# OpenCL: Utilizing DSP Accelerator from the ARM Processor

## I. Purpose

The goals of this lab are as follows:

* Demonstrate how to utilize DSP accelerators using a standard Linux programming.
* Demonstrate how OpenCL distributes an algorithm between multiple DSP cores.

## II. Hardware and software requirements

Ubuntu Laptop with the following attributes:

* Processor SDK release
* Linaro cross compiler
* DHCP server
* NFS server
* SCP server (sudo apt-get install ssh)
* Picocom or other terminal program
* Pre-configured SDcard with boot partition and file system partition

## III. Lab Setup

Each student station has an AM57X EVM (X15) connected to an Ubuntu laptop. In addition, each EVM is connected using an FTDI cable to a laptop running a terminal emulator such as Tera Term (for Windows) or Picocom (for Linux machines).

SD boot card with all Processor SDK kernel files and complete file system is inserted into the EVM, as shown.



## IV. Lab Sequence

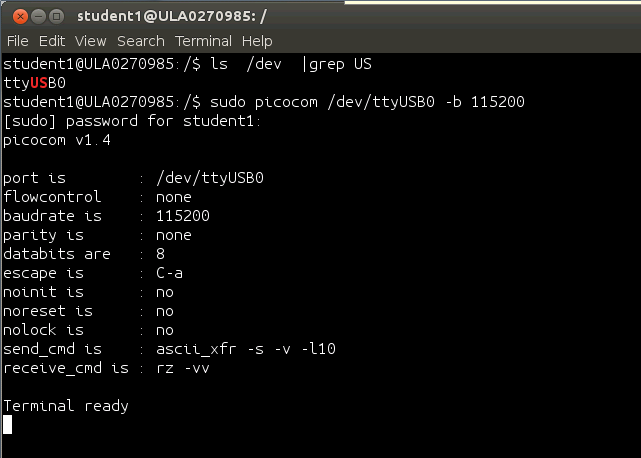
1. Run the pre-built openCL examples in the release.
2. Create a new example program on the Ubuntu server and build it using cross compiler tools.
3. SCP the new program to the EVM.
4. Run the new program on the EVM and observe the results.
5. Change some of the parameters of the new example program, then build, copy and run the program again. Observe the new results.

## Task 1.1: Run the Pre-built openCl Examples in the Release

**NOTE: You can connect to the EVM using either a Windows or Ubuntu laptop. The Windows laptop connection to the EVM uses Tera Term. The Ubuntu connection to the EVM uses Picocom. Depending on your settings, you have to choose ONLY one of the following: Tera Term on Windows or Picocom on Ubuntu.**

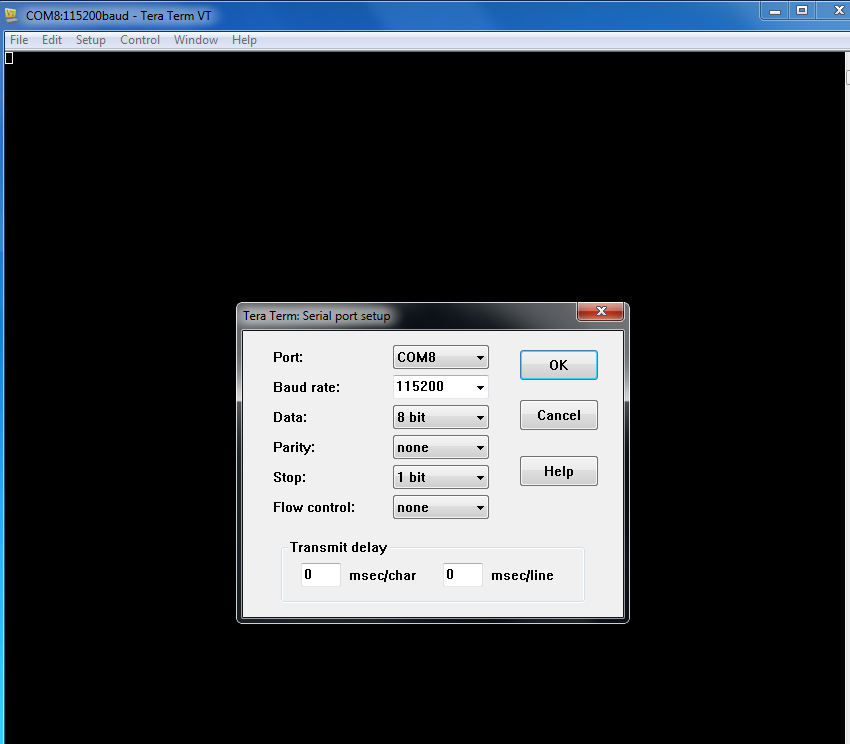
### 1.1A: Using Picocom on Ubuntu Laptop

