ECE 3270: Laboratory 5

LAB 5: OPENCL CONVOLUTION DESIGN

December 8, 2017

Rodrigo Ignacio Rojas Garcia
Clemson University
Department of Electrical and Computer Engineering
rrojas@clemson.edu

Abstract

Laboratory five focuses on creating a C program and using the OpenCL software to create and design a convolution algorithm. The laboratory was divided into two parts: Design of the convolution algorithm in the C programming language and the implementation of the algorithm with the OpenCL software. The laboratory consisted on designing a convolution algorithm to be used in part two and implement it with the OpenCL software. The outcome of the design was to display the time it took a computer running a Linux operating system and the DE1_SoC FPGA board to run the algorithm and compare their outputs. The purpose of the laboratory was to teach the student the process of designing a program using the C programming language and the OpenCL software.

1 Introduction

The fifth laboratory for ECE 3270, Digital Computing Design, had the purpose of introducing the student to the design of a Convolution calculation using OpenCL software. Laboratory five was divided into two parts. Part one consisted of the design of the convolution algorithm using the C programming language. Part two consisted of implementing the algorithm from Part one with the OpenCL software with the purpose of displaying the convolution of two vectors as well as the timing of the calculation in human readable form.

2 Part One: Design of Convolution Algorithm

In part one of the laboratory, the design of a convolution algorithm was to be created and designed in the C programming language. The purpose of the convolution algorithm is to "smooth" a signal by averaging each pixel with neighbors from both directions by using weighted values as "kernel" elements. The convolution algorithm was design was based on the convolution diagram provided in the Laboratory Five Manual[3] (Figure 1). The diagram demonstrates the convolution of a large vector of n elements and a vector of three elements. In the first and last stages of the calculation, only the overlapping elements are multiplied and added, the elements that are not used in the three element vector are ignored. The three element vector is called the "kernel". The three elements of the "kernel" sum to a value of "1" which is desired to compute the weighted averages. It should be noted that the highlighted elements on the diagram in each stage of the calculation are elements that use the dot product method to calculate the result. Each convolution result is stored in a single element of n element vector which is the same size as the n element vector used in the convolution calculation. The convolution algorithm was to be designed in the C programming language. The convolution algorithm was based on the diagram provided in the Laboratory Five Manual (Figure 1). As previously stated, the code contained three vectors, an input vector of size n, and three element vector of size three, and an output vector of same size as the input vector. The design of the C program had three requirements. The first requirement was for the three element vector was to have values of "1.0/3.0" on element 0, value of "1.0/2.0" on element 1, and value of "1.0/6.0" on element 2. The second requirement was to time the convolution algorithm. The timing consisted on only timing the convolution calculation from where it stores the calculation of the first element until the calculation of the last element is stored, not the code as a whole. It should be noted that the timing code used was based internet link provided from the Laboratory Five Manual. The last requirement was to verify that the beginning and the end neighbors of the input vector was to be handled properly because an error can cause an invalid output. To test the convolution calculation, a vector input of size 8 was given to the program and each result was checked; the output of the convolution calculation can be seen on figure 2. The C code for the convolution algorithm and timing can be seen in Figure 3.

