# AMD CodeXL Quick Start Guide

AMD Developer Tools Team  
Advanced Micro Devices, Inc.

Version 1.8  
Revision 1

## Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>3</td>
</tr>
<tr>
<td>LATEST VERSION OF THIS DOCUMENT</td>
<td>3</td>
</tr>
<tr>
<td>PREREQUISITES</td>
<td>3</td>
</tr>
<tr>
<td>DOWNLOAD AND INSTALL CODEXL</td>
<td>4</td>
</tr>
<tr>
<td>CODEXL HELP</td>
<td>5</td>
</tr>
<tr>
<td>SYSTEM INFORMATION</td>
<td>6</td>
</tr>
<tr>
<td>TEAPOT SAMPLE PROJECT</td>
<td>7</td>
</tr>
<tr>
<td>Debug the Teapot Sample Application</td>
<td>9</td>
</tr>
<tr>
<td>Basic Debugging</td>
<td>9</td>
</tr>
<tr>
<td>Source Code View</td>
<td>10</td>
</tr>
<tr>
<td>Breakpoint View</td>
<td>11</td>
</tr>
<tr>
<td>Watch and Locals Views</td>
<td>12</td>
</tr>
<tr>
<td>Explorer View</td>
<td>14</td>
</tr>
<tr>
<td>Call Stack View</td>
<td>17</td>
</tr>
<tr>
<td>Function Calls History View</td>
<td>17</td>
</tr>
<tr>
<td>Debugged Process Events View</td>
<td>17</td>
</tr>
<tr>
<td>Memory View</td>
<td>18</td>
</tr>
<tr>
<td>Statistics View</td>
<td>18</td>
</tr>
<tr>
<td>MATRIX MULTIPLICATION PROJECT Project</td>
<td>19</td>
</tr>
<tr>
<td>Perform CPU Profile for the Matrix Multiply Sample Application</td>
<td>20</td>
</tr>
<tr>
<td>CPU Time Based Profile Navigation</td>
<td>20</td>
</tr>
<tr>
<td>Source Code View</td>
<td>21</td>
</tr>
<tr>
<td>Run the classic textbook sample</td>
<td>22</td>
</tr>
<tr>
<td>Analyzing the classic implementation</td>
<td>23</td>
</tr>
<tr>
<td>Analyzing the improved implementation</td>
<td>25</td>
</tr>
<tr>
<td>PROFILE MODE</td>
<td>26</td>
</tr>
<tr>
<td>CPU Profiling</td>
<td>27</td>
</tr>
<tr>
<td>Overview Tab</td>
<td>28</td>
</tr>
<tr>
<td>Modules Tab</td>
<td>28</td>
</tr>
<tr>
<td>Call Graph Tab</td>
<td>29</td>
</tr>
<tr>
<td>Functions Tab</td>
<td>30</td>
</tr>
<tr>
<td>GPU Profiling</td>
<td>31</td>
</tr>
<tr>
<td>Summary Tab</td>
<td>32</td>
</tr>
<tr>
<td>Performance Counters View</td>
<td>33</td>
</tr>
<tr>
<td>CodeXL Explorer Tree</td>
<td>34</td>
</tr>
<tr>
<td>Power Profiling</td>
<td>35</td>
</tr>
<tr>
<td>Switching to Power Profiling mode</td>
<td>35</td>
</tr>
<tr>
<td>Starting a new Power Profiling session</td>
<td>35</td>
</tr>
<tr>
<td>Setting the Sampling Interval</td>
<td>36</td>
</tr>
<tr>
<td>Stopping a Power Profiling session</td>
<td>36</td>
</tr>
<tr>
<td>Power Profiling Real-Time Values</td>
<td>36</td>
</tr>
<tr>
<td>Power Profiling Timeline View</td>
<td>37</td>
</tr>
<tr>
<td>Power Profiling Summary View</td>
<td>38</td>
</tr>
<tr>
<td>Configuring Power Profiler Sessions</td>
<td>38</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>ANALYZE MODE</td>
<td>39</td>
</tr>
<tr>
<td>Static Kernel and Shader Analysis</td>
<td>39</td>
</tr>
<tr>
<td>Switching to Analysis mode</td>
<td>39</td>
</tr>
<tr>
<td>Creating a new project for Kernel Analysis</td>
<td>40</td>
</tr>
<tr>
<td>Adding OpenCL files to an existing project</td>
<td>40</td>
</tr>
<tr>
<td>Analyze Mode Options</td>
<td>41</td>
</tr>
<tr>
<td>Build Options - defining kernel/shader compilation options</td>
<td>42</td>
</tr>
<tr>
<td>Building a kernel/shader file</td>
<td>45</td>
</tr>
<tr>
<td>Output Tab</td>
<td>46</td>
</tr>
<tr>
<td>Statistics Tab</td>
<td>46</td>
</tr>
<tr>
<td>Viewing compilation output: ISA and AMD IL</td>
<td>47</td>
</tr>
<tr>
<td>Navigating through ISA code with the Enhanced ISA View</td>
<td>48</td>
</tr>
<tr>
<td>KNOWN ISSUES</td>
<td>49</td>
</tr>
<tr>
<td>SUPPORT</td>
<td>49</td>
</tr>
</tbody>
</table>
Introduction

AMD CodeXL™ is a tool suite with a unified user interface that lets you harness the benefits of AMD CPUs, GPUs, and APUs. It has powerful capabilities for APU/GPU debugging, CPU and GPU profiling, and static OpenCL™ kernel analysis. These features let you find bugs, optimize application performance, and easily access heterogeneous computing. AMD CodeXL is available as a stand-alone application for Windows® and Linux®, as well as a Microsoft® Visual Studio® extension for Windows.