1. Connect the FTDI cable to a USB port on the laptop.
2. If this is an Ubuntu laptop, use follow the following steps
   1. To discover devices to which the board is connected, use the command **ls /dev | grep US**
   2. The above command displays a list of devices connected to the USB. If you repeat the instruction before and after connecting the FTDI-USB cable, you will see to which port it is connected. Assume that the port is **ttyUSB0**.
   3. Start the Picocom terminal. Enter the command **sudo picocom /dev/ttyUSB0 –b 115200**. See the screen shot below.



### 1.1B: Using Tera Term on Windows Laptop

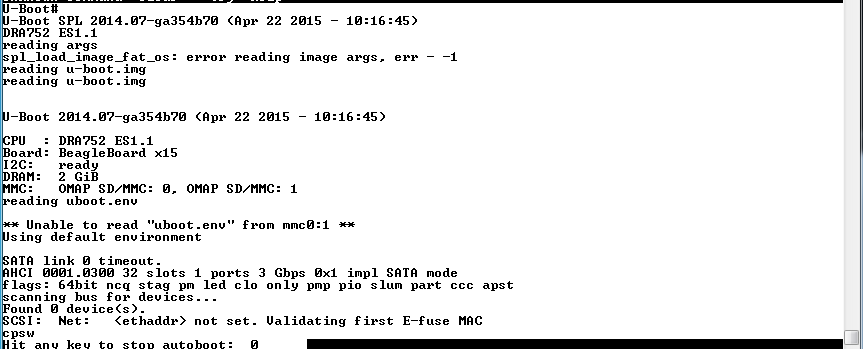
1. Connect the FTDI cable to a USB port on the laptop.
2. For Tera Term running on Windows, do the following:
   1. Start the Tera Term session.
   2. Configure the serial setup as shown (Choose the correct COM number. If more than one COM number is available for the serial port, try all of them.).



