

## 1 Introduction

In this lab, you will be implementing the CORDIC algorithm using Vivado HLS.

### 1.1 Introduction to CORDIC

CORDIC is an acronym for Coordinate Rotation Digital Computer introduced by Volder in 1959. Originally intended for the computation of trigonometric functions, this simple and efficient algorithm was later generalized to include hyperbolic functions, multiplication, and division. This iterative technique computes multiplicative and similar functions by shift and add operations. It is commonly used in FPGAs where no hardware multiplier is present.

If we were to have a brief mathematical overview of the method:  
Updating 3 variables, say  $x$ ,  $y$ , and  $z$ , in the following manner,

$$x_{k+1} = x_k - d_k 2^{-k} y_k$$

$$y_{k+1} = y_k + d_k 2^{-k} x_k$$

$$z_{k+1} = z_k - d_k \tan^{-1}(2^{-k}),$$

$$\text{where } d_k = \text{sign}(z_k) = \begin{cases} -1, & \text{if } z_k < 0 \\ +1, & \text{if } z_k \geq 0 \end{cases}.$$

After a sufficient number of iterations,  $x$ ,  $y$  and  $z$  converge to:

$$x_k = G(x_0 \cos z_0 - y_0 \sin z_0)$$

$$y_k = G(x_0 \sin z_0 - y_0 \cos z_0)$$

$$z_k = 0,$$

where  $G=1.64676$ .

If we choose  $x_0$  and  $y_0$  properly, we can infer the value of  $\cos(z_0)$  and  $\sin(z_0)$  from  $x_k$  and  $y_k$ .  
For example, if we choose  $x_0 = \frac{1}{G}$ ,  $y_0 = 0$ , we have  $x_k = \cos(z_0)$ ,  $y_k = \sin(z_0)$ .

Similarly, when we set  $d_k$  as  $d_k = -\text{sign}(y_k)$

After a sufficient number of iterations,  $x$ ,  $y$  and  $z$  converge to:

$$x_k = G \sqrt{x_0^2 + y_0^2}$$

$$y_k = 0$$

$$z_k = z_0 + \tan^{-1}\left(\frac{y_0}{x_0}\right)$$

If we choose  $x_0$ ,  $y_0$  and  $z_0$  properly, we can infer the  $\tan^{-1}$  function.

If we implement this algorithm in hardware, the schematic of the design would be the one shown in Figure 1.

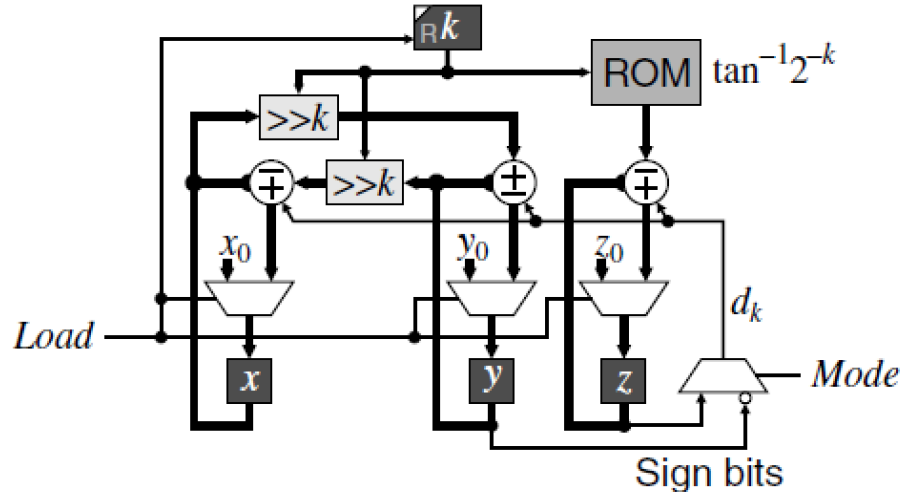


Figure 1: CORDIC Module Schematic.

## 2 Brief Instructions

In this section, we will implement the hardware design of the CORDIC module using Vivado HLS. Please download “**Lab1\_code.zip**” from Piazza and **extract them** before you proceed. In the extracted folders, you will find three files named as “**basic\_cordic.h**”, “**basic\_cordic.cpp**”, and “**basic\_cordic\_tb.cpp**”.

