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

---

Lab 1: CORDIC Design

Due Friday, September 8, 2017, 11:59pm

---

1 Introduction

COordinate **R**otation **DI**gital **C**omputer (CORDIC) is a method for calculating a variety of functions including trigonometric and hyperbolic. The various functions are calculated through an iterative set of vector rotations. At the end of these rotations, the value of the function is easily determined from the (x, y) coordinate. A CORDIC is often used to achieve low-cost multiplierless sine/cosine implementations in FPGA as well as ASIC designs. To obtain a good understanding of CORDIC for the purpose of this lab, **please read Chapter 15 of *A Practical Introduction to Hardware/Software Codesign*** [?].

2 Materials

You are given a zip file named *lab1.zip* on the ecelinux server, which contains the following files for you to build the project.

- **cordic**:
  - **kernel**:contains the actual kernel file.
    - `cnn.cpp`: **an incomplete** file that defines the kernel function cordic.
  - **host**:contains all the codes for host.
    - *cordic.h*: the header file with various macro and type definitions that may be useful for developing your code.
    - *check_result.cpp*: a test bench that helps verify your code.
    - *check_result.h*: a header file of check_result.cpp.
    - *utils.cpp*: some utility functions sued to parse command line arguments.
    - *utils.h*: a header file of utils.cpp.
    - *typedefs.h*: a header file that defines the target device for the host.
    - *Makefile*: a makefile to compile the this application using double data type.
    - *fixed_type.mk*: a makefile to compile the this application using fixed point data type.
    - *run_float.sh*: this script runs the emulation using double data type.
    - *run_fixed.sh*: this script runs the emulation using fixed point data type.

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

Before starting your assignment, please **copy and unzip the zip file to your home directory**. Please be sure to **source the setup script** using the following command before compiling your source code: `export XILINXD_LICENSE_FILE=2100@flex.ece.cornell.edu` and `source /opt/xilinx/Xilinx_SDx_2017.1_sdx_0623_1/SDx/2017.1/settings64.sh`. If you run into problem while doing this, and you are using a fairly new laptop, try run the following commands: `export LC_CTYPE=en_US.UTF-8` and `export LC_ALL=en_US.UTF-8`, and then restart sourcing the setup scrip.

You are encouraged to check out the Vivado HLS user guide [?] for more detailed descriptions of the Vivado HLS synthesis flow.

3 Goal

The goal of this assignment is to create and optimize a CORDIC core that calculates the sine and cosine values of a given input angle. You will write the code in C++ for the CORDIC core, perform design space exploration using Vivado HLS, and explore trade-offs between area, performance, and accuracy.

The first part of this assignment is to write a functional CORDIC core using the double-precision floating-point type. With this baseline design, you will explore the design trade-offs by varying the iteration count of the main computation loop. Since CORDIC is an iterative algorithm, the number of iterations will affect the output accuracy as well as the performance of the synthesized hardware.

The second part is to use the fixed-point data type to optimize the CORDIC core for area, performance, and accuracy. The primary design space exploration goal is to understand how the bitwidth setting affects accuracy, as well as area and performance.

The third part asks you to maximize the throughput of the CORDIC core using optimization pragmas in Vivado HLS. This exercise will help you familiarize with common HLS optimizations and understand the effect of each optimization on the microarchitecture, performance, area, and timing of the design.

You will create a report describing the various trade-offs that you would make and how you maximize the throughput of the CORDIC core. For each design point (or architecture) you should provide its results including the area in terms of **resource utilization** (number of BRAMs, DSP48s, LUTs, and FFs), and performance in **throughput** in terms of number of CORDIC operations / second (i.e., number of input angles processed / second). The throughput can be calculated based on the reported interval (in clock cycles) and the target clock period (fixed to $10ns$ in this assignment).

4 Guidelines and Hints

4.1 Coding and Debugging

- The input arguments to the *cordic* function are typed *theta_type* and *cos_sin_type*. These are currently set as double-precision floating-point type (i.e., *double*)[1]. In the second part of this assignment, **you are expected to change them to a fixed-point type to optimize your design.** Please carefully consider the number of integer bits necessary for representing the range of required values. Your fixed-point design should be free of multiplication and division.
- Enter `source run_float.sh` under the project folder to compile and execute the floating-point program; Enter `source run_fixed.sh` to run the fixed-point implementation (where the $FIXED\_TYPE$ macro is defined).
- The test bench creates an *out.dat* file which is useful for debugging [2]. This file lists the golden sine/cosine values from *math.h*, the sine/cosine values computed from your function, and the normalized difference (error). You will be able to assess the correctness and/or accuracy of your code based on the error reported by the test bench. Note that the errors are expected to be close to but NOT exactly zero even with the correct code. The accuracy should be improved by increasing the number of iterations. Otherwise, your code is not working.
- There is a constant array called *cordic_ctab* in *cordic.h*. You may find this useful although you do not necessarily have to use it.
- Please include meaningful comments in your code.