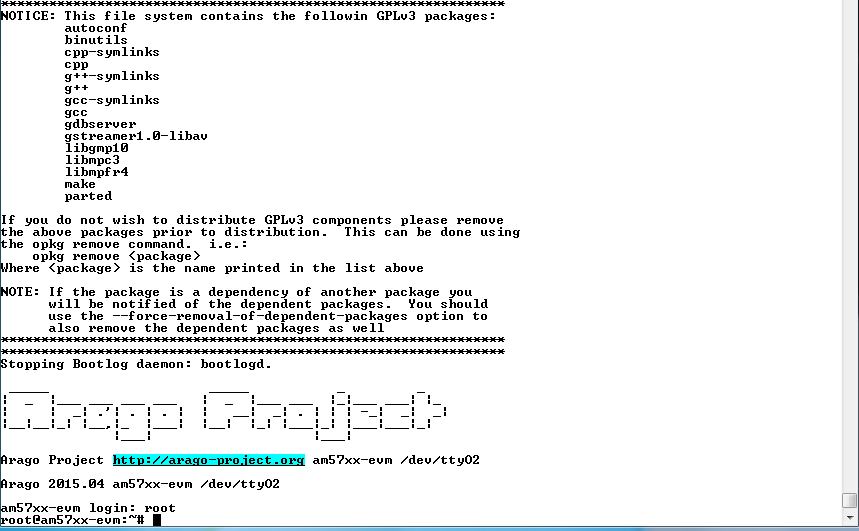
* 1. Click *OK.*

### 1.2 Prepare the Device

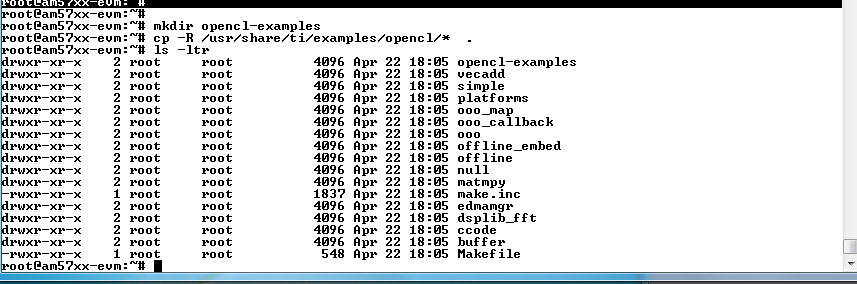
1. Make sure that the SD card is correctly inserted.
2. Power up the EVM
3. Push the blue button next to the power jack to start U-BOOT. The console shows the boot process:



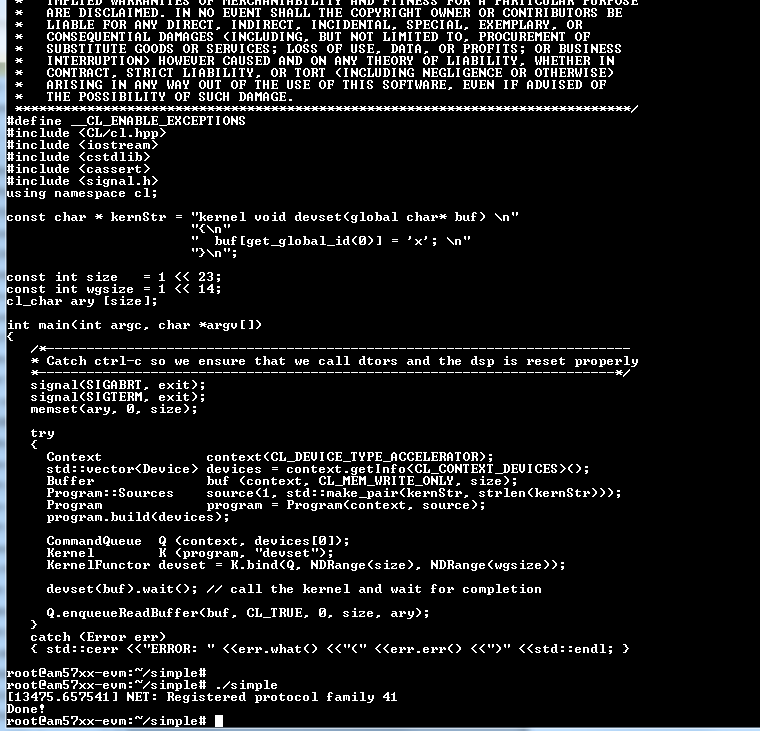
1. It takes few seconds for the kernel to start. When the login prompt appears, login as ***root***.



1. Build a new directory and use the following commands to move all openCL examples into the new directory:
   1. OpenCL examples are part of the release in the directory: **/usr/share/ti/examples/opencl/**
   2. Make a new directory: **mkdir opencl-examples**
   3. Copy the examples: **cp -R /usr/share/ti/examples/opencl/\* opencl-examples**

