Exercise Manual

for

Introduction to OpenCL™ for Intel® FPGAs

Lab Exercise 1

<u>Software Requirements</u>

CentOS 7 Linux* OS
Eclipse IDE
Intel® FPGA SDK for OpenCL™ version 19.1
Intel® Quartus® Prime Pro software version 19.1 with Arria® 10 family
Intel® Code Builder for OpenCL™ (Needed to run fast emulator)

Exercise 1

Setting Up OpenCL™ Host-Side Application

In this exercise, you will practice writing an OpenCL™ host-side application including constructs to get the OpenCL platform and device; create an OpenCL context, command queue, and buffers. We will use the C++ OpenCL API in this lab. C API usage is very similar, visit the Intel FPGA OpenCL web page for examples.

____ 1. Unzip Introduction_to_OpenCL_19_1_v1.tar.gz

a. Type “tar -xvf Introduction_to_OpenCL_19_1_v1.tar.gz”

This will unzip the course files into the appropriate folder. If the tar.gz files is not there with exactly the same name, please consult the instructor.

____ 2. Go into the course directory

a. Type “cd OCL_19_1”

____ 3. Source the course environment

a. Examine init.sh using your favorite text editor (For example gedit)

i. Type “gedit opencl_init.sh”

This script set up the environment.

b. Execute the script

i. Type “source opencl_init.sh” in the terminal

Step 2. Launch Eclipse

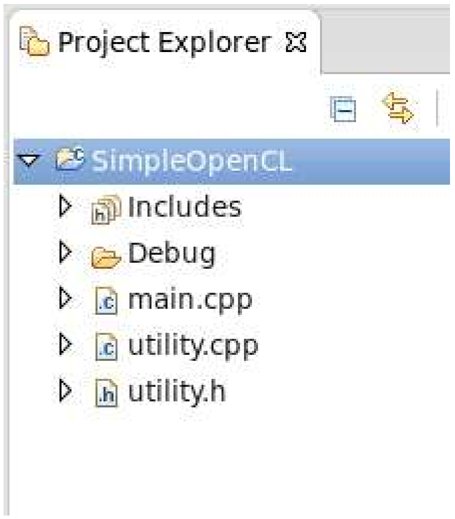
____ 1. Launch Eclipse

a. Type “eclipse &” in the terminal

____ 2. Set the workspace to /home/student/fpga_trn/OpenCL/OCL_19_1 and click Launch

____ 3. Close the welcome screen if it appears by clicking the X by the tab that says Welcome

____ 4. You will see a project named **SimpleOpenCL** open in the **Project Explorer** pane on the left. Expand this project by clicking the triangle next to the project’s name.



*The **SimpleOpenCL** project is the project we’re going to use to compile and run our OpenCL host program. You should notice that there are three source files. **utility.cpp** contains functions that perform useful tasks such as printing information about the OpenCL environment. **main.cpp** has main() that will contain all of our host side code, which is what we’re going to modify in this exercise. **SimpleKernel.cl** is the kernel file, but is not included in the project since it must be compiled separately in the terminal window.*

____ 5. Examine the project settings

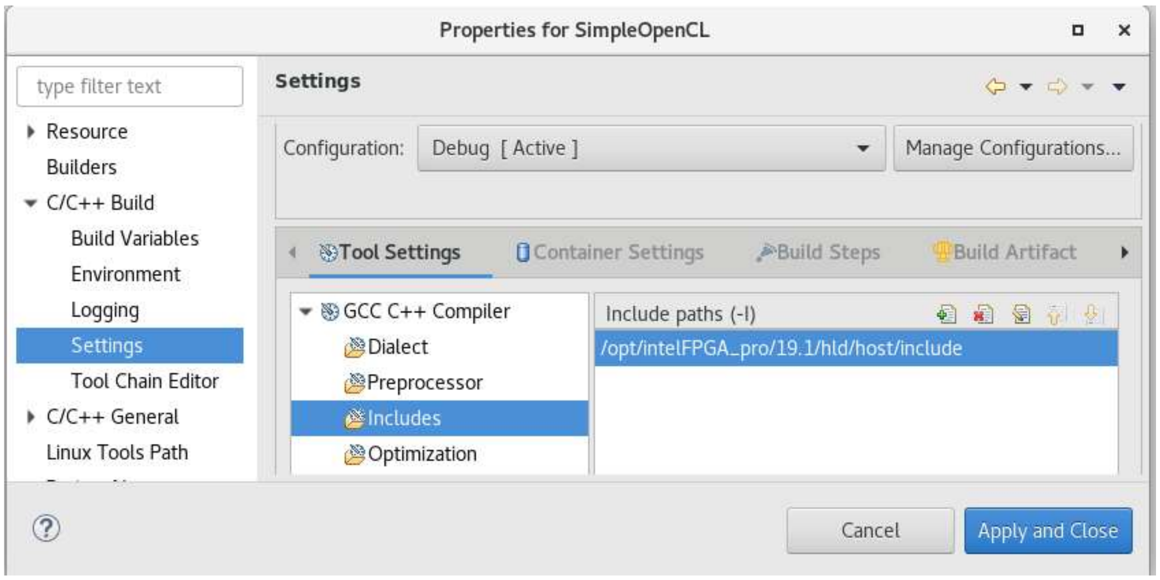
When being developed, this project was started by right-clicking in the Project Navigator pane, and selecting New -> C++ Project. Then, customizing the settings outlined below. Any version of Eclipse meant for C/C++ development should be similar to set up.