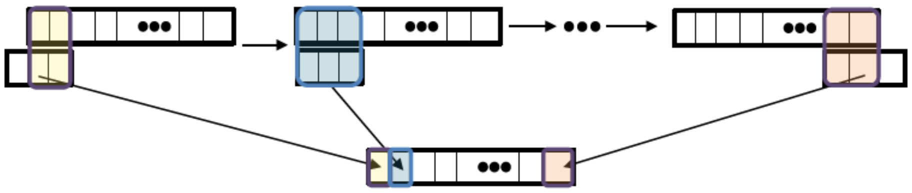
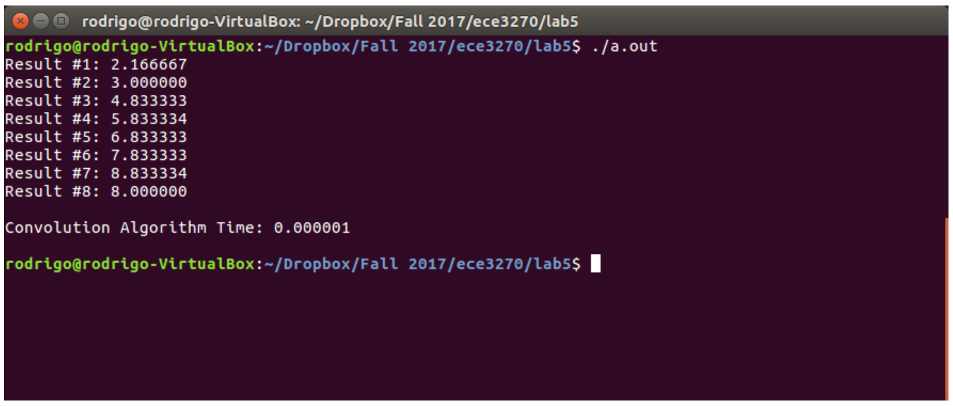
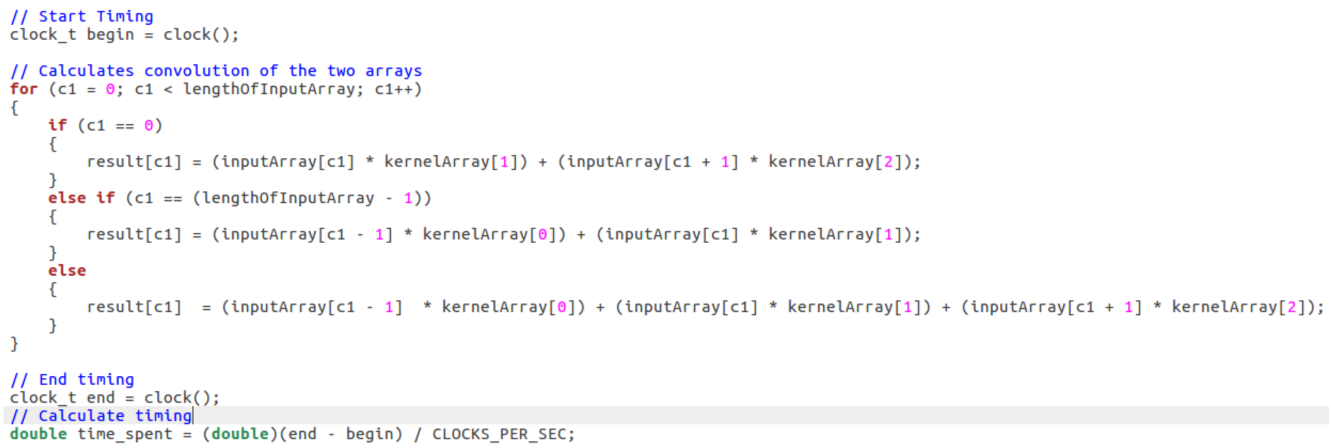


Figure 1: Convolution Diagram



```
rodrigo@rodrigo-VirtualBox: ~/Dropbox/Fall 2017/ece3270/lab5
rodrigo@rodrigo-VirtualBox:~/Dropbox/Fall 2017/ece3270/lab5$ ./a.out
Result #1: 2.166667
Result #2: 3.000000
Result #3: 4.833333
Result #4: 5.833334
Result #5: 6.833333
Result #6: 7.833333
Result #7: 8.833334
Result #8: 8.000000

Convolution Algorithm Time: 0.000001

rodrigo@rodrigo-VirtualBox:~/Dropbox/Fall 2017/ece3270/lab5$ 
```

Figure 2: Convolution Calculation C Program Output



```c
// Start Timing
clock_t begin = clock();

// Calculates convolution of the two arrays
for (c1 = 0; c1 < lengthOfInputArray; c1++)
{
    if (c1 == 0)
    {
        result[c1] = (inputArray[c1] * kernelArray[1]) + (inputArray[c1 + 1] * kernelArray[2]);
    }
    else if (c1 == (lengthOfInputArray - 1))
    {
        result[c1] = (inputArray[c1 - 1] * kernelArray[0]) + (inputArray[c1] * kernelArray[1]);
    }
    else
    {
        result[c1]  = (inputArray[c1 - 1]  * kernelArray[0]) + (inputArray[c1] * kernelArray[1]) + (inputArray[c1 + 1] * kernelArray[2]);
    }
}

// End timing
clock_t end = clock();
// Calculate timing
double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
```

Figure 3: Convolution Algorithm C Code

3 Part Two: OpenCL Implementation

Part two of laboratory five focused on implementing the design convolution algorithm with the OpenCL software. The implementation of the convolution algorithm on the