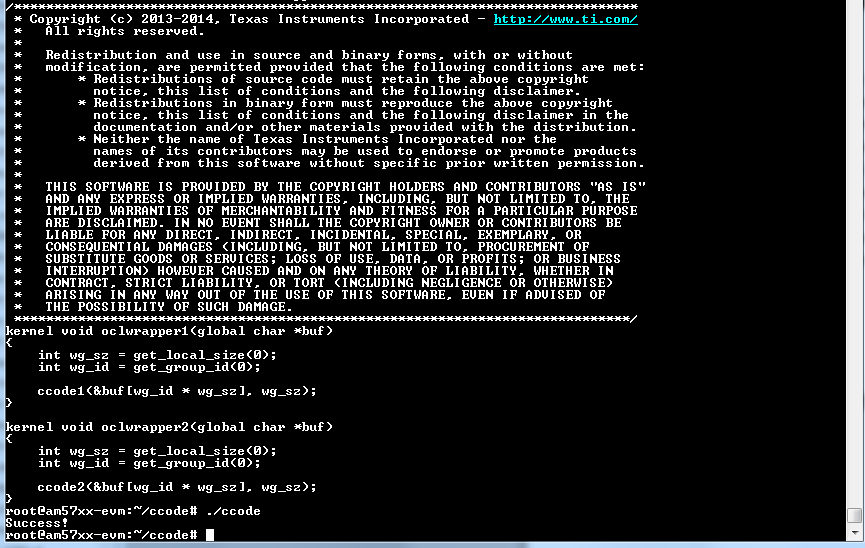


### 1.3 Run the Simple Example: Pure C++ Code

1. Go to the **/simple** directory and observe the files: ***cd opencl-examples/simple***
2. Notice that there is only one source file: **simple.cpp**. Look at the file and see how the kernel is defined. Notice that even though the kernel operation runs on the accelerators (DSP), no DSP code needs to be developed.
3. Run the example **./simple**
4. Observe the printout (Done!), as shown

### The ccode Example: Using C File to Build Kernel

1. Go to ccode directory and observe the files there: **cd ../ccode**
2. Notice that there are three source files: **main.cpp**, **ocwrapper.cl** and **ccode.c**. Look at the **ccode.c** file and see how the kernel is defined. Look at the oclwrapper.cl and see how the kernel is called. Last look at main.cpp and see how the kernel is built. Developing c code enables DSP engineer to optimize the code using DSP intrinsic and pragmas.
3. Run the example **./ccode**
4. Observe the printout (Success!).
5. On the screen shot shown, the top portion is part of **oclwrapper.cl**, and the bottom portion is the **ccode** run.



## Task 2: Build and Run the Example Code (Embedded Accelerator Case)

At the time of developing this lab, there is no native openCL compiler on the target. Thus, the cross compiler is used in this lab. On the Ubuntu server, a version of processor SDK is installed.

### 2.1: Build the OpenCL Examples using Cross Compiler on the *Ubuntu Laptop*

**Note – Base address of the opencl Lab is /home/sitara/opencl**

**User name is sitara, the password is sitara.**

**Note: The processor SDK directory name may be different (depends on the release) so in this document we will refer to it as ti-processor-sdk.**

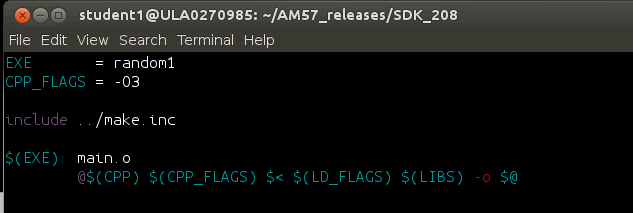
1. Go to student account ***cd /home/sitara/opencl*** or ***cd ~/opencl***
2. Use the command **ls –ltr** to display the Processor SDK directory
3. Go to the SDK directory (**cd ti-processor-sdk**) and look at the file **Rules.make**
   1. Make sure that the **DEFCONFIG** is set to **tisdk\_am57xx-evm\_defconfig**
   2. Set the **DESTDIR** to **/opt/filesys** directory
   3. For vi users use ***sudo vi Rules.make***
   4. The following table describes these parameters:

|  |  |  |
| --- | --- | --- |
| **Item to Verify/Modify** | **Description** | **Settings** |
| **DEFCONFIG** | Describes the actual architecture for which the applications are built. The directory**board-support/linux-3.14.35-gb60f54e/arch/arm/configs/**has a list ofsupported architectures. | For AM57X **tisdk\_am57xx-evm\_defconfig**  For Keystone: **keystone\_defconfig** |
| **DESTDIR** | Tells the make utility where to copy the build results. If NFS is used, then put the result binaries in a directory of the file system that is mounted onto your EVM. | **/opt/filesys** |