a. Click the **Project** menu item near the top of Eclipse, then select **Properties** at the bottom of the list

b. Expand **C/C++ Build** at the left and click **Settings**

c. In the tab that says Settings, ensure the configuration says **Debug [Active]**, and that the **Tools Settings** tab is selected. Expand **GCC C++ Compiler** in the left pane, and click **Includes**.

d. Observe the compile include path that has been included.



Configuration: Debug [Active]

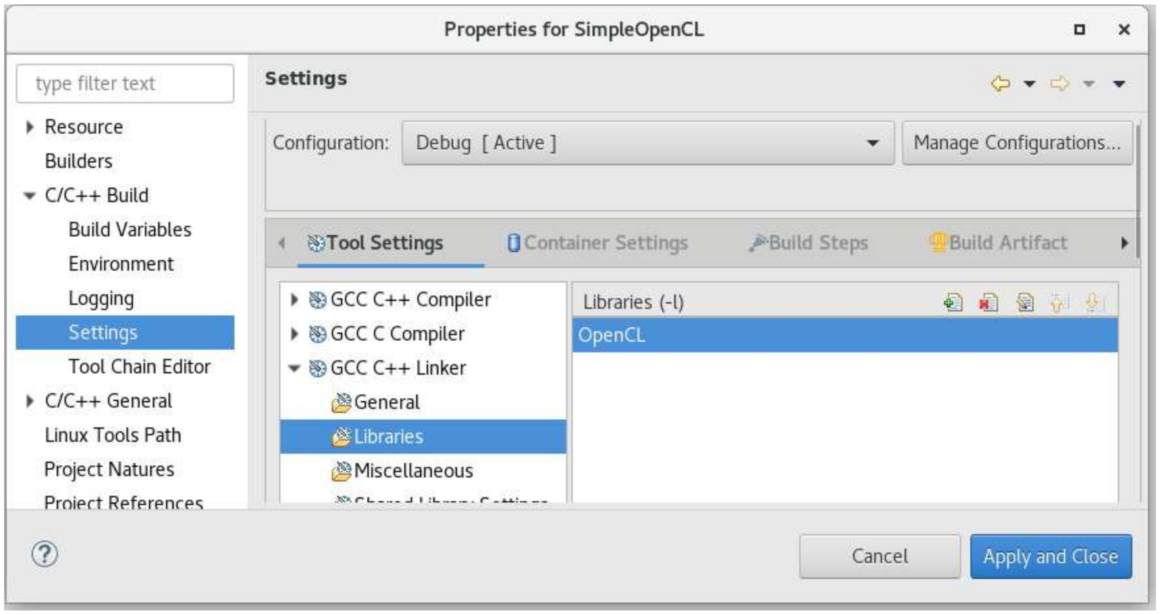
Include paths (-I)
/opt/intelFPGA_pro/19.1/hld/host/include

____ 6. In the terminal window, type "aocl compile-config" (may need to press return first)

```
[student@localhost OCL_19_1]$ aocl compile-config
-I/opt/intelFPGA_pro/19.1/hld/host/include
```

This command gives you the configuration you need to use for the compiler settings. Observe that it is the same path set up in Eclipse for the project.