Getting the most out of the AMD CodeXL tool suite requires a relatively recent AMD APU, a recent version of Catalyst, and the OpenCL APP SDK.

This document describes how to

- get started using CodeXL
- find information about known CodeXL issues
- contact AMD for support

Latest Version of This Document

- For the latest and greatest version of the documentation, go to the CodeXL Website.

Prerequisites

Operating Systems

- Microsoft Windows 7 64-bit
- Microsoft Windows 8.1 64-bit
- Microsoft Windows 10 64-bit
- Linux 64-bit (Red Hat, Ubuntu, SUSE)

For detailed system requirements see the CodeXL Release Notes in the CodeXL installation folder or on the Documentation section of the CodeXL web page.

CodeXL Visual Studio Extension


Profiling OpenCL™ Applications

- [GPU device] AMD Catalyst driver with OpenCL™ GPU support
• [GPU device] AMD Radeon™ HD 5000 series or newer
• AMD APP SDK (requirements)

For detailed system requirements see the CodeXL Release Notes in the CodeXL installation folder or on the Documentation section of the CodeXL web page.

Download and Install CodeXL

Installation is system-specific (Windows or Linux); but once installed and started, the CodeXL operation is system-independent.


**For Windows**
1. Download the .exe file AMD_CodeXL_Win*.exe.
2. When the download completes, double-click the .exe file to install CodeXL.
   - The installer guides you through the installation process.
   - The CodeXL Visual Studio 2010, 2012 and 2013 extensions are part of the installer package and are installed by default.
3. Choose “Custom” installation, and de-select the Visual Studio extensions if you do not want to install them.
4. Launch CodeXL from C:\Program Files (x86)\AMD\CodeXL\CodeXL.exe or from the created Desktop shortcut.

**For Linux**

Either install the dedicated distribution package or use the tar archive.

The default installation folder

**For Red Hat and other Fedora based Linux distributions**
1. Download the 64-bit Linux .rpm package AMD_CodeXL_Linux*.rpm.
2. Install the RPM package:
   - $ sudo rpm -Uvh AMD_CodeXL_Linux*.rpm
4. Launch CodeXL using ./CodeXL.

**For Ubuntu and other Debian based Linux distributions**
1. Download the 64-bit Linux .deb package amdcodexl_*.*.deb.
2. Install the DEB package:
   - $ sudo dpkg -i amdcodexl_x.x-x_amd64.deb
   - $ sudo apt-get -f install
4. Launch CodeXL using ./CodeXL.
Use the tar archive

1. Download the 64-bit Linux .tar package AMD_CodeXL_Linux*.tar.gz.
2. Extract the tar package:
   
   ```
   $ tar --xvzf AMD_CodeXL_Linux*.tar.gz
   ```
3. Install the PowerProfile driver:
   
   ```
   $ sudo <codexl-install-dir>/AMDTPwrProfDriverInstall.run
   ```
4. Navigate to <codexl-install-dir>
5. Launch CodeXL using ./CodeXL.

CodeXL Help

To bring up a CodeXL Help window:

   OR
2. Select Help >> View Help from the CodeXL toolbar.

CodeXL Help provides some of the same information provided in this document, but also includes additional details about CodeXL views and modules.
To bring up the CodeXL Help window for the Visual Studio extension:

1. Select CodeXL >> Help >> View Help from the VS menu.

**System information**

To display system information:


The tabs let you select a category of information. The following screenshot shows OpenCL device information for a GPU device and a CPU device on the runtimes available locally - a 32-bit and 64-bit runtime.

To display project settings, the project must be stopped.

To edit the settings of a project:

1. Select File >> Project Settings from the drop-down File menu.
See the CodeXL Help for more details about project settings.

**Teapot Sample Project**

The CodeXL distribution includes a sample project that displays a smoking teapot. The project uses OpenCL kernels to solve Navier-Stokes equations. It shares a 3D texture between OpenCL and OpenGL, copies a density field grid into the 3D texture, and renders the smoke using OpenGL.

For the Visual Studio extension:

1. Select CodeXL >> Open Teapot Sample Project from the VS menu. Visual Studio displays the teapot sample project.
Screenshots in the remainder of this document show the standalone version of CodeXL. The Visual Studio version is similar, but contains a VS window rather than a CodeXL window.

For Windows or Linux:

1. In the CodeXL welcome page (in the CodeXL menu bar, click on File- >Welcome Page), Under the Samples header, click the AMD Teapot link.

