Laboratory Exercise 1

Getting Started with the Intel FPGA SDK for OpenCL

This is the first in a series of laboratory exercises that cover the use of the Intel FPGA SDK for OpenCL for accelerating workloads in artificial intelligence. Readers are expected to have prior experience with the Intel FPGAs through the completion of the *Digital Logic* laboratory exercises or similar. Completion of the *Linux Embedded Systems* laboratory exercises is also recommended. These exercises are designed to be used with Intel DE-series boards containing an SoC FPGA such as the DE10-Standard, DE10-Nano, and the DE1-SoC.

These exercises require you to run a Linux distribution on the board that has been designed to support compilation and execution of OpenCL applications. You must boot the Linux distribution on your board by following the instructions provided in the tutorial *Using Linux on DE-Series Boards*. The tutorial also describes how establish and use a commandline interface to the board, which you will require to complete these exercises.

If you have not yet installed the Intel FPGA SDK for OpenCL, follow the instructions provided in the tutorial *Using Intel FPGA SDK for OpenCL on DE-Series Boards*. The tutorial also describes the process for compiling and running an OpenCL application.

These exercises are not designed to teach you how to write OpenCL code. They instead focus on providing application-specific details, such as how a particular computer vision algorithm functions. Readers are encouraged to read the following documents which describe how to write and optimize OpenCL code:

- *Intel FPGA SDK for OpenCL Programming Guide*
- *Intel FPGA SDK for OpenCL Best Practices Guide*

Part 1

Compile the vector addition sample OpenCL application, whose source code is provided in */design_files/part1/*. The kernel code is provided in */design_files/part1/device/* and the host program code is provided in */design_files/part1/host/*. Execute the application on the board and observe its output. Record the runtime of the application when carrying out 1 million additions.

Part 2

As part of the compilation process in Part 1, the `aoc` compiler will have produced an HTML report in the */reports/* directory. Examine the report to determine:

- The total number of ALUTs used to compile the design.
- The initiation interval (II) of the for loop.
- The number of ALUTs and FFs used to implement the floating-point add operation.

Part 3

Examine *acl_quartus_report.txt* to determine the clockrate of the vector addition kernel. Based on the clockrate and the initiation interval of the main loop, estimate the runtime of the kernel to complete 1 million additions. How does your estimate compare to the actual measured runtime from Part 1?

Part 4

A modified version of the kernel from Part 1 is provided in */design_files/part4/device/*. This new version uses the pragma `#pragma unroll 16` to unroll the main loop 16 times. This means that each iteration of the loop will now carry out 16 additions concurrently, reducing the total number of loop iterations by a factor of 16.

Compile the modified kernel using the flag `-profile=all` to enable profiling. Determine the clock rate and estimate the runtime for 1 million inputs. Run the kernel and compare your estimated runtime to the actual runtime. You will notice that the kernel ran slower than expected, and is far from 16 times faster than the previous version. To determine the source of the discrepency, use the profiler to examine the contents of *profiler.mon*. Note that *profiler.mon* is automatically generated while running the OpenCL application if the kernel(s) were compiled using the flag `-profile=X`. Based on the profiled data, what was the bottleneck that caused the kernel to be slower than estimated? For detailed instructions on using the profiler, refer to the section *Launching the Intel FPGA Dynamic Profiler for OpenCL GUI (report)* in *Intel FPGA SDK for OpenCL Programming Guide*.