____ 7. In the **Tools Settings** tab in Eclipse, expand **GCC C++ Linker** and select **Libraries**. Notice the libraries that have been set up for the linker.



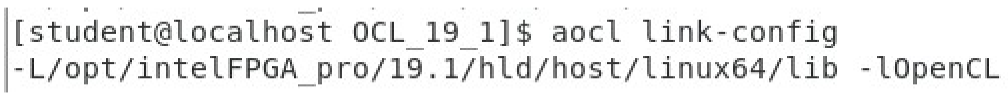
Settings

Configuration: Debug [Active]

Libraries (-l)
OpenCL

____ 8. Next, click on **Miscellaneous** (under **Libraries**), and observe the Linker flags (which are command line options) that have been set up.

____ 9. In the terminal window, type “aocl link-config”



```
[student@localhost OCL_19_1]$ aocl link-config
-L/opt/intelFPGA_pro/19.1/hld/host/linux64/lib -lOpenCL
```

This command gives you the configuration you need to use for the linker settings. Observe that they are the same things set up in the Eclipse project.

____ 10. For more information on how installable client driver and the various different OpenCL implementations are installed. Type “aocl diagnose icd-only”

For our training, we’ll be using the Fast Emulator implementation implemented with libintelocl_emu.so, the regular FPGA platform is implemented with libalteracl.so

____ 11. Click **Cancel** at the bottom of the Property window to close the window.

Step 2. Write the host side code

If you prefer to edit your code in another editor, it is fine. Simply navigate to the SimpleOpenCL/ subdirectory and open the files from there.

____ 1. Open **main.cpp** from the **Project Explorer** pane in Eclipse by double-clicking on its name.

____ 2. Look for the comment “Exercise 1 Step 2.3” in **main.cpp**.

____ 3. Right below the comment, complete the line of code that will fill in the values for the vector **PlatformList.** Set the value returned to the variable **err**.

Remember this lab file is designed to use the OpenCL C++ API.

Since we linked to the installable client driver, there should be 3 OpenCL Platforms, the FPGA Platform, the FPGA Fast Emulation Platform, and the CPU Platform

____ 4. Look for the comment “Exercise 1 Step 2.5” in **main.cpp**

____ 5. Write the code that will fill in the values for the vector **DeviceList** below the comment “Exercise 1, Step 2.5”

*Hint: Use the Platform **PlatformList[currnet_platform_id]** to use the fast emulation platform. Use the device type **CL_DEVICE_TYPE_ALL**.*

Since we’re going to execute this host program in emulation mode for which we’ve set the number of devices to 1, there will only be 1 device.

____ 6. Create a cl::Context object called **mycontext** below the comment “Exercise 1, Step 2.6.”. Pass the DeviceList into the constructor

Everything we wrote up until this point is considered setup code using OpenCL platform layer APIs. This is code that only needs to be written once and can be used in many application scenarios. In your own code, you would likely store the setup code in a function so it can be easily reused.

____ 7. Create a command queue named **myqueue** below the comment “Exercise 1, Step 2.7.”. Use only the first (0th) device in the DeviceLIst

____ 8. Below the comment “Exercise 1, Step 2.8”, create three buffers named **Buffer_In**, **Buffer_In2,** and **Buffer_Out**. The first two buffers are READ_ONLY and kernelOut is WRITE_ONLY. The size of the buffer for all three should be sizeof(cl_float)*vectorSize which is the total size of the array in bytes.

These are used to represent memory on the device and will be used as arguments to the kernel later.