The CodeXL Explorer view now shows:

AMDTTeaPot | Debug Mode - Not running

The CodeXL window also displays several other views, but since the program is not running, those views do not display any information.
Debug the Teapot Sample Application

Note: Before debugging the Teapot sample application, you must load it (see the previous section).

After the teapot sample is loaded, run the debug program:

1. Select Debug >> Start Debugging from the taskbar,

   or

2. Click on the green right arrow taskbar.

The program begins execution, and soon displays a rotating smoking teapot in a separate window.

To stop the program:

1. Select Debug >> Stop Debugging from the taskbar,

   or

2. Click the black square taskbar Stop button ,

   or

3. Click the close button in the upper-right corner of the teapot window.

Basic Debugging

The CodeXL GPU Debugger lets you examine the runtime behavior of your OpenCL/OpenGL application in detail. You can use the information it provides to find bugs and to improve application performance. You can debug OpenCL
kernels, inspect variable values across different work items and work groups, and inspect call stacks, among other things.

This quick start guide presumes you are familiar with the use of a GUI debugger; so the guide provides only a quick introduction to the basic CodeXL debugging features.

The following three buttons, at the far left of the CodeXL taskbar, let you select Debug mode, Profile mode or Analyze mode.

Hovering over a taskbar button displays a pop-up help description.

The following taskbar buttons control program execution during debugging.

These controls are (left to right): start, frame step, draw step, step over, step in, step out, break, and stop debugging. You can also perform these actions from the taskbar Debug pull-down menu, or by using function keys.

The following taskbar buttons show, or hide, various views.

These buttons are (left to right): CodeXL Explorer, Properties, Function Calls History, Debugged Process Events, Call Stack, Locals, Watch, OpenGL™ State Variables, OpenCL Multi-Watch (1,2,3), Breakpoints, Memory, and Statistics.

You can resize views, drag, and drop views to rearrange them, or move them to a separate window. The next sections of this guide describe individual CodeXL views in more detail.

**Source Code View**

Source Code views display C, C++, or OpenCL code. To display the Source Code view:

1. Start the teapot program, as described above.
2. Hit the Break button \(\text{\texttt{\textbackslash}}\) to interrupt it.
   
   A Source Code view displays the source file where the break occurred, with a yellow arrow \(\text{\texttt{\textbackslash}}\) indicating the current line number. In the following screenshot, it is line 431 in the `amdtteapottogc canvas.cpp` file.
Breakpoint View

The Breakpoint view shows active breakpoints. Initially, the Breakpoint view shows no breakpoints:

To add a breakpoint:

1. Double-click “Double-click to add or remove breakpoints...”
   A new Breakpoints window appears.
2. Select the API Functions tab to set a breakpoint on an API function, or select the Kernel Functions tab to set a breakpoint on a kernel function. When program execution hits a breakpoint, the Source view displays the line where the breakpoint occurs. A yellow arrow indicates the current location. A red dot next to the line number indicates a set breakpoint.

**Watch and Locals Views**

The Watch view shows the values and types of program variables you specify. The Locals view displays the values and types of local variables in a kernel.
In the image above, the Watch view displays the value of variable `dPlaneDist`. The Locals view displays the values of all local variables in the current kernel (in this case, `computeIntersection` in `tpVolumeSlicing.cl`). For a structured variable, click on the triangle to the left of the variable name to see the name and value of each member.

When CodeXL is in Kernel Debugging mode, move the mouse cursor to hover over any variable name in the OpenCL kernel source code to display a tooltip with the variable value. This is demonstrated in the screenshot below.
A Multi-Watch view lets you compare the values of an OpenCL kernel variable across work items and work groups.

**Explorer View**

The Explorer view displays OpenCL-allocated objects and OpenCL/OpenGL shared contexts.
1. Click on an object to bring up information about the object in the Properties view.
   For example, clicking on Texture 2 in the view above brings up its properties, as shown in the next screenshot.

2. Click on Vertex Buffer object VBO 1 to display its data, with a variety of available drop-down menu display and format options in the right-hand panel.
3. Double-click on an object to display an appropriate view. For example, double-click on Vertex Shader 1 under Shaders to bring up a Source Code view of its source file tpVertexSharder.glsl. Alternatively, double-click on Depth buffer to bring up an Image view of the depth buffer.

You can manipulate an Image view with the following image manipulation buttons on the CodeXL toolbar:
These buttons let you select, zoom in, zoom out, pan, enable R/G/B/alpha channels, enable grayscale mode, enable color invert mode, original size, best fit, and rotate CCW/CW. Hovering over the image displays pixel-specific information (position and color) in the Image Information panel.

Alternatively, select the Data view tab of the depth buffer to display the buffer as raw spreadsheet data rather than as an image.

**Call Stack View**

The Call Stack view displays a combined C/C++/OpenCL call stack.

**Function Calls History View**

The Function Calls History view displays a log of OpenCL API calls.

1. Click on a function call to display call details in a Properties view.

**Debugged Process Events View**

The Debugged Process Events view displays process events.
Memory View

The Memory view summarizes memory use.