4.2 Design Exploration

- In this assignment, you will use a fixed 10ns clock period targeting a specific Xilinx FPGA device. Clock period and target device have been specified in the $makefile$.
- The number of iterations in your *cordic* function will play an important role in the accuracy and performance of the design. You should explore this aspect with your floating-point design. You are encouraged to specify the iteration count in $run\_float.sh$ to run simulation. **You can get the important stats (i.e., accuracy, performance, and resource usage) from the Vivado HLS report** ***system_estimate.xtxt*** **file under the folder.**
- The data types of the variables in your *cordic* function would also make a significant difference in area, accuracy, and performance. This should be another form of your design space exploration. **For this part, the number of iterations is fixed to 20.** You should experiment extensively with the data types and your report should show how different data types affect the accuracy as well as area and performance. You are encouraged to specify the bitwidth settings in *cordic.h* to run simulation and synthesis. Similar to $run\_float.sh$, the script will also automatically generate important stats in

[1] You can either use a floating-point division `x/(double)(1ULL<<SHIFT_AMOUNT)` to perform a right shift on variable x of type *double*, or create a for loop to realize the shift in an iterative fashion `for (int i=0; i<SHIFT_AMOUNT; i++) {x=x*0.5;}`.

[2] You are welcome to use *gdb* as well.

the Vivado HLS reports *system_estimate.xtxt* file under the folder.

- In addition to adjusting the bitwidth of the fixed-point type, you should also experiment with the signedness of the data type (i.e., signed ap_fixed vs. unsigned ap_ufixed) and try to explain the results you obtained.
- Although synthesis takes some time to initialize after the $run\ run_float.sh$ command, it should finish within several minutes for each design point based on our past experience. It is not normal if Vivado HLS runs for more than 10 minutes. You can use the *top* command to check the real-time system usage to see if *ecelinux* is overloaded with other processes.

4.3 Performance Optimization

- In this section, we use a design with 20 iterations and 32-bit signed fixed-point type with 8 integer bits. **We fix the configuration in this part for convenience only. This configuration may not be most area-efficient.** The goal is to maximize the throughput of this design using optimization pragmas provided by Vivado HLS, you may use array partition, pipeline and unroll, **You are strongly encouraged to carefully study sections relevant to these pragmas** and modify the C++ source code to synthesize different microarchitectures.
- For this part of the lab, **please maximize the throughput of the CORDIC core using a minimum number of the optimization pragmas**.

4.4 Report

- Please write your report in a **single-column and single-space format with a 10pt font size. Page limit is 3, including necessary figures and tables.**
- 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 comparing and contrasting the various design points that you generated. **It is important to summarize the results of these design points in a table that clearly lists the design choices, resulting performance, area/resource allocation, and accuracy. It is also important to plot the results to demonstrate interesting observations and tradeoffs.[3]** Please avoid plotting a lot of random data without providing insights.[4]
- There should be a section describing how you maximize the throughput of the design. **It is important to compare the quality of results of your synthesized design with and without optimization pragmas in terms of performance and area.** A table will likewise be helpful in this case. Explain why you add each pragma. Be sure to discuss key metrics such as target partition factor, achieved partition factor,

[3]For example, error vs. number of iterations. Consider bar graph vs. line graph, or Log vs. linear scale on the axis.

[4]If it is difficult to explain, then you should rethink about what you are presenting, i.e., organize the data in a different way.

target initiation interval, achieved initiation interval, target unroll factor, and achieved unroll factor. Provide a conceptual comparison of the execution behavior and microarchitecture for the design with and without the optimization pragmas.

- 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.
- Showing small code snippets to demonstrate the restructuring is fine, but please avoid listing every single line of the code. It is possible to be succinct and thorough at the same time.
- 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 assignment on CMS. You are expected to submit your report and your code and scripts in a single zipped file named *cordic.zip* that contains the following contents:

- *report.pdf*: the project report in pdf.
- A folder named *solution*: the set of source files and scripts required to reproduce your experiments. Note that only these files should be submitted. Please run `make clean` to remove all the automatically generated output files.

6 Acknowledgement

This document is adapted from a project description originally developed by Prof. Ryan Kastner for CSE 237C at UCSD.

References

[1] P. Schaumont, *A Practical Introduction to Hardware/Software Codesign*, Springer, 2013.

[2] Xilinx Inc., *Vivado Design Suite User Guide: High-Level Synthesis UG902 (v2017.1)*, Available at http://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug902-vivado-high-level-synthesis.pdf