OpenCL software had six requirements. The first requirement asked for the complete convolution algorithm in OpenCL by referring to the code designed on part one. The second requirement was to utilize the OpenCL emulation to verify that the convolution result was correct. The third requirement asked to modify the "Makefile" with the purpose of when compilation performed the executable was named appropriately and also to not link to a library since laboratory five does not use VHDL code. The fourth requirement was to modify the folder structure and all the file names based on the name of laboratory five. The first requirement was completed by modifying the file "convolution.cl" and file "main.cpp". In the "convolution.cl" file, the convolution algorithm was implemented on the kernel function named "convolution_kernel" in two steps. First, the kernel inputs and output were modified. The inputs consisted of two "__global" variables named "inputArray" and "size". The input "inputArray" is of type "float" and corresponds to the input vector used to calculate the convolution, the input "size" is of type "int" and corresponds to the size of the input vector. The output was a "__global" variable of type "float" named "result" which corresponds to the output vector which contains the convolution result of each element. Second, inside the kernel function the convolution algorithm designed on part one was implemented with minor changes. The algorithm consisted of a variable named "c1" of type "int" which stored the value of the current element accessed on the input vector and output vector. This was done by setting variable "c1" equal to the function "get_global_id(0)". After this, the three element vector named "kernelArray" of type "float" was declared and initialized to the same values as the design from part one. Then, three conditional statements were declared. The first conditional statement consisted of an "if" statement that ran only once when the first element of the "inputArray" vector was accessed to calculate the convolution and store it on the "result" output vector. The second conditional statement, "else if" consisted of the convolution calculation of all other elements except for the first and last element of the "inputArray" vector and the calculation was stored on "result" output vector. The last conditional statement "else" that ran only once when the last element of the "inputArray" vector was accessed to calculate the convolution and store it on the "result" output vector. In the "main.cpp" file, there were minor code changes made for the "convolution_kernel" function could be utilized and ran in the OpenCL software. First, the function "init_opencl()" of return value "bool" was modified. The function was modified by changing the name of the binary file to "convolution" which changes the name of the "aocx" once compiled to that determined name. Also, the pointer "kernel_name" was changed to the name of the kernel from first requirement, "convolution_kernel", with the purpose of referencing to this kernel function when the OpenCL software creates a kernel. Second, the function "init_problem" was of return value "void" was modified. The function was modified by setting "int" variable "N" equal to "8" inside the "if" conditional statement. Also, in this conditional statement, the inputs "input_a" and "input_b" which correspond to the input vector and size of the vector were allocated in memory by using function "malloc". After this, a "for" loop was implemented which ran from value of "0" to value of "N" in which stored random numbers to array "input_a" and also "input_b" was set to the value of variable "N". At the end of the "if" conditional the variable "output" was allocated in memory with function "malloc" which corresponds to the output "result" of the convolution algorithm. third, the function "run" of return value "void" was modified. The function was modified by setting the timing varibles "start_time" and "end_time" before and after the variable "status" and after the function "clWaitForEvent()" correpsondingly with the purpose of timing the algorithm in the OpenCL software. Also, two "for" loops were implemented to display the value of each element of the input vector as well as the value of each element of the output vector. The second requirement of utilizing the emulation to verify the result was accomplished by compiling and running the convolution algorithm of the C program and checking manually that the output was correct. The third requirement was completed by modifying two "Makefile" files correspoding to the "aocx" file and executable file "convolution". The "Makefile" corresponding to the compilation file "aocx" was modified by removing all reference to lib, the lib oject, and the "-l $(PROJECT)_rtl.aoclib -L" section of the default and optm targets. This modification was made so the "Makefile" does not link a library to the compilation file since no VHDL code is implemented in this laboratory. After this the "Makefile" corresponding to the executable file "convolution" was modified by chaning the value of variable "TARGET" to "convolution". This was done so the executable file corresponds to the convolution algorithm created. The fourth requirement was completed by completing the second and third requirements.

3.1 Compilation of Program and Storage of SD Card

After the completion of fourth requirement, compilation of the program could proceed. First, a terminal was opened in the directory "Convolution/aolc_.convolution". This directory contained file named "Makefile" which was executed by using the command "make", the outcome of executing this command was an executable file named "convolution". Second, directory of terminal was changed to "Convolution/aolc_.convolution/device". This directory also contained a file named "Makefile" which was executed by using command "make", the outcome of executing the command was an executable file named "convolution.aocx". The purpose of compilation was to obtain the files "convolution" and "convolution.aocx" with the purpose of executing them on board DE1_.SoC board. After the compilation of the kernel files was complete, the executable files "convolution" and "convolution.aocx" were transferred to a SD card. The process of transferring the executable files to the SD card could only be completed in a computer running a Linux Operating System. Once the SD card was inserted into the computer, a folder named "lab5" was created in directory "home/root" in which executable files "convolution" and "convolution.aocx" were stored at.