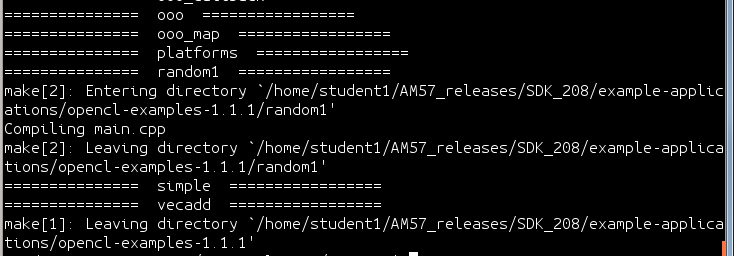
1. Build the OpenCL examples of the release:
   1. Clean previous builds: ***sudo make clean***
   2. Build the new release: ***sudo make******opencl-examples***
2. Verify that the examples were built:
   1. Use the command ***ls example-applications/opencl-examples-1.1.1/simple*** and verify that there are four files:
      * simple.cpp
      * Makefile
      * simple.o
      * simple
   2. Use the command ***ls example-applications/opencl-examples-1.1.1/ccode*** and verify that there are seven files:
      * Makefile
      * main.cpp
      * main.o
      * ccode.c
      * ccode.obj
      * oclwrapper.cl
      * ccode

## 2.2: Create a New Example and Build It

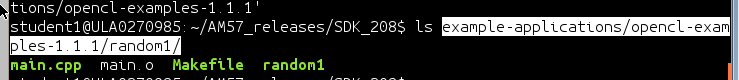
1. Go to the examples directory: ***cd example-applications/opencl-examples-1.1.1***
2. Make a new directory: ***sudo mkdir random1***
3. Move to the new directory: ***cd random1***
4. Copy source file **main.cpp** from the project directory: ***sudo cp ~/opencl/projects/random1/main.cpp .***
5. Copy the **Makefile** from the **/simple** directory: ***sudo cp ../simple/Makefile .***
6. Find the kernel definition in **main.cpp**. The kernel writes random numbers between uniformly distributed between 0 and 1000, and adds the core number multiplied by 10000. Thus, the value of the output tells us if it was generated by Core 0 or Core 1.
7. Edit the Makefile that you just copied:
   1. Enable write (modify) of all files: ***sudo chmod 777 \****
   2. Use an editor (the following instructions are for vi): ***sudo vi Makefile***
      1. Change the EXE (executable) name from **simple** to **random1**
      2. Change the EXE dependency to **main.o**
   3. A screen shot of the Makefile is shown:

**

1. Return back to the SDK directory: **cd ../../../**
2. Build the examples again: ***sudo make opencl-examples*** Notice that **random1** is built.



1. Use the command **ls example-applications/opencl-examples-1.1.1/random1**and verify that **random1** was built, as shown.



### 2.3: Copy Random1 to the EVM and Run it

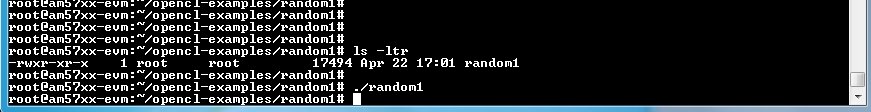
1. Next the random1 executable that was just built need to be copied to the EVM. This will be done following these steps:
   1. On the EVM make a new directory random1

