ECE5775 High-Level Digital Design Automation, Fall 2017
School of Electrical Computer Engineering, Cornell University

---

Lab 3: Digit Recognition System (Part 2)

Due Friday, October 6, 2018, 11:59pm

---

1 Introduction

In the previous lab, we introduced how to actually deploy applications on AWS FPGA. In this lab, you will be responsible for implementing your `digitrec` design from Lab 2 on the AWS FPGA instance. Furthermore, you will apply **loop pipelining** to optimize the design and measure the execution time of the software-emulation, baseline-FPGA, and optimized-FPGA implementations of `digitrec`. Finally, you will use the information found in Vivado HLS's generated reports to estimate the runtime of the FPGA designs and compare your numbers with measurements made on the actual FPGA hardware.

2 Materials

You are given a zip file named *lab3.zip* on *ecelinux* under */classes/ece5775/labs*. It contains two directories: **lab3** and **harness**.The directory **harness** is the same as before, actually you should placed it in your working directory and reuse for all the labs.

- **lab3**: contains all the codes and data of this lab.
- **harness**: contains the wrapper code of OpenCL APIs and top-level makefile. Students are **not required** to understand the content of this directory.

The structure of directory **lab3** is described as following:

- **src**:
  - **host**: contains all the codes for host.
    - `check_result.cpp`: contains function to print out the errors and calculate the overall error rate.
    - `check_result.h`: the header file of check_result.cpp.
    - `digit_recognition.cpp`: the main file, defines the implementation flow of this application
    - `testing_data.h`: declares two constant arrays including the testing data and corresponding expected labels.
    - `training_data.h`: combines all the training data sets (i.e., data/testing set.dat) into a constant array.
    - `typedefs.h`: the constant definitions and typedefs for the host.
    - `utils.cpp`: contains the functions to print usage and parse the command line arguments
    - `utils.h`: the header file of
  - **ocl**: contains the actual kernel file.
    - `digitrec.cpp`: **an incomplete** file that defines the kernel function Digitrec.
    - `functions.cpp`: an **incomplete** file of the functions used in `digitrec.cpp`.
- **data**:

  - `training_set_#.dat`: training set for digit #, where # = 0, 1, 2, ..., 9.
  - `testing_set.dat`: a set of testing instances with corresponding expected labels.
  - `expected.dat`: the expected labels of the testing set.
- **makefile**: the makefile to compile the this application.
- **run.sh**: this script runs the emulation with k value set to 3 by default.

3 Design Overview

You will again use the k-nearest-neighbors (k-NN) algorithm for digit recognition (consult the Lab 2 document for details on the k-NN algorithm). **Because the focus in this lab is the hardware implementation, the value of `k` will be fixed to 3.** You will implement and evaluate the performance for three designs:

- A **baseline** `digitrec` design that does not use any HLS optimization directives.
- An **unrolled** `digitrec` design which is similar to what you did in Lab 2 where unrolling and array partitioning are applied.
- A **pipelined** `digitrec` design which applies loop pipelining in addition to the previous optimizations.

You will need to apply HLS optimazation directives in the kernel and then run each design on the hardware. You will use the information from the Vivado HLS synthesis reportet

to **estimate** the performance of your hardware design, and verify that the physical hardware achieves a performance close to the estimate. Also, you will measure the performances of the software execution on both the **AWS f1 instance** and **AWS m4 instance** for comparison.

4 Guidelines and Hints

4.1 Coding and Debugging

Your first task is to complete the application implementation on **AWS m4 instance**. Similar to Lab 2, the main body of the `digitrec` function is finished, and you only need to complete `update_knn` and `vote_knn`. If you add print statements to debug your code, make sure to remove them before doing runtime measurement.

Your next task is do the hardware synthesis and then run the application on the AWS F1 instance.The process to run application on hardware is exactly the same as in lab1. Please be sure to finish your designing and debugging on the m4 instance, run software emulation to make sure your implementation is correct and then start the hardware synthesis, because **the hardware synthesis could take around three hours to generate the xclbin file**.

4.2 Hardware Design Optimization

You are required to modify your kernel to optimize the application. This is done by adding HLS optimization pragmas to the appropriate place in your kernel. Use the provided script `run.sh` to test your optimization to make sure it is correct, then deploy on the AWS f1 instance to run on hardware. Your source code will be identical between these designs.