3.2 DE1_SoC Configuration and Program Execution

After the executable files were compiled and stored in an SD card the process of configuring the DE1_SoC board's hardware and software configurations and program execution could proceed. First, the DE1_SoC board was configured to run on an OpenCL configuration. The DE1_SoC board as default runs in the JTAG configuration, to configure it to the

OpenCL configuration, switch SW10 on the back of the board had to be set to value ”110101”. See Figure 4 for reference. After this was completed, the microSD card could be inserted into the board and upon powering on the board, red LEDs started showing a binary counting sequence. After the hardware OpenCL configuration was completed, the software configuration had to be completed. The software setup consisted on accessing the DE1_SoC board through a ULLAB computer. To do this, a terminal was opened and command ”minicom” was entered. This command made the board start initialization. Then, the type of user was prompted and user ”root” was entered. After board was accessed, command command ”./init_opencl.sh” was entered which configured the board to run OpenCL code. All steps regarding DE1_SoC configuration were based on manual provided by laboratory instructor[2]. After the software configuration was completed and execution of executable files transferred to the SD card could be executed. To accomplish this, directory ”root/home/lab5” was accessed. In this directory, executable file ”convolution” and ”convolution.aocx” were located here. To run these executable files, command ”chmod +x convolution” was entered which made file ”convolution” an executable file. After this, command ”./convolution -t” was entered which executed the file and the output displayed the input vector values as well as the output vector values. See Figure 5. It can be concluded that OpenCL program gave the correct results for the convolution algorithm by comparing the output of the C program and the OpenCL program figures 2 and 5 respectively.

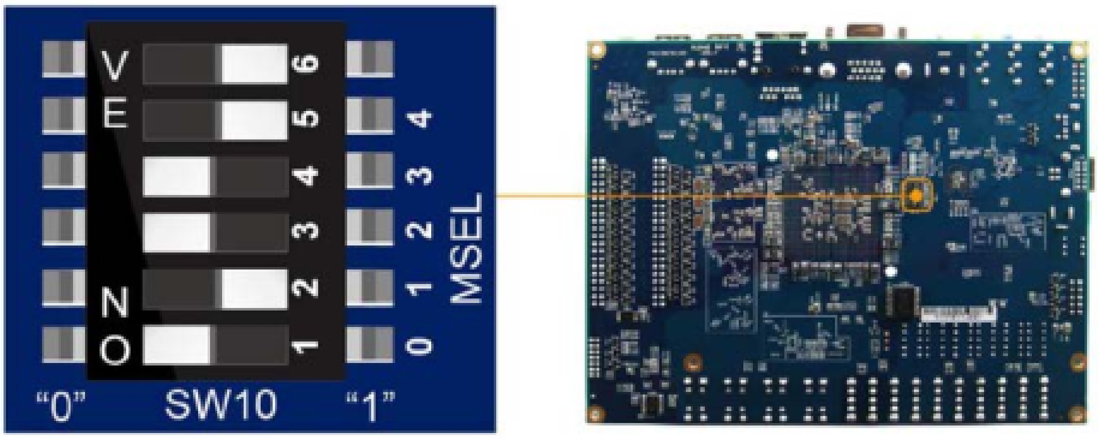
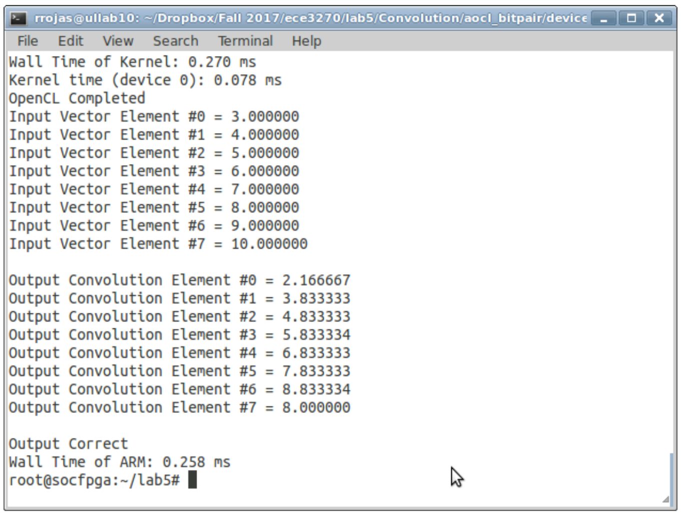