Launch Vivado HLS, add the three files above to the project, and implement your design according to Section 1.1. For the detailed steps, please find Section 3 for reference.

Please make sure the functionalities of your design (basic\_cordic.cpp) meet the following requirements:

- 1) The *cordic\_sin\_cos* module calculates *sin* and *cos* with initial input  $x_0$ ,  $y_0$ , and  $z_0$ .
- 2) The *cordic\_arctan* module calculates  $\tan^{-1}$  with input sine and cosine values. You need to specify  $x_0$ ,  $y_0$  and  $z_0$  with given inputs.
- 3) For the calculation, please use a 16-bit signed fixed-point number with 12 bits for the fraction.
- 4) Please make sure the errors of your results are less than 0.05.

Please complete the blank part in **basic\_cordic.cpp** and use the provided testbench (**basic\_cordic\_tb.cpp**) to check the correctness.

In the testbench, *test\_sin\_cos* tests your implementation of the *cordic\_sin\_cos* module, and *test\_arctan* checks the correctness of the *cordic\_arctan* module.

- 1) In *test\_sin\_cos*,  $x_0$ ,  $y_0$  and  $z_0$  are set as 1, 0 and  $\theta$ , where  $\theta$  is the random input angle in the range  $[-\pi, \pi]$ .
- 2) In *test\_arctan*, 10 angles are used to check the correctness of your code. If your implementation

is correct, the output of the *cordic\_arctan* module should be almost the same as the provided angles.

Once you pass the testbench, we will do the behavior simulation to verify the functionalities of our design.

### 3 Implementation of CORDIC circuit using Vivado HLS

In this lab, we will implement the hardware design of the CORDIC module using Vivado HLS and use Vivado HLS to convert C/C++ code to Verilog without directly implementing it in Verilog. Please go through the following steps.

#### 3.1 Create a new project in HLS

In this subsection, we will learn how to create a new project and add source files in Vivado HLS.

- a) Please download “**Lab1\_code.zip**” from Piazza and **extract them** before you proceed. In the extracted folders, you will find three files named as “**basic\_cordic.h**”, “**basic\_cordic.cpp**”, and “**basic\_cordic\_tb.cpp**”, and another three files named as “**hyperbolic\_cordic.h**”, “**hyperbolic\_cordic.cpp**”, and “**hyperbolic\_cordic\_tb.cpp**”.
- b) Launch **Vivado HLS 2018.3**. Click on “Create New Project” to create a new project. Enter the project name with your preference on the welcome page, as shown in Figure 2. Then, click on “**Next**”.
- c) In the next window, add “basic\_cordic.cpp” and “basic\_cordic.h” in the project by clicking on “**Add Files...**”. Then click on select *cordic\_sin\_cos* as the top function, (Note that we will switch to *cordic\_arctan* later for testing implementation of your arctan.) and click on “**Next**”. Please refer to Figure 3 on this step.
- d) Add “basic\_cordic\_tb.cpp” in the Testbench Files and click on “**Next**”.
- e) Click on “...” in the Part Selection block, choose “**Boards**”, select “**Zybo-Z7-10**” and click on “**OK**” as shown in Figure 4. If you don’t find “**Zybo-Z7-10**” as an option, please refer to the Lab 0 manual. Choose “**OK**” and “**Finish**” after you are done.
- f) After the project is successfully created, you should be able to see the source code and testbench on the left column. Please find Figure 5 for reference.