Statistics View

The Statistics view provides statistical information about the program. Select a tab to choose among options, such as Function Types:

or Function Calls:
Matrix Multiplication Project

The CodeXL distribution includes a CPU Profile sample. The sample includes 3 functions that implement matrix multiplication.

Use command line arguments to call each of the 3 implementation:

- Running the sample **without any argument** will invoke inefficient implementation of matrix multiplication.
- Running the sample with `-c` will invoke the classic textbook implementation of matrix multiplication.
- Running the sample with `-i` will invoke improved implementation of matrix multiplication.

For the Visual Studio extension:

1. Select CodeXL >> Open Matrix Multiplication Sample Project from the VS menu. Visual Studio displays the matrix multiply sample project.

Screenshots in the remainder of this document show the standalone version of CodeXL. The Visual Studio version is similar, but contains a VS window rather than a CodeXL window.

For Windows or Linux:

2. In the CodeXL welcome page (in the CodeXL menu bar, click on File->Welcome Page), Under the Samples header, click the **AMD Matrix Multiply** link.
The CodeXL Explorer view now shows:

AMD Matrix Multiply | Profile Mode (CPU:Time-Based Sampling) - Not running

**Perform CPU Profile for the Matrix Multiply Sample Application**

**Note:** Before profiling the matrix multiply sample application, you must load it (see the previous section).

After the sample is loaded, run a Time-Based profile session for the sample:

1. Select CodeXL >> Profile >> Time-Based Sampling to switch to Time Based CPU Profile mode.
2. Start Profile the inefficient implementation:
   
   Click CodeXL -> Start Profiling or Click on the green right arrow taskbar. (Local means local host profiling. Use the right black arrow to configure remote host settings)

The program begins execution, and soon displays a command line window that will run the matrix multiplication executable. Once the sample execution completes, CodeXL will open a CPU Profile session overview that displays the profile results.

**CPU Time Based Profile Navigation**

After the execution is complete, CodeXL display a profile session overview window.
This screenshot display the session overview window. See marked red rectangles:

1. **CodeXL Explorer** - The current profile session selected in CodeXL explorer. Double click on this node in the tree will open the session after it is closed.
2. **Functions view** – display the 5 most sampled functions. See that the function “inefficient_multiply_matrices” was sampled 490 times.
3. **Modules view** – display the 5 most sampled modules. In this example this table is not useful. Use it to find inefficient modules in multiple modules executables.
4. **Profile Overview** – displays general information of the session. Executable path, working directory, etc’.

**Source Code View**

While looking at the overview of this session, we can see that the function “inefficient_multiply_matrices” is consuming resources. Right click on this function, and click “Open Source Code”.
Clicking this will open the source code view for the file containing “inefficient_multiply_matrices”.

The source code view display a line-by-line performance table for the requested function. In this sample, we can see, marked in red in the above screenshot, that line 126 had 100% of the samples for this function. Looking at the marked comment we can see that the function is called 3 times, which is redundant.

To call the “classic_multiply_matrices” only once, we will change the command line arguments.

**Run the classic textbook sample**

In order to call “classic_multiply_matrices” in this sample, user should give “-c” command line argument. In order to configure the command line argument sent for the session, open project session window.

1. Click File->Project Settings
2. Go to the “General” page
3. Click “-c” in the command line argument text box
4. Click OK.
5. Run the profile session again (click Profile -> Start Profile)
Analyzing the classic implementation

Looking at the session again, we can see that “classic_multiply_matrices” was sampled 163 times (which is expected, since we call it once, instead of three times).
times). Right click on “classic_multiply_matrices”, and click “Open Source Code” to open the source code again for further analysis.

The source code shows the following line was sampled 157 times.

\[
\text{sum} = \text{sum} + \text{matrix}_a[\text{row}][\text{inCol}] \times \text{matrix}_b[\text{inCol}][\text{outCol}];
\]

In order to look for ideas how to improve this line, we can run an Assess Performance session, to see how our sample is consuming the current system resources.

1. Click on Profile -> CPU: Assess Performance to select this profile type.
2. Click on Profile -> Start Profile to run an assess performance session.
3. Look at the displayed session.
4. Right click on “classic_multiply_matrices” to see a line-by-line display of the system resources.
In the source code view, look at the values for each of the counters for the specified line. We can see that we have many data cache misses, which might cause a performance bottle neck.

**Analyzing the improved implementation**

The third sample implementation is called “improved_matrix_multiplication”. This implementation is only slightly different from the classic one. We change this line:
sum = sum + matrix_a[row][inCol] * matrix_b[inCol][outCol];

to this line:

matrix_r[row][outCol] = matrix_r[row][outCol] + matrix_a[row][inCol] * matrix_b[inCol][outCol];

This change is constructed of 2 changes:

1. We do not use “sum” as temporary variable for the summary accumulation.
2. We change the order of the summary operation. Instead of multiplying matrix A row by a matrix B column, we go over each element of A, and multiply by a whole row of B. Since the matrix is kept as a single line in the memory, we perform much less skips in the memory, therefore expect to have less Cache Misses.

