAY_PATH`
    ii. For 64-bit platforms:
    Export `OPENNI2_INCLUDE64` to `<<OPENNI-INSTALL_PATH>>/Include`
    Export `OPENNI2_LIB64` to `<<OPENNI-INSTALL_PATH>>/Redist`
    Add `OPENNI2_LIB64` to `LD_LIBRARAY_PATH`

2 How to Run the Application

This section describes how to run the application that was just built.

2.1 On Windows

There are three ways to run the application: by using Microsoft Visual Studio 2010 Edition, using Microsoft Visual Studio 2012 Edition, or using the command line. Note that Microsoft Visual Studio 2010 cannot be used for all the samples, as indicated in the earlier sections. The following

description is applicable for OpenCL samples only and is to be used as a general guide for the other classes of samples.

Using Visual Studio

1. Open `OpenCLSamplesVS10.sln` with Microsoft Visual Studio 2010 Professional Edition, or `OpenCLSamplesVS12.sln` with Microsoft Visual Studio 2012 Professional Edition, and build it.
2. Select the desired project file in the Solutions Explorer.
3. Right-click on it, and select *Set as StartUp Project*. To run the application, press Ctrl+F5. To run the application in debug mode, simply press F5.

Using the Command Line

1. Open a command prompt.
2. Go to the `$<APPSDKSamplesInstallPath>/samples/opencl/bin`.
3. Go into the appropriate architecture directory (`x86` or `x86_64`) and further into the config directory (Debug/Release).
4. Run the samples by typing the name of their executables. See the individual sample documents for their respective command line arguments.

2.2 On Linux

1. Ensure the path is set to include the location of `libOpenCL.so`.
2. Open a terminal window.
3. Go to the `$<APPSDKSamplesInstallPath>/samples/opencl/bin/` directory.
4. Go into the appropriate architecture directory (`x86` or `x86_64`).
5. Run the samples by typing the name of their executables. You may have to prepend the executable name with `./ `. See the individual sample documents for their respective command line arguments.

Note: -The prebuilt samples on Linux have been compiled with GCC 4.7.3.

2.3 Sample Code

The simplest OpenCL sample in the SDK is the HelloWorld sample. It is for developers that are new to OpenCL programming. See the HelloWorld sample documentation (in `$<APPSDKSamplesInstallPath>/samples/opencl/cl/HelloWorld/docs`), which explains the workflow for setting up a basic AMD APP application using OpenCL. The HelloWorld sample is included for instructional purposes.

For basic debugging techniques, see the BasicDebug sample.

There are two kinds of OpenCL samples in the AMD APP SDK. One is written using native OpenCL calls (in `$<APPSDKSamplesInstallPath>/samples/opencl/cl/`); the other is written using the AMD C++ bindings to OpenCL (in `$<APPSDKSamplesInstallPath>/samples/opencl/cpp_cl`).

Most of the samples make use of a set of Utility header files. These header files contain all the utility functions such as parsing command line options, loading and writing bitmap images,

printing formatted output, comparing results, and reading files. The samples include the required utility header files. The SDKUtil header files are available in the $<*APPSDKSamplesInstallPath*>\include\SDKUtil folder.

3 Aparapi

The samples for Aparapi are different from the other samples because all the samples are coded in Java. These samples can be executed on Windows and Linux; they do not require a particular version of Microsoft Visual Studio.

To build and run the Aparapi examples requires the following environment variables to be configured:

- Set JAVA_HOME to the directory containing JRE/JDK, version 1.7 or above.
- Set ANT_HOME to the directory containing ANT, version 1.8 or above.
- Set LIBAPARAPI to the directory where aparapi-2012-11-14.zip (or above) is unzipped.
- Ensure that PATH is set such that the java and javac executables are used from JDK version 1.7 or above, and the ant executable is used from ANT_HOME.
- In Linux, set LD_LIBRARY_PATH to the directory where aparapi-2012-11-14.zip (or above) is unzipped.

To build the example, first cd into <*APPSDKSamplesInstallPath*>\samples\aparapi\AparapiUtilfolder.

If building from the command line: Type ant to create AparapiUtil.jar.

If building on Eclipse, open the AparapiUtil project and build.

Note: Before importing projects into Eclipse, do the following:

- On Linux: In the $<*APPSDKSamplesInstallPath*>/samples/aparapi, run sh ./setenv_aparapi.sh.
- On Windows: In the $<*APPSDKSamplesInstallPath*>/samples/aparapi, run setenv_aparapi.bat.

This updates the .classpath for all the examples, which allows Aparapi projects to be successfully imported and built using Eclipse.

Then, change the directory to <*APPSDKSamplesInstallPath*>\samples\aparapi\<example>

If building from the command line: Type ant to create <example>.jar.

If building on Eclipse, open the <example> project, and build.

To run the sample on Windows or Cygwin, run the appropriate <example>.bat from the command line. Right-clicking on the <example> project in Eclipse and running it has the same effect. <example>.bat -h lists the options exposed by the sample.

To run the sample on a Linux terminal, run the appropriate <example>.sh.

4 Important Notes

- Unless specifically recommended otherwise, developers must use the latest graphics drivers for their platform. These drivers can be downloaded from http://support.amd.com/us/Pages/AMDSupportHub.aspx.

  For current recommendations, click here.
- The following values are returned when querying strings from OpenCL:

  `CL_PLATFORM_VERSION`: OpenCL 1.2 AMD-APP (*build.revision*)
  `CL_PLATFORM_NAME`: AMD Accelerated Parallel Processing
  `CL_PLATFORM_VENDOR`: Advanced Micro Devices, Inc.
- Check the Platform Vendor string, not the Platform Name, to determine AMD hardware. For example code that shows how to check and use the `CL_PLATFORM_VENDOR` string, see the AMD APP OpenCL samples.

5 Supported Devices

We are continually qualifying devices. For an up-to-date list of supported devices, please visit the APP SDK System Requirement and Driver Compatibility page.

Contact

Advanced Micro Devices, Inc.
One AMD Place
P.O. Box 3453
Sunnyvale, CA, 94088-3453
Phone: +1.408.749.4000

For AMD Accelerated Parallel Processing:

URL: developer.amd.com/appsdk
Developing: developer.amd.com/
Forum: developer.amd.com/openclforum