Figure 4: DE1_SoC Pins

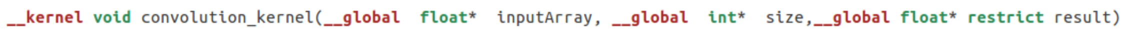

rrojas@ullab10: ~/Dropbox/Fall 2017/ece3270/lab5/Convolution/aocl_bitpair/device
File Edit View Search Terminal Help

```
Wall Time of Kernel: 0.270 ms
Kernel time (device 0): 0.078 ms
OpenCL Completed
Input Vector Element #0 = 3.000000
Input Vector Element #1 = 4.000000
Input Vector Element #2 = 5.000000
Input Vector Element #3 = 6.000000
Input Vector Element #4 = 7.000000
Input Vector Element #5 = 8.000000
Input Vector Element #6 = 9.000000
Input Vector Element #7 = 10.000000

Output Convolution Element #0 = 2.166667
Output Convolution Element #1 = 3.833333
Output Convolution Element #2 = 4.833333
Output Convolution Element #3 = 5.833334
Output Convolution Element #4 = 6.833333
Output Convolution Element #5 = 7.833333
Output Convolution Element #6 = 8.833334
Output Convolution Element #7 = 8.000000

Output Correct
Wall Time of ARM: 0.258 ms
root@socfpga:~/lab5#
```

Figure 5: Convolution OpenCL Output

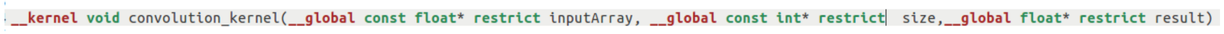
3.3 OpenCL Optimization

After the compilation and execution of the OpenCL software, another requirement had to be completed. The requirement was to optimize the timing of the algorithm implemented on OpenCL software by using the "AOCL Best Practices Guide Document" provided in by the laboratory instructor. The optimization requirement of optimizing the OpenCL software was completed by using the "AOCL Best Practices Guide Document" provided by the laboratory instruction and modifying the "convolution.cl" file. The "convolution.cl" file was modified by changing input variable from "__global" to "__global const". The difference between using "__global const" and "__global", is that "it creates a private cache for each load or store operation"[1]. Another optimization made to optimize the timing of the algorithm was to include the "restrict" word after the variable type declaration (e.g "float * restrict). The inclusion of the word "restrict" "prevents the Altera Offline Compiler from creating unnecessary memory dependencies between non-conflicting load and store operations"[1]. To demonstrate the difference in the optimization of OpenCL, the program was ran with a input vector of size of 10 million without the optimization's and with the optimization's. The optimization code and OpenCL can be seen by comparing Figures 6 and 7 and figures 8 and 9 correspondingly. It can be seen that the timing between the non-optimized OpenCL and optimized OpenCL differ, in which the optimized OpenCL output runs the convolution algorithm faster.



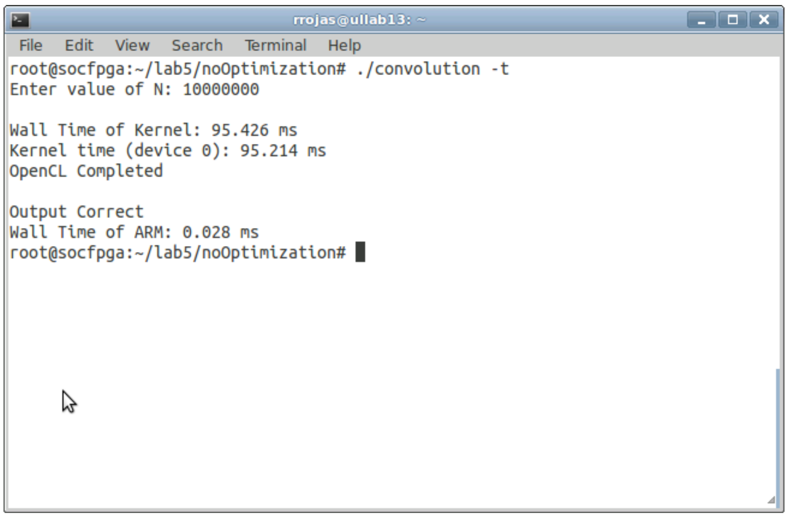
```
__kernel void convolution_kernel(__global  float*  inputArray, __global  int*  size,__global float* restrict result)
```