***cd ~/opencl-examples***

***mkdir random1***

***cd ~***

1. Push the file **random1** from the *Ubuntu laptop* to the EVM using Secure Copy (scp):
   1. Find the IP address of the EVM do ***ifconfig***
   2. Record the value of ipaddress of evm to be used in the next step
   3. **scp random1 root@<ipaddress of evm>: .**
   4. If scp asks you to confirm, write **yes**
   5. This will copy random1 to the /home/root/ directory on the EVM
   6. Move random1 from the home directory of the EVM to the random1 directory
   7. On the EVM do ***mv ~/random1 ~/ opencl-examples/random1/.***
2. ***cd ~/opencl-examples/random1***
3. Use the command ***ls –ltr*** and verify that **random1** executable was copied.
4. Run the code*:* ***./random1***
5. The output will appear on the display. Notice that different cores generated different numbers at the middle of the output sequence.

****

## Task 3: Build and Run Example Code (C Code Accelerator)

At the time of developing this lab, there is no native openCL compiler on the target. Thus, the cross compiler is used in this lab. On the Ubuntu server, a version of processor SDK is installed.

### 2.1: Build the OpenCL Examples using Cross Compiler on the *Ubuntu Laptop*

NOTE: If you successfully built the OpenCL examples in the previous task, skip to the Step 2.2

**Note – Base address of the opencl Lab is /home/sitara/opencl**

**User name is sitara, the password is sitara.**

**Note: The processor SDK directory name may be different (depends on the release) so in this document we will refer to it as ti-processor-sdk.**

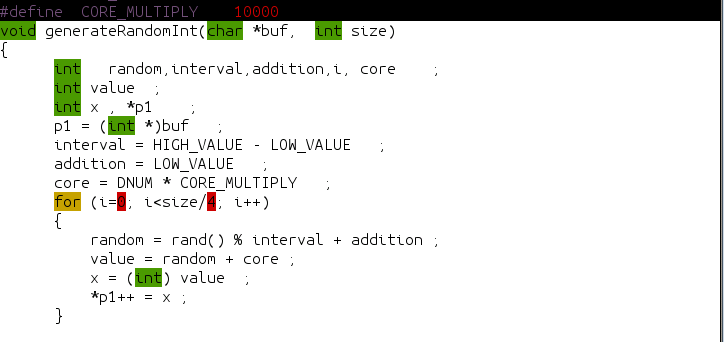
1. Go to student account ***cd /home/sitara/opencl*** or ***cd ~/opencl***
2. Use the command **ls –ltr** to display the Processor SDK directory
3. Go to the SDK directory (**cd ti-processor-sdk**) and look at the file **Rules.make**
   1. Make sure that the **DEFCONFIG** is set to **tisdk\_am57xx-evm\_defconfig**
   2. Set the **DESTDIR** to **/opt/filesys** directory
   3. For vi users use ***sudo vi Rules.make***
   4. The following table describes these parameters:

|  |  |  |
| --- | --- | --- |
| **Item to Verify/Modify** | **Description** | **Settings** |
| **DEFCONFIG** | Describes the actual architecture for which the applications are built. The directory**board-support/linux-3.14.35-gb60f54e/arch/arm/configs/**has a list ofsupported architectures. | For AM57X **tisdk\_am57xx-evm\_defconfig**  For Keystone: **keystone\_defconfig** |
| **DESTDIR** | Tells the make utility where to copy the build results. If NFS is used, then put the result binaries in a directory of the file system that is mounted onto your EVM. | **/opt/filesys** |

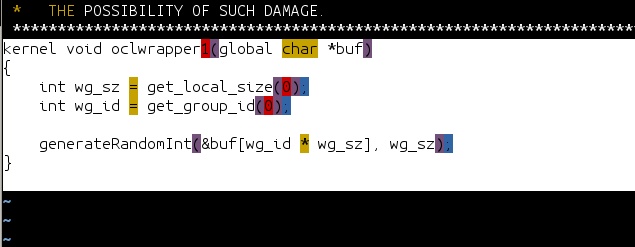
1. Build the OpenCL examples of the release:
   1. Clean previous builds: ***sudo make clean***
   2. Build the new release: ***sudo make******opencl-examples***
2. Verify that the examples were built:
   1. Use the command ***ls example-applications/opencl-examples-1.1.1/simple*** and verify that there are four files:
      * simple.cpp
      * Makefile
      * simple.o
      * simple
   2. Use the command ***ls example-applications/opencl-examples-1.1.1/ccode*** and verify that there are seven files:
      * Makefile
      * main.cpp
      * main.o
      * ccode.c
      * ccode.obj
      * oclwrapper.cl
      * ccode