To run the improved implementation:

1. Click File->Project Settings
2. Go to the “General” page
3. Click “-c” in the command line argument text box
4. Click OK.
5. Run the profile session again (click Profile -> Start Profile)

Now open the source code view again:

![Image of source code view]

The improved function, for the same line, we only have 97 cache misses.

Profile Mode

CodeXL profile mode is a powerful performance analysis tool that supports CPU and GPU profiling to provide program performance data. CodeXL profiling does
not require modifications to your source code or project. Profiling does not require recompilation, except for CPU profiling, which requires compilation with debugging enabled. Profiling lets you find performance hotspots and issues, determine the top data transfer and kernel execution operations, and identify problems such as failed API calls and resource leaks. You can use profiling to improve application performance through proper synchronization, bottleneck elimination, and load balancing.

CodeXL provides several modes of profiling. These modes let you assess program performance, use instruction-based sampling (IBS) or time-based sampling (TBS), or investigate branching, data access, instruction access, or L2 cache access. GPU profiling provides application timeline trace and performance counter modes.

The following is a quick introduction to CPU and GPU profiling. For further details, see the CodeXL Help information.

**CPU Profiling**

To profile a program:

1. Click on the profiling mode taskbar button.
2. Use the Profile drop-down menu to select the profiling mode.
   For example, for CPU performance profiling, select Profile >> CPU: Assess Performance.
3. Click the start button to launch the application for profiling.
4. To stop it, use the stop button any time during profiling. The bottom of the CodeXL window displays the elapsed clock time.

Profiling is available up to the time the application is closed. For the teapot example: click on the ‘x’ in the upper right corner of the teapot window.

After profiling is complete and data translation is over, a node in the left session tree is added for this session.
Overview Tab

The first page shown is the overview page. It shows the Modules and Functions tables and a brief description of the execution environment and profile detail. If multiple processes are profiled, then the Process table is shown. Each table shows the top five hot items.

Modules Tab

To open the Modules view:

1. Double click or use the “Open Modules” command from the context menu of the tree’s Modules node.
Call Graph Tab

To open the Call Graph view:

1. Double click the Call graph node in the Explorer tree, or use the “Open Call Graph” command from the context menu of the Call Graph node (available only if Call Stack Sampling was enabled).
Functions Tab

To open the Functions view:

1. Double-click the Functions node in the Explorer tree, or use the “Open Functions” command from the context menu of the Function node. The Functions list can be filtered based on the module to which they belong. To do this, invoke a dialog from the hyperlink at the top of function table that lists the displayed and hidden modules. The Functions list also can be filtered to display functions for a specific process using the Process drop list.
GPU Profiling

For GPU application timeline trace profiling:

1. Click on the profiling mode taskbar button.
2. Select Profile >> GPU: Application Timeline Trace from the Profile drop-down menu.
3. Run the program, then let it complete, or terminate it.
   An Application Timeline Trace view appears with a timeline of the program execution. This timeline shows the created OpenCL contexts and command queues, as well as the relationships between them.
   To select a subrange of the timeline, hold down <Ctrl>, and click and drag on a section of the timeline.
   To shift the timeline display left or right, simply click on it and drag.
   To zoom in/out, use the mouse wheel or the +/- keys. Selecting a small subrange lets you zoom in to see details about each event.
   For additional information, hover over an event; this displays a pop-up.

The following screenshot is an example of a COPY_BUFFER_TO_IMAGE data transfer event at 7752.980 ms on the timeline. The pop-up provides detailed timing data.
Summary Tab

The Summary tab provides several options for viewing profiling data: API, context, kernel, top 10 data transfer, top 10 kernel, warnings/errors.

The following screenshot shows an example of a Top 10 Kernel Summary.

The Warning(s)/Error(s) summary also includes a helpful list of best practice recommendations to improve program performance. The following example indicates issues with blocking write calls and small global work size.
Performance Counters View

The Performance Counters view in a GPU Performance Counters profile provides kernel performance details, including global work size and time. This mode collects performance counters from the GPU or APU for each kernel dispatched to the device. It also displays statistics from the shader compiler for each kernel dispatched. The performance counters and statistics can be used to discover kernel bottlenecks.

To display a Code viewer with kernel code:

1. Click on a kernel name (Method) in the Performance Counters view.

A pull-down bar at the top of the window under the Code Viewer tab (see following screenshot) lets you select OpenCL source (CL), intermediate language (IL), or instruction set architecture (ISA) code.
CodeXL Explorer Tree

The Explorer view lets you switch between profiling sessions. This view lists all profiling sessions for the current project.

To display a session's data, double-click on it. To rename or delete it, right-click on it.

To import profiling data, right-click or drag/drop a session data file to the Explorer.
Power Profiling

Power Profiling mode allows monitoring the power consumption of supported AMD APUs in real-time. Additional attributes can also be monitored: GPU and CPU frequencies, APU thermal trend and CPU Cores P-State.

For a list of currently supported APUs see the System Requirements section of the CodeXL Release Notes, available in the CodeXL installation folder and on the CodeXL web page.

Switching to Power Profiling mode

In the Profile Mode drop-down list, choose Power Profiling.

Starting a new Power Profiling session

After switching to Power Profiling mode, double click New Power Session... in the CodeXL Explorer tree.