Figure 6: Convolution OpenCL Without Optimization Kernel



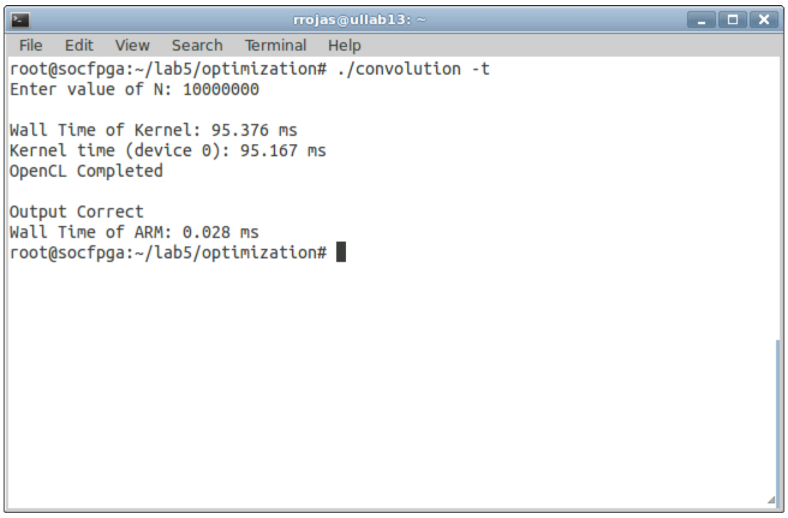
```
__kernel void convolution_kernel(__global const float* restrict inputArray, __global const int* restrict  size,__global float* restrict result)
```

Figure 7: Convolution OpenCL Optimization Kernel



```
root@socfpga:~/lab5/noOptimization# ./convolution -t
Enter value of N: 10000000

Wall Time of Kernel: 95.426 ms
Kernel time (device 0): 95.214 ms
OpenCL Completed

Output Correct
Wall Time of ARM: 0.028 ms
root@socfpga:~/lab5/noOptimization#
```

Figure 8: Convolution OpenCL Without Optimization Output Time

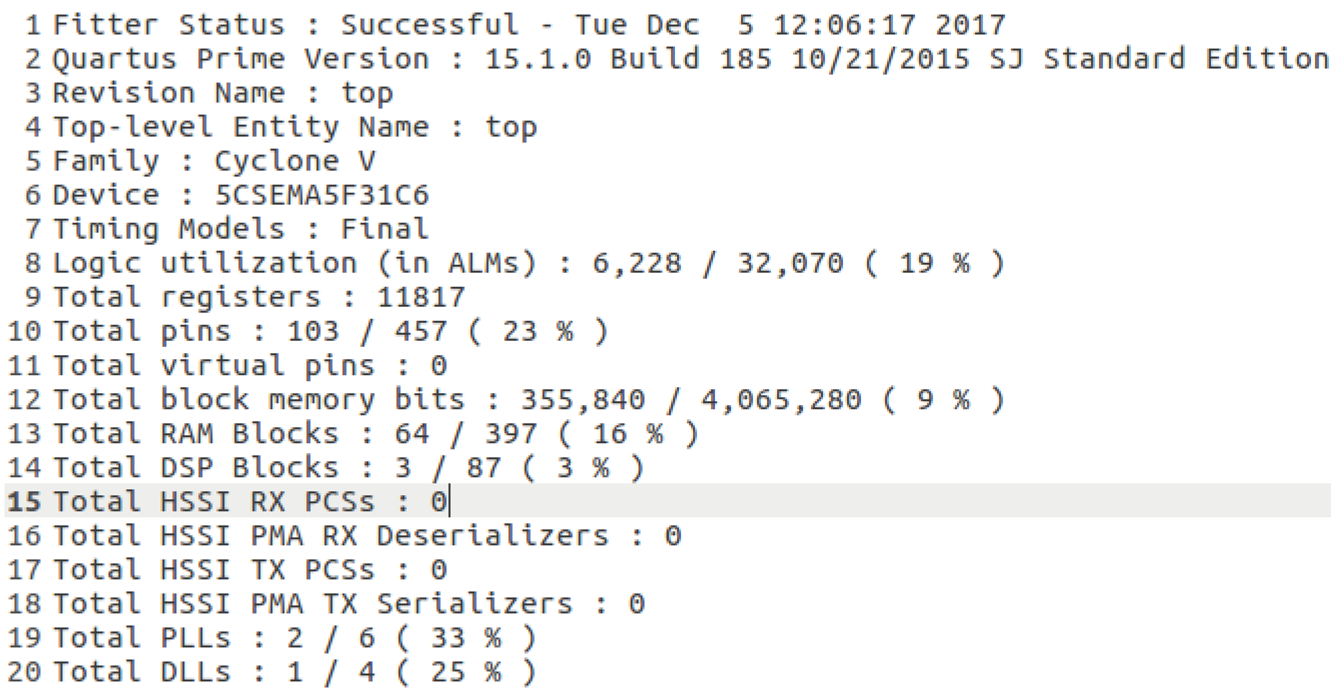


```
root@socfpga:~/lab5/optimization# ./convolution -t
Enter value of N: 10000000

Wall Time of Kernel: 95.376 ms
Kernel time (device 0): 95.167 ms
OpenCL Completed

Output Correct
Wall Time of ARM: 0.028 ms
root@socfpga:~/lab5/optimization#
```