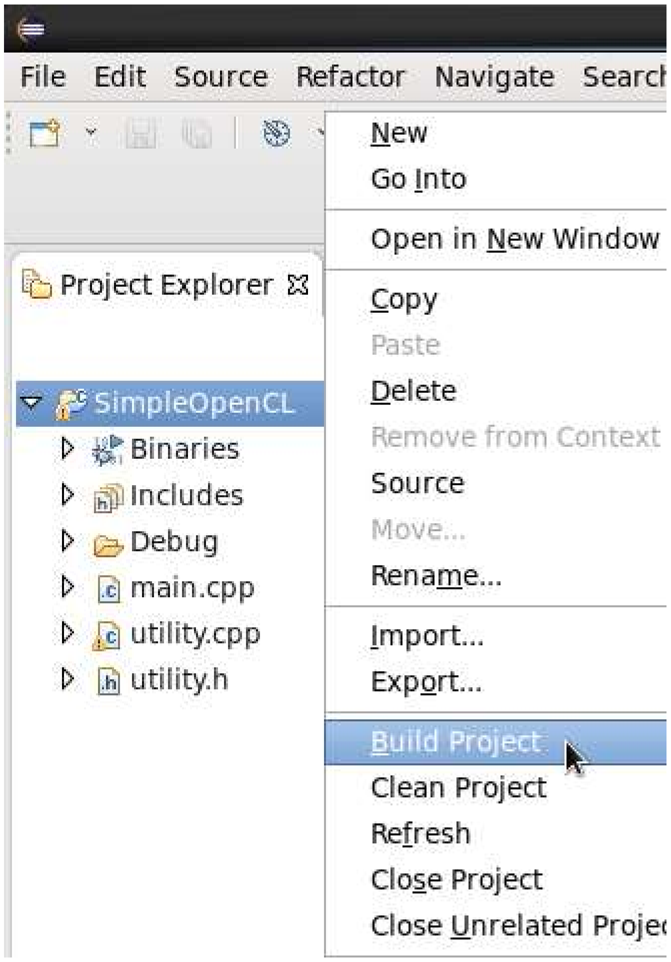
____ 9. Below the comment “Exercise 1, Step 2.9”, write two lines that would transfer the contents of **X** and **Y** into the buffers **Buffer_In** and **Buffer_In2** on the device respectively. Remember that in C++, an array’s name is a pointer.

*X, Y, and Z are arrays of floating point numbers.. The function **fill_generate** randomly generates numbers between the values of **LO** and **HI** to fill the array.*

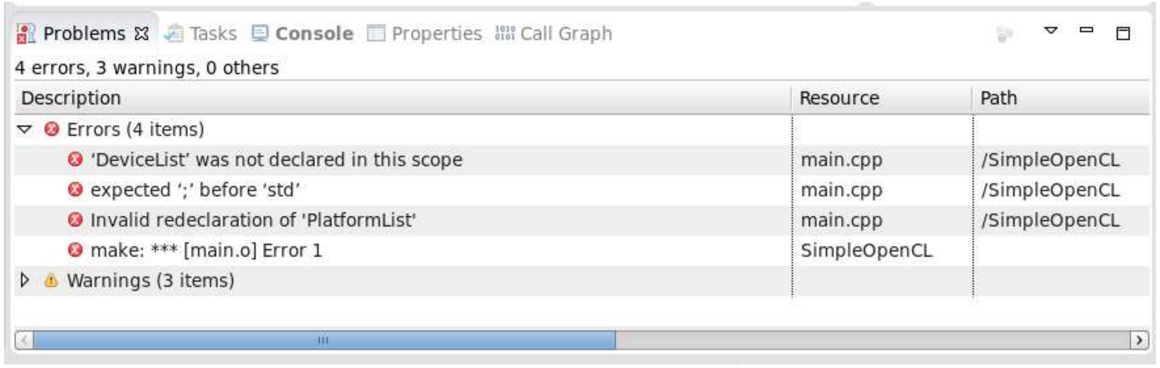
Step 3. Run and debug the host-side program

____ 1. Save **main.cpp**.

____ 2. Compile the project by right-clicking the project's name in the Project Navigator pane and selecting Build Project. For subsequent compiles, before selecting Build Project, first run Clean Project (the next option down on the right-click menu).



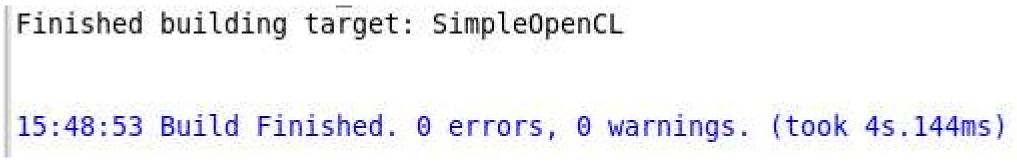
____ 3. If you have any errors, they will show up in the **Problems** tab at the lower part of Eclipse. Expand **Errors** and scroll right to see which file and line number.



4 errors, 3 warnings, 0 others

Description	Resource	Path
Errors (4 items)		
'DeviceList' was not declared in this scope	main.cpp	/SimpleOpenCL
expected ';' before 'std'	main.cpp	/SimpleOpenCL
Invalid redeclaration of 'PlatformList'	main.cpp	/SimpleOpenCL
make: *** [main.o] Error 1	SimpleOpenCL	
Warnings (3 items)		

____ 4. Correct any errors with your compile by modifying main.cpp. Remember to run Clean Project before Build Project for any subsequent compiles after the first one. When you see "Build Finished" in blue in the **Console** tab at the bottom of Eclipse, your compile was successful.