### 3.2: Create a New Example and Build It

1. On the Ubuntu server, go to the examples directory*:* ***cd example-applications/opencl-examples-1.1.1***
2. Make a new directory: ***sudo mkdir random2***
3. Move to the new directory: ***cd random2***
4. Copy source file **main.cpp** from the project directory: ***sudo cp ~/projects/random2/main.cpp .***
5. Copy source file **ccode.c** from the project directory: ***sudo cp ~/projects/random2/ccode.c .***
6. Copy the **Makefile** from the **/ccode** directory: ***sudo cp ../ccode/Makefile .***
7. Copy the **oclwrapper.cl** from the **/ccode** directory: ***sudo cp ../ccode/oclwrapper.cl .***
8. Find the kernel definition in **ccode.c**. The kernel writes random numbers between uniformly distributed between 0 and 1000, and adds the core number multiplied by 10000. Thus, the value of the output tells us if it was generated by Core 0 or Core 1.



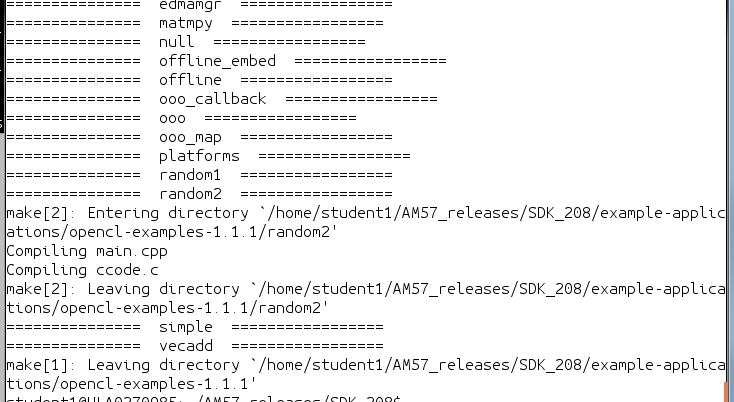
1. Enable write (modify) of all files: ***sudo chmod 777 \****
2. Look at the **oclwrapper.cl** file. It calls two functions: **ccode1** and **ccode2**. Each function has two parameters: 1) A pointer to the buffer and 2) The number of elements.
3. Use an editor to modify **oclwrapper.cl** (these instructions are for vi): ***sudo vi oclwrapper.cl***
   1. Change **oclwrapper1** to call **generateRandomInt** instead of **ccode1**
   2. Delete **oclwrapper2**
   3. A screen shot of **oclwrapper1.cl** is shown. Notice that when oclwrapper1 is called, it is called with a parameter to the buf-based address.



1. Edit the Makefile that you just copied:
2. Open the editor (these instructions are for vi): ***sudo vi Makefile***
3. Change the EXE (executable) name from **ccode** to **random2**
4. A screen shot of the **Makefile** is shown:



1. Return back to the SDK directory: **cd ../../../**
2. Build the examples again: ***sudo make opencl-examples*** Notice that **random2** is built.



1. Use the command ***ls example-applications/opencl-examples-1.1.1/random2*** and verify that **random2** was built, as shown.