Figure 9: Convolution OpenCL With Optimization Output Time

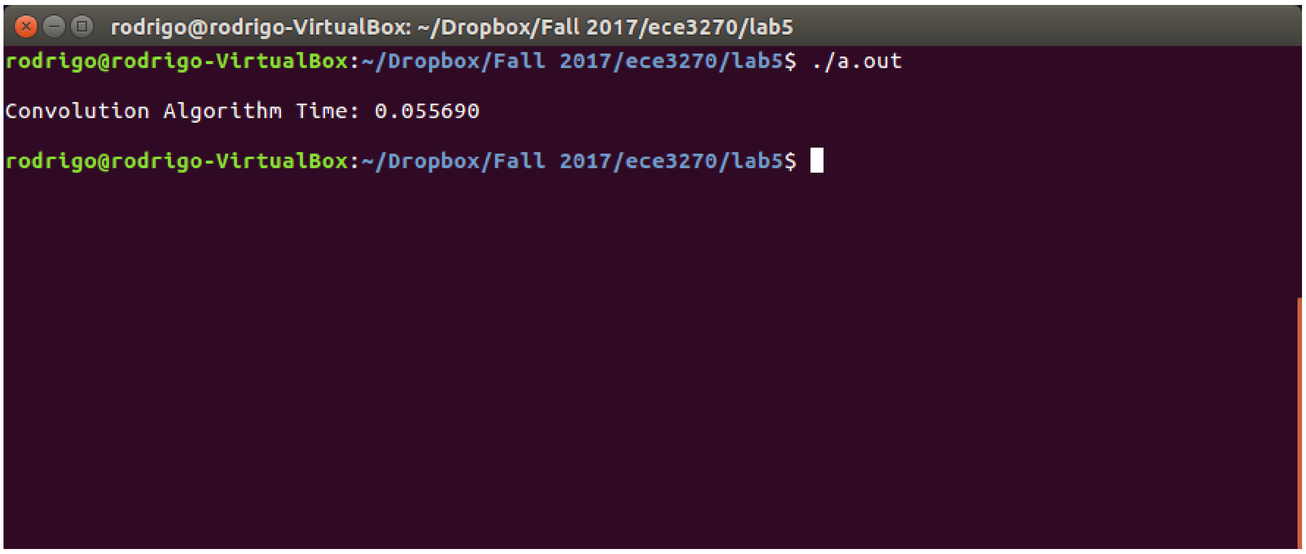
4 C Program and OpenCL Software

After the optimization of OpenCL was accomplished, the C convolution algorithm timing could be compared with the convolution algorithm ran on the DE1_SoC board. The C program and the OpenCL software were compared by running the convolution algorithm with an input vector of 10 million. The expected outcome was for the board to run the algorithm faster than the timing of the C algorithm because the board calculates the convolution in parallel and not sequentially. The expected outcome was wrong, the C algorithm timing was faster than the FPGA algorithm timing. The C algorithm can be faster for various of reasons. The computer in which the C program is ran, the CPU has a higher clock speed of 3.9 GHz compared to the board which has a clock speed of 1 GHz. It could be inferred that the difference of CPU speed can give an advantage to the C program even though the FPGA board runs the program in parallel instead of a sequential manner. Another reason that the C algorithm timing can be faster is because of the utilization of the hardware on the FPGA. Based on the file ”top.fit.summary” (Figure 10), the DE1_SoC board only uses 19 percent of its logic. It can be concluded that the board could run a faster algorithm that the C program, but due to hardware limitations, the C program beats the FPGA board in this occasion. The difference between the C program timing and the OpenCL program timing can be seen on figures 11 and 12. It can also be shown that the C algorithm timing is faster by using the speedup equation given in the Laboratory Five manual (Figure 13). The calculation of this equation gives a value of 0.583899514 by using values from figures 11 and 12, meaning that the C algorithm timing is faster than the OpenCL algorithm timing which demonstrates that there was no speedup gained by running the algorithm through hardware.