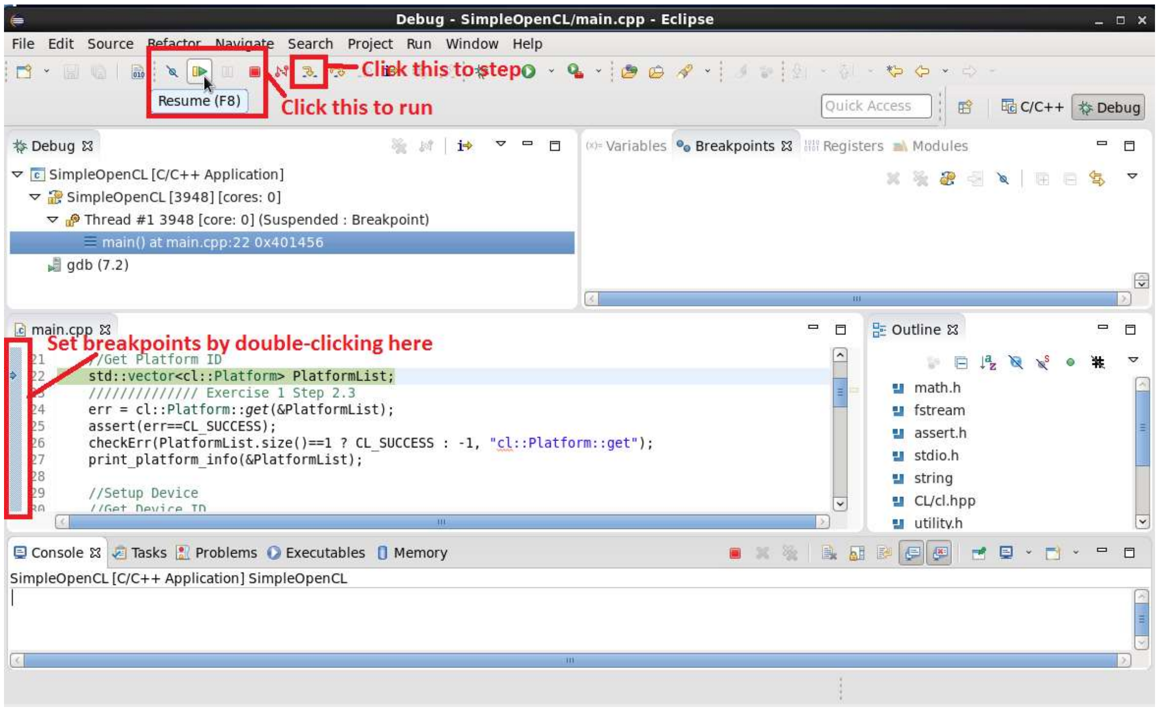
```
Finished building target: SimpleOpenCL

15:48:53 Build Finished. 0 errors, 0 warnings. (took 4s.144ms)
```

____ 5. Debug the program. Right-click on the project's name in the **Project Navigator** pane and select Debug As -> Local C/C++ Application.

____ 6. Click **Switch** when asked to Confirm the Perspective Switch.

____ 7. You are now in the **Debug** view of Eclipse. Click the green arrow with the yellow vertical bar beside it near the top to run. The program will run until it finishes or a breakpoint is reached. Set breakpoints by double-clicking in the area to the left of the code. Step by clicking the button that has a yellow arrow bending down.

The output of the host program will show up in the Console tab at the bottom of the screen.