### 2.3: Copy Random1 to the EVM and Run it

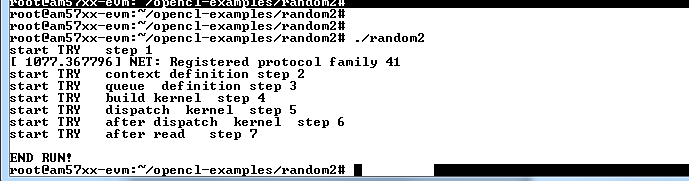
1. Next the executables that were just built and the oclwrapper.cl file need to be copied to the EVM. This will be done following these steps:
   1. On the EVM make a new directory random1

***cd ~/opencl-examples***

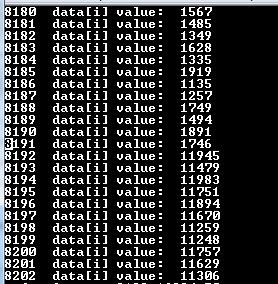
***mkdir random2***

***cd ~***

1. Push the file **random2** from the *Ubuntu laptop* to the EVM using Secure Copy (scp):
   1. Find the IP address of the EVM do ***ifconfig***
   2. Record the value of ipaddress of evm to be used in the next step
   3. **scp random2 root@<ipaddress of evm>: .**
   4. If scp asks you to confirm, write **yes**
   5. This will copy random2 to the /home/root/ directory on the EVM
   6. Move random2 from the home directory of the EVM to the random2 directory
   7. On the EVM do ***mv ~/random2 ~/ opencl-examples/random2/.***
2. Repeat step 2 for **ccode.obj** and **oclwrapper.cl** from the *Ubuntu laptop* to the EVM using Secure Copy (scp):
   1. **scp random2 root@<ipaddress of evm>: .**
   2. **scp ccode.obj root@<ipaddress of evm>:**
   3. **scp oclwrapper.cl root@<ipaddress of evm>: .**
   4. If scp asks you to confirm, write **yes**
   5. This will copy the above files to the /home/root/ directory on the EVM
   6. Move ccode.obj and oclwrapper.cl from the home directory of the EVM to the random2 directory
   7. On the EVM do ***mv ~/ccode.obj ~/ opencl-examples/random2/.***
   8. On the EVM do ***mv ~/oclwrapper.cl ~/ opencl-examples/random2/.***
3. ***cd ~/opencl-examples/random2***
4. Use the command ***ls –ltr*** and verify that **random2** executable was copied.
5. Run the code*:* ***./random2***



1. The output data is in the file **dataOut.txt**  
   Use the utilities more, cat, or any editor to look at the results.
2. The first value is the **output index**.
3. The second value in each row is a random number between 0 and 999, plus the core number multiplied by 10000. Thus, the most significant digit tells what DSP core generated the random number.
4. In the example shown, the first 8192 values were generated by Core 0. The next 8192 variables were generated by Core 1.



## Linux Instructions Used in this Lab

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Meaning** | **Usage Example** |
| ls | list elements in directory | **ls –ltr** list files details in reverse order in the current directory  **ls \*.ext** list all files with ext extension  **ls ../** list files in one directory above the current one |
| cp | Copy file | **cp random1 random2** copy file random1 to random2  **cp –R** copy directory |
| mv | Move file | **mv random1 random2** move file random1 to random2  **mv –R** move directory |
| sudo | Get super user permission | **sudo cp** use super-user permission to copy |
| mkdir | Make directory | **mkdir dir1** |
| pwd | Print working directory | **pwd** |
| whoami | Who am I? Displays the current user name | **whoami** |
| chmod | Change permission on a file or directory | **chmod 777 file** let anyone read/write and execute file  **chmod 777 –R dir** change permission of a directory and all files and sub-directories |
| make | Run Makefile script | **make** runs a script named Makefile in the directory  **make –f filename** make the script named filename |
| hostname | Identifies the computer name (needed for scp protocol) | hostname |
| scp | Getting file using secure protcol | **scp userName@hostName:/absoluteAddress/random1 .**  Copy the file random1 in directory with absolute address on the computer with host name to the current directory |
| gedit | Gnome graphical text editor on the Ubuntu machine | For instructions how to use gedit: <https://help.gnome.org/users/gedit/stable/> |
| vi | Unix visual editor text editor on the Ubuntu machine | For instructions how to use vi: <http://www.atmos.albany.edu/daes/atmclasses/atm350/vi_cheat_sheet.pdf> |