Please note that if your hardware does not support Power Profiling, you will not be able to start the session.

After double clicking on New Power Session... in the CodeXL Explorer tree, an empty Power Profiling session window will be opened.
Clicking the Start button ▶️ will start the session with the current configuration. We will now see how to change a Power Profiling session’s configurations.

**Setting the Sampling Interval**

To configure the sampling interval (the time period which will pass between every two consecutive samples of the active counters), click on Profile -> Profile Settings -> Power Profile:

Please note that the current minimum sampling interval is 100 milliseconds. Click OK to apply your changes.

**Stopping a Power Profiling session**

To stop a running Power Profiling session, click on the Stop button ⏹️.

**Power Profiling Real-Time Values**

Throughout the Power Profiling session, the collected data will be streaming in real-time to the graphs. There are two tabs that present data: Timeline View and Summary View.
The Power Profiler Timeline View displays the measured values of the activated counters throughout the session. The horizontal axis of all charts represents the time that has passed since the beginning of the session. During profiling sessions, all of the charts in the Timeline View are being updated in real-time with the measured values which are streaming in. The uppermost ribbon (titled “Total APU Power”) displays the overall power consumption of the APU throughout the session.

The top chart has an adjustable range slider that controls the display of all the other timeline charts. By performing such actions as dragging the slider sideways, extending or retracting it, you set the scope of attention and the focus of the timeline charts. Each of the charts below displays only the data that was collected in the time range corresponding to the slider’s position and length. That is, the data in all timeline charts, except for the Total APU Power chart itself, is dictated by the time range which is selected by the Total APU Power chart’s range slider.

Below the Total APU Power ribbon, you will find additional ribbons containing more graphs, according to the set of activated counters: A Power chart which displays the power consumed by specific APU components (such as CPU cores or integrated GPU), a Frequency chart which displays the frequency of the selected components, a Temperature chart which displays the thermal trend of the selected components, and a CPU State chart which displays the CPU core states. The APU Power graph is always displayed, since the Total APU Power counter is activated by default and cannot be deactivated. The other charts (frequency, temperature and CPU core state) are optional, and will only be displayed if the relevant counters were activated.

To the right side of each chart you will find a legend that displays the measured values at a specific point in time. To change the point in time for which the values
are displayed, reposition the mouse cursor horizontally on one of the graphs. The list of counters in the legends is customizable, and specific counters can be removed/added between profile sessions.

**Power Profiling Summary View**

The Power Profiler Summary View displays an analysis of the values measured throughout the session. Similarly to the Timeline View, this view is updated in real-time when power profiling sessions are running.

At the upper-left side of the summary view, you can see the session duration which is the amount of time that the profiling session was in progress.

The following Power histograms graphs are plotted in the Summary View:

- **Total Energy Consumption**
  This histogram graph displays the cumulative energy consumed by the APU and discrete GPU components, measured in Joules.

- **Average Power Consumption**
  This graph displays the average power consumption of the APU and the discrete GPU components, measured in Watts.

If CPU or GPU frequency counters were activated for the session, you will find additional histogram graphs below the Power graph in the Summary View:

- **CPU Frequency Graph**
  This stacked histograms graph displays for each CPU core how much time it spent at each frequencies range.

- **GPU Frequency Graph**
  This graph displays how much time the GPU spent at each frequencies range.

**Configuring Power Profiler Sessions**

For further configuration options of the Power Profiler such as selecting which counters will be sampled and launching an application when the session begins, see the CodeXL User Guide by selecting from the menu bar Help -> View Help.
Analyze Mode

Static Kernel and Shader Analysis

In Analyze mode, you can compile and generate performance statistics for OpenCL kernels, DirectX Shaders and OpenGL shaders. The compilation and statistics generation processes can be targeted at a variety of AMD GPUs and APUs, regardless to the actual type of GPU/APU that is installed on your system. Using the Analyzer, you can generate and inspect ISA code, AMD IL code and D3D ASM code.

Switching to Analysis mode

Option 1:

Click on the Analyze Mode button in the CodeXL Mode toolbar.

Option 2:

Click Analyze in the main menu.
Once you switch to Analyze mode, you can create a new project, open a previously saved project or load the Teapot sample.

Creating a new project for Kernel Analysis

Click on the “Create New Project” link. The following CodeXL Project Settings dialog will appear:

Choose the executable file you want to work on. Now, in order to begin working, you simply need to add the .cl file you want to compile and analyze.

Note: The chosen executable has no part in Analyze mode. Choosing an executable file is required so that the other modes can be used too, but if you plan to use Analyze mode only, you don’t have to select an executable at all.

Adding OpenCL files to an existing project

Option 1:

Double click on the plus sign and add your file:
Option 2:

From the main toolbar, select **Add existing kernel/shader files to project**.

Option 3:

Right click on the project name, and select **Add kernel/shader file**.

Note: You can add as many OpenCL/DirectX files to a single project as you need. The OpenCL/DirectX files do not necessarily need to be relevant to the executable you chose for your project.