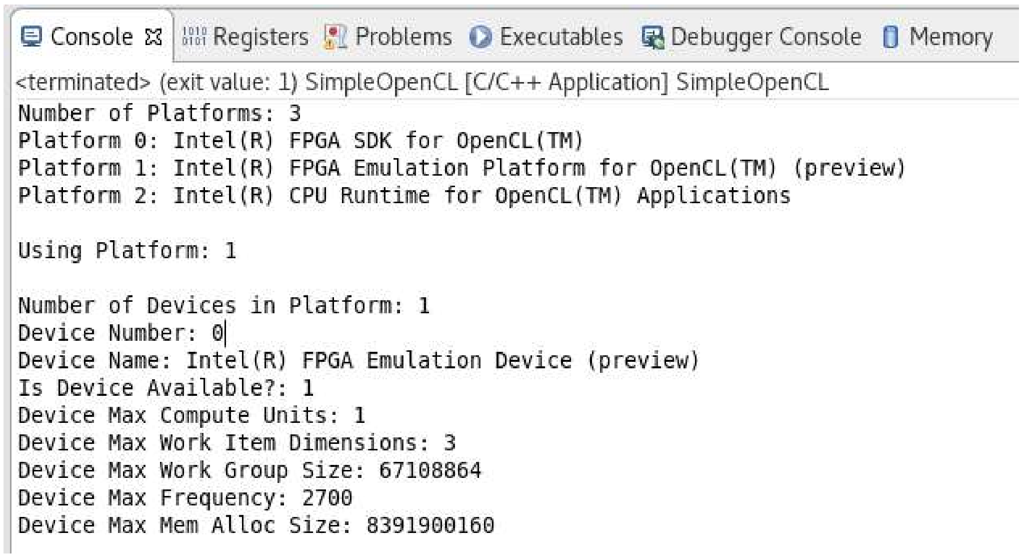
```
SimpleOpenCL [C/C++ Application]
  SimpleOpenCL [3948] [cores: 0]
    Thread #1 3948 [core: 0] (Suspended : Breakpoint)
      main() at main.cpp:22 0x401456
  gdb (7.2)
```

```
    //Get Platform ID
    std::vector<cl::Platform> PlatformList;
    ////////////// Exercise 1 Step 2.3
    err = cl::Platform::get(&PlatformList);
    assert(err==CL_SUCCESS);
    checkErr(PlatformList.size()==1 ? CL_SUCCESS : -1, "cl::Platform::get");
    print_platform_info(&PlatformList);

    //Setup Device
    //Get Device ID
```

```
SimpleOpenCL [C/C++ Application] SimpleOpenCL
```

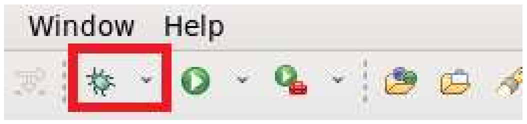
____ 8. Run the program by clicking the green arrow with the yellow vertical bar to its left. When the program runs toward the end, you should see information regarding the Platforms and Device printed in the **Console** tab

Here you see the various properties of your OpenCL Platforms and device.

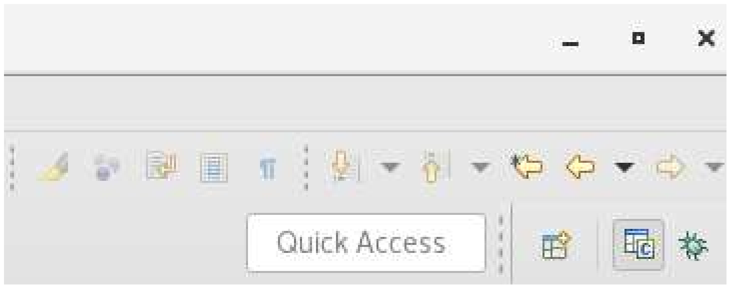


```
<terminated> (exit value: 1) SimpleOpenCL [C/C++ Application] SimpleOpenCL
Number of Platforms: 3
Platform 0: Intel(R) FPGA SDK for OpenCL(TM)
Platform 1: Intel(R) FPGA Emulation Platform for OpenCL(TM) (preview)
Platform 2: Intel(R) CPU Runtime for OpenCL(TM) Applications

Using Platform: 1

Number of Devices in Platform: 1
Device Number: 0
Device Name: Intel(R) FPGA Emulation Device (preview)
Is Device Available?: 1
Device Max Compute Units: 1
Device Max Work Item Dimensions: 3
Device Max Work Group Size: 67108864
Device Max Frequency: 2700
Device Max Mem Alloc Size: 8391900160
```

____ 9. If you need to restart your debugging session because it terminates, press the bug icon near the **Window** menu item at the top of the screen.



____ 10. When you have successfully run your code, switch back to the C/C++ perspective by clicking the **C/C++** button near the top of the Eclipse window.



Exercise Summary

- Practiced writing OpenCL host-side code that moved content into the device memory.

END OF EXERCISE 1