After synthesizing the unrolled design, you should check that the latency is reduced by around `10x` in the synthesis report. In pipelined design, we are pipelining the outer loop (labeled as `L1800`, which iterates 1800 times). **So please verify after pipelining that the `L1800` loop is indeed pipelined to an II of 1 in the report.**

If you are interested to learning more about the pipelining directive, check out the following reference:

- Vivado Design Suite User Guide, High-Level Synthesis, UG902 (v2016.2) [1]
  - *set_directive_pipeline* p.472

  You can find examples of code snippets using the pipeline pragma throughput the user guide.

4.3 Bonus

There is a **bonus** section on this assignment. Currently the software-emulation implementation of `digitrec` is poorly optimized for a microprocessor, since the code was written to facilitate hardware synthesis. To make the comparison between the CPU and FPGA more fair, you are encouraged to optimize the software-only `digitrec` on `AWS m4 instance`. You can use any methods (e.g., replace HLS-specific ap_int data types with native integer types), as long as the final solution remains a valid 3-NN algorithm that yields an error rate below 10%. **The top 4 students who manage to achieve the fastest software runtime would be receiving ONE extra point (out of 5).**

You should create a new directory named **bonus** for this part, and copy over any files you need. Do not make changes to `digitrec_test.cpp`.

4.4 Report

(not updated yet, will be updated after the previous parts are settled)

- Please write your report in a **single-column and single-space format with a 10pt font size. Page limit is 2. Please include your names and NetIDs on the report.**
- The report should start with an overview of the document. This should inform the reader what the report is about, and highlight the major results. In other words, this is similar to an abstract in a technical document. Likewise there should be a summary, describing the results, and highlight the important points.
- There should be a section describing how you implemented the `digitrec` system. Explain how you communicated data between the processor and FPGA, as well as what preprocessing was necessary to put the data in the right format.
- There should be a section discussing the effects of each design (baseline, unrolled, and pipelined) on the synthesized hardware. This section should contain a table which reports the latency and resource usage of each design. Compare these numbers discuss them using your understanding of how the unrolling and pipelining optimizations work.
- There should be a section reporting the measured performance of each `digitrec` system implementation. Specifically, **this section should contain a table which lists the observed runtime for each implementation, including ecelinux-software, zedboard-software (i.e., ARM), zedboard-fpga-baseline, zedboard-fpga-unrolled, and zedboard-fpga-pipeline**. This table should also have a column that reports the runtime speedup normalized against zedboard-software.

  In addition, for the FPGA implementations, please use the synthesis report to estimate execution time in hardware (note that the FPGA clock is operating at 100MHz in this particular setup). **Please discuss how you obtained your estimations, and compare your theoretical and observed results and explain any discrepancies.**
- If you do the **bonus** portion of the assignment, include a section describing what optimizations you made to the software-only implementation and report its performance in the tables above. If you made a number of changes, discuss which ones you believe to be the most significant. **This section does not count towards the page limit.**
- All of the figures and tables should have captions. These captions should do their best to explain the figure (explain axis, units, etc.). Ideally you can understand the report just by looking at the figures and captions. But please avoid just putting some results and never saying anything about them.
- The report should only show screenshots from the tool when they demonstrate some significant idea. If you do use screenshots, make sure they are readable (e.g., not blurry). In general, you are expected to create your own figures. While more time consuming, it allows you to show the exact results, figures, and ideas you wish to present.

5 Deliverables

Please submit your lab on CMS. You are expected to submit your report and your code and scripts (and only these files, not the project files generated by the tool) in a zipped file named **digitrec2.zip** that contains the following contents:

- **report.pdf**: the project report in pdf format.
- The folders **ecelinux**, **zedboard**, and **bonus**. These should contain the completed source files for the software-only, FPGA, and optimized software-only implementations of the `digitrec` design. Make sure the design can be built using the Makefile and scripts in the folders. Please run `make clean` to remove all the generated output files.

6 Acknowledgement

The baseline FPGA+Linux setup used in tutorial is based on the Xillinux distribution provided by Xillybus Ltd. (http://xillybus.com/xillinux)

References

[1] Xilinx Inc. *Vivado Design Suite User Guide, High-Level Synthesis, UG902 (v2016.2)*. 2014.

[2] Xilinx Inc., *Introduction to FPGA Design with Vivado High-Level Synthesis*, 2013.