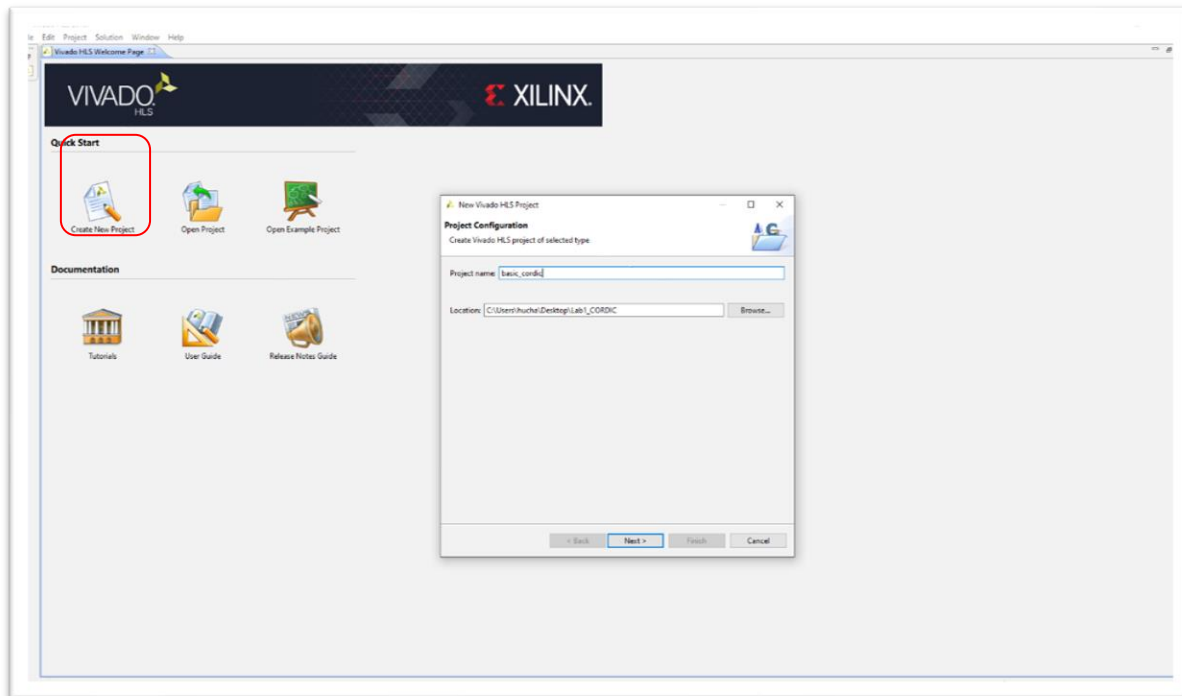


Figure 2: Create a new project in Vivado HLS.

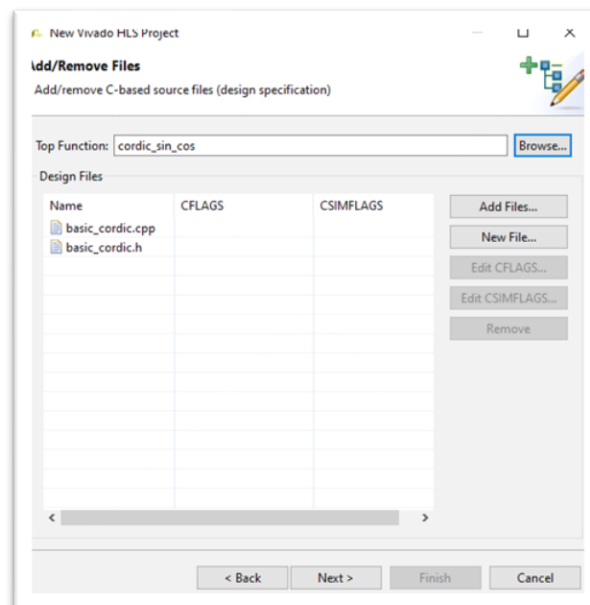


Figure 3: Add the source code.

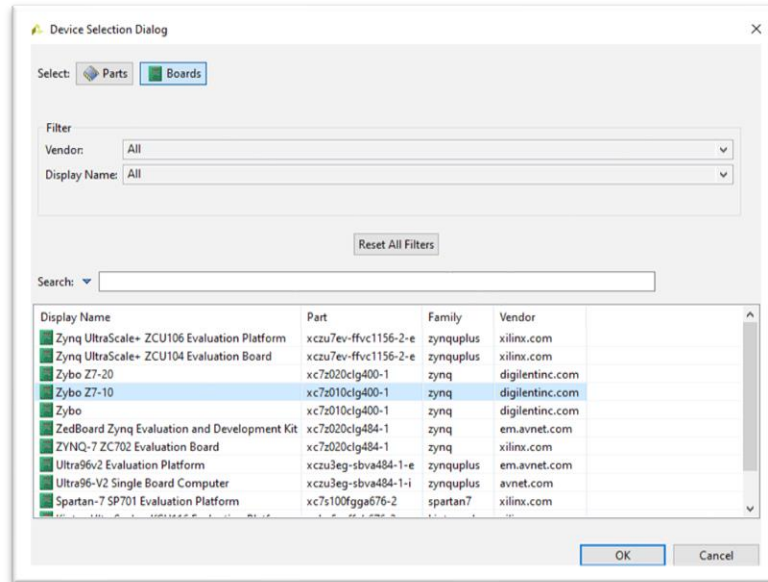


Figure 4: Board selection.

