ECE6775 High-Level Digital Design Automation, Fall 2025
School of Electrical Computer Engineering, Cornell University

---

Lab 3: Digit Recognition System (Part 2)
Due Friday, October 10, 2025, 11:59pm
Late submission: 4% penalty per day; cannot be late by more than 6 days

---

1 Introduction

In the tutorial on *Software-Hardware Codesign with CORDIC*, you saw how we modified the CORDIC design from Lab 1 to implement the synthesized RTL on the Xilinx Zynq field-programmable system-on-chip (SoC). In this lab, you will be responsible for implementing your `digitrec` design from Lab 2 on the same device. Furthermore, you will apply **loop pipelining** to optimize the design and measure the execution time of the software-only, baseline-FPGA, and optimized-FPGA implementations of `digitrec`.

2 Materials

You are given a zip file named *lab*3.*zip* on *ecelinux* under */classes/ece*6775/*labs*, which contains the following directories:

- **ecelinux**: contains an **incomplete** C++ project for you to build the `digitrec` HLS design and synthesize it to RTL. This code should be completed on `ecelinux`.
- **zedboard**: contains *symbolic links* to the files in the **ecelinux** directory required for software execution of `digitrec` on CPU. Also contains an **incomplete** C++ project to build the host program for invoking the `digitrec` module on FPGA.

Note that a *symbolic link* is a special file which points to another file — accessing the link basically accesses the original file. The files `digitrec.h`, `digitrec.cpp`, and `digitrec_test.cpp` in the **zedboard** directory point to their counterparts in the **ecelinux** directory. This means you only have to complete the `digitrec` design in the **ecelinux** directory and the changes will be automatically propagated to the **zedboard** directory. Make sure to copy both directories to the ZedBoard or else the link will be broken.

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` is 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 (`vivado_hls -f run_base.tcl`).
- An **unrolled** `digitrec` design which is similar to what you did in Lab 2 where unrolling and array partitioning are applied (`vivado_hls -f run_unroll.tcl`).
- A **pipelined** `digitrec` design which applies loop pipelining in addition to the previous optimizations (`vivado_hls -f run_pipeline.tcl`).

You will use the information from the Vivado HLS synthesis report to **estimate** the performance of your hardware design, and examine if the physical hardware achieves a performance close to the estimate. You will also measure the performance of the software execution on both the ZedBoard and **ecelinux** for comparison. Timers identical to those from the CORDIC tutorial will be used. In total, you will be comparing across 5 different design points: x86 software, ARM software, baseline accelerator, unrolled accelerator, and pipelined accelerator.

4 Guidelines and Hints

4.1 Coding and Debugging

Your first task is to complete the software-only implementation and test it on **ecelinux** and a ZedBoard. Similar to Lab 2, the main body of the `digitrec` function is finished, and you only need to complete `update_knn` and `vote_knn`. But this time you are required to **only use constant-bound loops in the `update_knn` function**. To run the software on ecelinux, first go to the `lab3/ecelinux` directory, then run `make`.

```
unzip lab3.zip                              # unzip the archive
cd lab3/ecelinux
source /classes/ece6775/setup-ece6775.sh  # setup env
make                                        # builds and runs the csim
```

To run the software on a ZedBoard (i.e., ARM CPU), first copy lab3 files to a ZedBoard, then go to the directory `lab3/zedboard`, finally run `make sw`. Outside your lab3 working directory (assume it's named *lab3*), do:

```
zip -y -r lab3.zip lab3                          # archive your files
scp lab3.zip <user>@zhang-zedboard-xx.ece.cornell.edu:~
ssh <user>@zhang-zedboard-xx.ece.cornell.edu   # log in to a ZedBoard
unzip lab3.zip                                   # unzip the archive
cd lab3/zedboard
make sw                                          # builds and runs software
```

To complete the hardware design, you will also need to fill in `dut` with code to communicate with the FIFO channels. The *digitrec_test.cpp* testbench is responsible for running the software-only implementation of `digitrec`. Timers have already been added to the csim. **If you add print statements to debug your code, make sure to remove them before measuring the execution time.**

Once csim is successful, the next step is to generate the FPGA implementation for a ZedBoard. The process of compiling the bitstream, logging onto a ZedBoard, and programming the ZedBoard using the bitstream is identical to what we showed in the CORDIC tutorial. **Be aware that it can take 20-30 minutes to generate the bitstream**. If the bitstream is generated without errors, it is time to copy the lab root directory over to the ZedBoard with `scp`. Please do not rename this bitstream, since the board automatically looks for it by name after rebooting.

Lastly, you are also responsible for completing the `host` program. We have taken care of reading the input and reference data sets from file and activating the timers. However, you need to write a routine that uses **batch data transfer** in your code to minimize communication overhead. Then, to execute the accelerator on the FPGA, you need to go to directory *lab3/zedboard* and run `make fpga`.

4.2 Hardware Design Optimization

Use the provided scripts *run_base.tcl*, *run_unroll.tcl*, and *run_pipeline.tcl* to synthesize your design. Your source code will be identical between these designs. Note that we have already added the necessary HLS optimization directives in each of these Tcl scripts.

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 `LOOP_I_1800`, which iterates 1800 times). **So please verify that after pipelining the achieved II of the `LOOP_I_1800` loop is indeed 1 in the report.** If the achieved II is not 1, please check the HLS log for relative warnings. If you are interested in learning more about the pipelining directive, check out the following reference:

- Vivado Design Suite User Guide, High-Level Synthesis, UG902 (v2019.2) [1]
  - Function and Loop Pipelining (p.125)
  - `set_directive_pipeline` (p.450)

4.3 Report

- Please write your report in a **single-column, single-spaced format with a 10pt font**. The main text must fit on **TWO pages**. You may add **one optional appendix page** for tables and figures only. You may place figures side-by-side in one row to save space. **Include your name and NetID 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.
- There should be a section discussing the effects of each design (baseline, unrolled, and pipelined) on the synthesized hardware. Please include a table which reports the latency and resource usage of each design. Compare these numbers and 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 execution time 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 speedup normalized against zedboard-software.

- 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** and **zedboard**. 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.

It is preferred that you run `make digitrec2.zip` in the lab root directory to create your submission to prevent any unneeded files or directories being included.

6 Acknowledgement

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

References

[1] Xilinx Inc., *Vivado Design Suite User Guide: High-Level Synthesis UG902 (v2019.2)*, Available at `https://docs.amd.com/v/u/2019.2-English/ug902-vivado-high-level-synthesis`