

Project 3: CORDIC

1. Introduction

COordinate **R**otation **D**igital **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, e.g., from the (x, y) coordinate. A CORDIC is often used in FPGA implementations, e.g., Vivado uses it for sine and cosine calculations, as it provides a low area implementation.

2. Materials

You are given a zip file with one folder named `cordic`. This contains the documents necessary to build the project.

- `cordic.cpp` → The place where you write your synthesizable code.
- `cordic.h` → header file with various definitions that may be useful for developing your code.
- `cordic_test.cpp` → test bench
- `script.tcl` and `directive.tcl` → These allow you to easily create a project. To do this, open the Vivado HLS Command Prompt tool (Start-->Xilinx Design Tools-->Vivado HLS-->Vivado HLS Command Prompt) and go to the directory where source files and script files reside by using `cd` command. Then type `vivado_hls script.tcl`. This will create a HLS project automatically, so you do not have to add source files, set the top function, select the target device, etc. This will create a folder called `hls` in that directory with the project. It will also synthesize the project.
- `project4-cordic.pdf` → project instructions: this file

3. Project Goal

The goal of this project is to create a CORDIC core that calculates the sine and cosine values of a given input angle. You will write all of the code for the CORDIC core, and perform design space exploration that trades off between area, performance and accuracy.

The first part of this project is to write a functional CORDIC core. The second part is to optimize that core for area, performance and accuracy. The primary design space exploration goal is to understand how accuracy affects both area and performance. For example, since the CORDIC is an iterative algorithm, the number of iterations will change your accuracy. But this could also affect your area and performance.

You will create a report describing the different tradeoffs that you made, and how you (code restructuring, pragmas utilized, etc.). For each architecture you should provide its results including the resource utilization (BRAMs, DSP48, LUT, FF), accuracy (as reported by the testbench), and performance in terms of throughput (number of FFT operations/second), latency, clock cycles, clock frequency (which is fixed to 10 ns). Since it is expected that you will do a significant amount of design space exploration around data types, you do not need to include an architecture for every different of data type. However, it should be obvious from your report

exactly how you changed the architecture to get the results that you are reporting. You should include an architecture for major optimizations and code restructuring.

4. Optimization Hints and Guidelines

- You must always use a clock period of 10 ns.
- You will be able to judge the “correctness” of your code based upon the error reported by the testbench. Note that this may not be 0 even with correct code, e.g., correct code that does not perform a large number of iterations will not be accurate.
- The testbench creates an `out.dat` file that is useful for debugging. This gives the golden sin/cos values (from `math.h`), the sin/cos values computed from your function, and the error. This file is put into the `hls/Launch` directory.
- The input arguments to the `cordic` function have types `theta_TYPE` and `cos_sin_TYPE`. These are initially set as `double` and you can change those in `cordic.h` if you wish.
- You should experiment extensively with the data types. I expect to see figures in your report that detail how different data types affect the accuracy (and area/throughput). These should be presented in a cogent manner; don’t just plot a bunch of data, but plot important data that makes sense. Hint: if it is hard to explain, then you should rethink about what you are presenting, i.e., organize the data in a different way.
- The number of iterations in your `cordic` function plays an important role in the accuracy, performance and area. One design space exploration should explore how this varies.
- The data types of the variables in your `cordic` function also can make a difference in area, accuracy and performance. This should be another form of design space exploration. You can vary the data types of the input values in the `cordic.h` file. Your “internal” variables in the `cordic` also have a role in this.
- It is possible to write the `cordic` function in less than 15 lines of code. This is not necessary to do so, but the actual implementation is not that difficult. Therefore, a good design space exploration, and more importantly a good report detailing this, is key to doing well on this project.
- Your report should describe your code at a high level. Since each of you will likely write different code anyone that reads your report should be able to fully understand what each line of your code does. However, this does not mean that you need to explain each line of code in excruciating detail. It is possible to be succinct and thorough at the same time.
- The `script.tcl` file adds the `-DBIT_ACCURATE` flag to synthesis and execution. Therefore, if you use `cordic.h` to set your data types (and this is a good idea), you need to define these in the `BIT_ACCURATE` define section.
- There is a constant array called `cordic_ctab` in `cordic.h`. You may find this useful though you do not have to use it. That is all that will be said about that.
- Comment your code.

5. Submission Procedure

You must submit your code (and only your code, not project files). Your code should have everything in it so that we can synthesize it directly. This means that you should use pragmas in your code, and not use the GUI to insert optimization directives. Each folder should have the necessary files to synthesize and test the code, e.g., `cordic_test.cpp`, `cordic.h`, etc.

The folder should be zipped with a similar file structure as below:

CORDIC_lab_YOUR_LAST_NAME.zip

Contents:

- “report.pdf” – A document describing all of your optimizations.
- Folder “architecture1”: the set of files required to synthesize your first architecture
- Folder “architecture2”: the set of files required to synthesize your second architecture
- ...
- Folder “architectureN”: the set of files required to synthesize your Nth architecture

Email this to Janarbek Matai <jmatai@cs.ucsd.edu> with the title “CSE 237C Project 4”.