**Analyze Mode Options**

To open the Analyze Mode options tab in the CodeXL Options dialog, use the **Analyze Options** toolbar button.

© 2015 Advanced Micro Devices, Inc.
The Analyze tab of the CodeXL Options dialog appears:

**Select target devices**

The ASICs table contains a list of devices by series.

Use the checkboxes to select or unselect an entire series, or click the small triangle on the left to expand a tree node and expose specific families of target devices.

Note: AMD ASIC's are grouped now by generation name. e.g. “Graphics IP v7” which groups together devices of similar type and characteristics.

**Define build bitness**

On Windows, the build can be executed either via the 32-bit or the 64-bit compilation chains. This can be defined by the bitness combo, in the platform section of the toolbar.

*Build Options - defining kernel/shader compilation options*

In the Static Analyze toolbar, there is a text box where you can manually define specific OpenCL/HLSL build options (there is no support for GLSL build options):

*Note:* The display of this toolbar is dynamic; you can set it from the right-click menu in the main CodeXL frame:
The Build Options box is a place to set compiler build flags such as `-x clc++` or `-o3`. Any compiler build flag can be placed in this box.

**OpenCL Build Options Dialog**

This dialog will help you choose the correct OpenCL build options for you and hopefully will prevent you from making spelling mistakes while typing the options manually.

To open the **OpenCL Build Options** dialog, press the Button. You can browse between the ‘General & Optimization’ tab and the ‘Other’ tab to view all the available options. Once you choose an option, the option text will appear in the text box below marked as ‘OpenCL Build Command Line’. This string will also appear in the menu bar after you click the **OK** button.

Typing the command line in the text box will also mark the corresponding check boxes in the dialog.
Examples of using build options