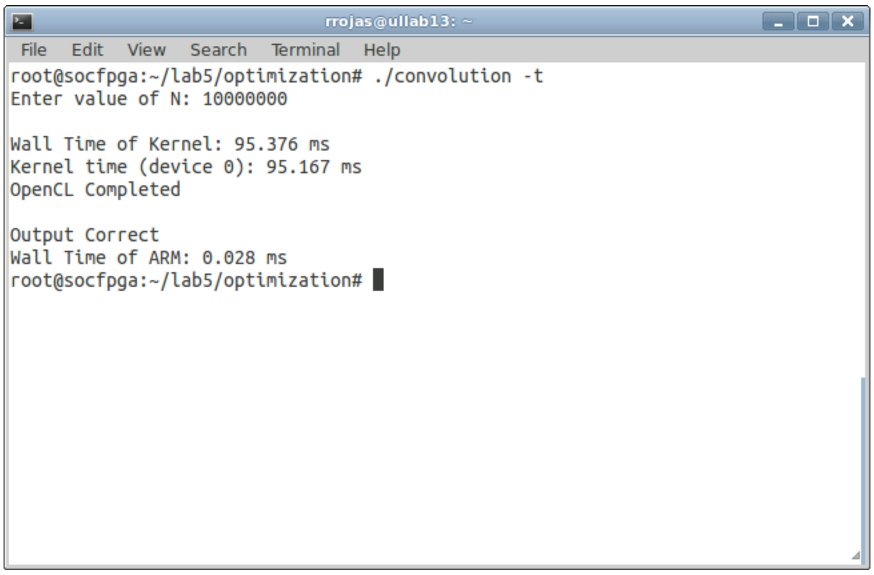
```
Fitter Status : Successful - Tue Dec  5 12:06:17 2017
Quartus Prime Version : 15.1.0 Build 185 10/21/2015 SJ Standard Edition
Revision Name : top
Top-level Entity Name : top
Family : Cyclone V
Device : 5CSEMA5F31C6
Timing Models : Final
Logic utilization (in ALMs) : 6,228 / 32,070 ( 19 % )
Total registers : 11817
Total pins : 103 / 457 ( 23 % )
Total virtual pins : 0
Total block memory bits : 355,840 / 4,065,280 ( 9 % )
Total RAM Blocks : 64 / 397 ( 16 % )
Total DSP Blocks : 3 / 87 ( 3 % )
Total HSSI RX PCSs : 0
Total HSSI PMA RX Deserializers : 0
Total HSSI TX PCSs : 0
Total HSSI PMA TX Serializers : 0
Total PLLs : 2 / 6 ( 33 % )
Total DLLs : 1 / 4 ( 25 % )
```

Figure 10: FPGA board summary usage


rodrigo@rodrigo-VirtualBox: ~/Dropbox/Fall 2017/ece3270/lab5

```
rodrigo@rodrigo-VirtualBox:~/Dropbox/Fall 2017/ece3270/lab5$ ./a.out

Convolution Algorithm Time: 0.055690

rodrigo@rodrigo-VirtualBox:~/Dropbox/Fall 2017/ece3270/lab5$
```

Figure 11: Convolution Algorithm Time for C program



```
root@socfpga:~/lab5/optimization# ./convolution -t
Enter value of N: 10000000

Wall Time of Kernel: 95.376 ms
Kernel time (device 0): 95.167 ms
OpenCL Completed

Output Correct
Wall Time of ARM: 0.028 ms
root@socfpga:~/lab5/optimization#
```

Figure 12: Convolution OpenCL With Optimization Output Time

$$Speedup = \frac{Time_C}{Time_{OpenCL}}$$

Figure 13: Speedup Equation

5 Conclusion

The purpose of laboratory five was to teach the student on how to implement an algorithm made in the C programming language on the OpenCL software. The outcome of the laboratory five taught the student how to create and design OpenCL software and how parallel calculations are faster on hardware rather than software. The laboratory accomplished this by making the student design a convolution algorithm in the C programming language and implementing it with the OpenCL software to compare the execution time of the algorithms in software and hardware as well as displaying the output in human readable form.

References

[1] Canvas. *AOLC Best Practices Guide.pdf*. Canvas, 2017.

[2] Canvas. *Compilation and Usage.pdf*. Canvas, 2017.

[3] Canvas. *Convolution OpenCL.pdf*. Canvas, 2017.