For building the `tpAdvectFieldScalar.cl` kernel from CodeXL’s AMDTTeaPot sample project, enter the following options:

```-D GRID_NUM_CELLS_X=64 -D GRID_NUM_CELLS_Y=64 -D GRID_NUM_CELLS_Z=64
-D GRID_INV_SPACING=1.000000f -D GRID_SPACING=1.000000f -D
GRID_SHIFT_X=6 -D GRID_SHIFT_Y=6 -D GRID_SHIFT_Z=6 -D
GRID_STRIDE_Y=64 -D GRID_STRIDE_SHIFT_Y=6 -D GRID_STRIDE_Z=4096 -D
GRID_STRIDE_SHIFT_Z=12 -I path_to_example_src```

On windows, `path_to_example_src` should be:

`C:\Program Files (x86)\AMD\CodeXL\Examples\Teapot\res`

On Linux, `path_to_example_src` should be:

`/opt/AMD/CodeXL_X.X/examples/Teapot/AMDTTeaPotLib/`

Adding the option `-h` will dump the list of OpenCL compiler available options in the output tab.

HLSL Build Options Dialog

This dialog will help you choose the correct DirectX build options for you and hopefully will prevent making spelling mistakes while typing the options manually.

To open the dialog, press The Button. The dialog will be opened. Click the"HLSL Build Options” node to view the available options. Once you choose an option, the option text is displayed in the"HLSL Build Command Line” text box that appears below. This build option string will also appear in the toolbar’s build options box after you click the OK button.

As an alternative to selecting options through the radio buttons, it is possible to type a command in the “HLSL Build Command Line” text box. Build options types in the text box will automatically be translated to update of the relevant controls.
Getting Started with CodeXL

accordingly. For example, typing “D3DCOMPILE_DEBUG” in the lower text box automatically updates the “Debug” check box to be checked.

HLSL Compiler Tools
First select the build tool of choice for compiling the shader – D3d compiler / FXC compiler.

The CodeXL installation includes a copy of the Microsoft DirectX compiler DLL: d3dcompiler_47.dll. You may specify a different path if you want CodeXL to use a different d3dcompiler module. If you select the FXC compiler tool you must specify a path to the location of this tool.

To select the path of the compiler module, click the 'Browse…' option from the combo-box. When selecting Browse, a dialog box will open for selecting the compiler file.

- For D3D compiler – any file called d3compiler_* .dll can be selected
- For FXC compiler – only FXC.exe file can be selected.

Note: for D3D compiler the bundled file is selected by default.

Building a kernel/shader file

Build

The Build command builds the currently selected kernel/shader file for the designated target devices, which produces ISA code and AMD IL / D3D ASM code for each device, and displays statistics for each kernel. The compiler output (such as warnings and errors) is shown in the output tab.

To Build and Analyze an OpenCL file, do one of the following:

- Press CTRL + F7
- From the menu bar, click Analyze >> Build
- Right-click the designated file in the explorer tree, and select “Build”
Output Tab

The compiler output appears in the Output tab. The example below shows successful builds (no warnings or errors) for 4 devices.

If errors occur, the output will display the error and the line in which the error occurred:

![Output Example]

If there were errors, the output pane will display the error and the line where the error occurred:

![Output Error Example]

Double clicking on an error navigates the user to the Source Code view, displaying the kernel/shader source code:

![Source Code Example]

Statistics Tab

The Statistics tab gives detailed statistics for the selected kernel/shader for each target device. To open the Statistics tab, expand the desired kernel in the project tree, and double-click the Statistics node:
Note: the statistics table for pre-GCN devices (v4 & v5 generations) is a bit different. For more information, see the complete help manual document.

**Viewing compilation output: ISA and AMD IL**

To view the compilation output, double click the node of the desired ASIC in the explorer tree. This will open a tab containing the source code, the IL and the ISA. The program source code and the AMD IL code will be presented as standard text documents. The ISA will be presented in the “Enhanced ISA View” for GCN devices, and as a standard text document for pre-GCN devices.
The context menu enables to display/hide line numbers for each source code/IL/ISA tab.

Navigating through ISA code with the Enhanced ISA View

Using this view, you can inspect the ISA code of GCN devices and see the estimation for instruction cost in clock cycle. The view contains 5 columns:

- **Address**: the instruction’s offset within the program (in bytes)
- **Opcode**: the operation to be performed
- **Operands**: the data for the operation
- **Cycles**: the number of clock cycles which are required by a Compute Unit in order to process the instruction for a 64-thread Wavefront, while neglecting the system load and any other runtime-related factor.
- **Instruction Type**: the category of instructions to which the instruction belongs
- **Hex**: binary representation of the instruction, in hexadecimal format

Notes:
1. Note that code labels which appear in the Operands column are clickable. By clicking on a label link, you can navigate to the label’s spot in the code.

2. Note that this view is only available for GCN devices. For pre-GCN devices, the plain textual ISA view will be displayed.

<table>
<thead>
<tr>
<th>ISA Code</th>
<th>callPrice/yega</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01C2</td>
<td>V.CVIEF64.U32</td>
</tr>
<tr>
<td>0x01C2</td>
<td>V.NOV32</td>
</tr>
<tr>
<td>0x01C2</td>
<td>V.NOV32</td>
</tr>
<tr>
<td>0x01C4</td>
<td>V.BCPF64</td>
</tr>
<tr>
<td>0x01C6</td>
<td>V.CVIEF64.U32</td>
</tr>
<tr>
<td>0x00CC</td>
<td>V.CVF64.U32</td>
</tr>
<tr>
<td>0x00CC</td>
<td>V.CVF64.U32</td>
</tr>
<tr>
<td>0x01D4</td>
<td>V.CVF64.U32</td>
</tr>
<tr>
<td>0x01D7</td>
<td>V.CVF64.U32</td>
</tr>
<tr>
<td>0x01D7</td>
<td>V.CVF64.U32</td>
</tr>
<tr>
<td>0x01E4</td>
<td>S_AND_SAVEEXEC.B64</td>
</tr>
<tr>
<td>0x01F3</td>
<td>S_BRANCH.Exec</td>
</tr>
<tr>
<td>0x01F7</td>
<td>S_BRANCH.Exec</td>
</tr>
<tr>
<td>0x01F7</td>
<td>S_BRANCH.Exec</td>
</tr>
<tr>
<td>0x01F9</td>
<td>S_BRANCH.Exec</td>
</tr>
<tr>
<td>0x01F9</td>
<td>S_BRANCH.Exec</td>
</tr>
<tr>
<td>0x020A</td>
<td>S_BRANCH.Exec</td>
</tr>
<tr>
<td>0x020A</td>
<td>S_BRANCH.Exec</td>
</tr>
<tr>
<td>0x020B</td>
<td>S_BRANCH.Exec</td>
</tr>
<tr>
<td>0x020B</td>
<td>S_BRANCH.Exec</td>
</tr>
<tr>
<td>0x020C</td>
<td>S_BRANCH.Exec</td>
</tr>
<tr>
<td>0x020C</td>
<td>S_BRANCH.Exec</td>
</tr>
<tr>
<td>0x020C</td>
<td>S_BRANCH.Exec</td>
</tr>
<tr>
<td>0x020C</td>
<td>S_BRANCH.Exec</td>
</tr>
</tbody>
</table>

**Known Issues**

For a list of known CodeXL issues, review the release notes on the CodeXL web page and the AMD Developer Tools CodeXL forum:

community.amd.com/community/devgurus/codexl

**Support**

AMD general developer support page:

http://developer.amd.com/support/

Tools & SDKs section in AMD Developer Tools website:

Getting Started with CodeXL


AMD Accelerated Parallel Processing OpenCL Programming Guide:

For GPU development issues relating to other AMD tools, see the AMD GPU Developer Tools Forum:
community.amd.com/community/devgurus/gpu_developer_tools

To report a specific problem or request help with AMD CodeXL, visit the CodeXL Forum at:
community.amd.com/community/devgurus/codexl

DISCLAIMER
The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of non-infringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD’s Standard Terms and Conditions of Sale.

AMD, the AMD Arrow logo, AMD Radeon, AMD FirePro, gDEBugger, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Windows and DirectX are registered trademarks of Microsoft Corporation. PCIe and PCI Express are registered trademarks of PCI-SIG Corporation. OpenCL is a trademark of Apple Inc. Linux is a registered trademark of Linus Torvalds. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies. © 2015 Advanced Micro Devices, Inc. All rights reserved.