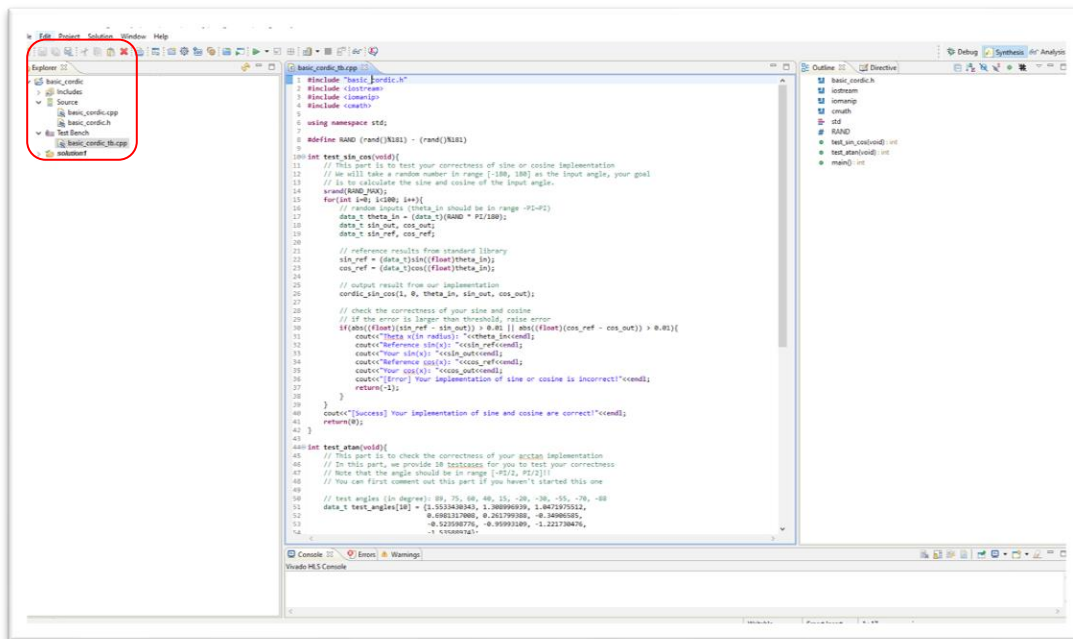


Figure 5: Project is successfully created.

Now the project should be properly configured for you to work on. We will continue to work on this project in the next section.

## 3.2 Run C++ verification

In this section, we will show how to verify the correctness of your implementation in Vivado HLS by running C Simulation.

- On the main page, click on **“Project -> Run C Simulation”**. Check the box of **“Clean Build”**,

and then click “OK”, as shown in Figure 6.

- b) Wait until the simulation is done. If your implementations are all correct, you should see “[Success] Your implementation of xxx is correct!” in the console. Please refer to Figure 7.

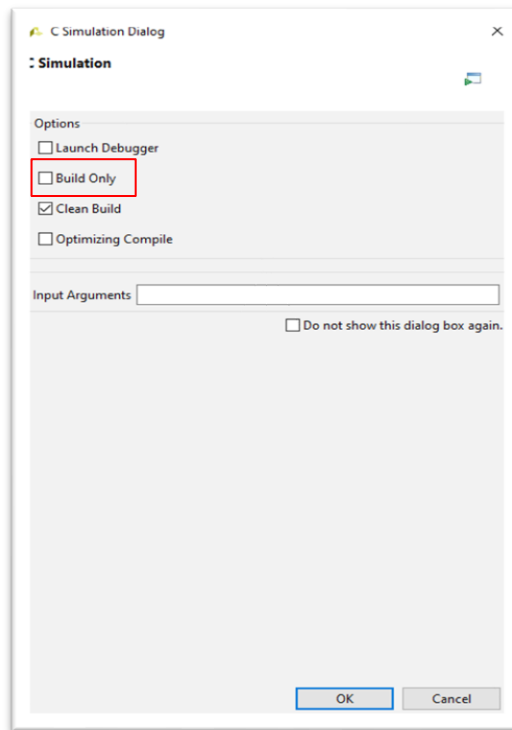


Figure 6: C simulation configuration.

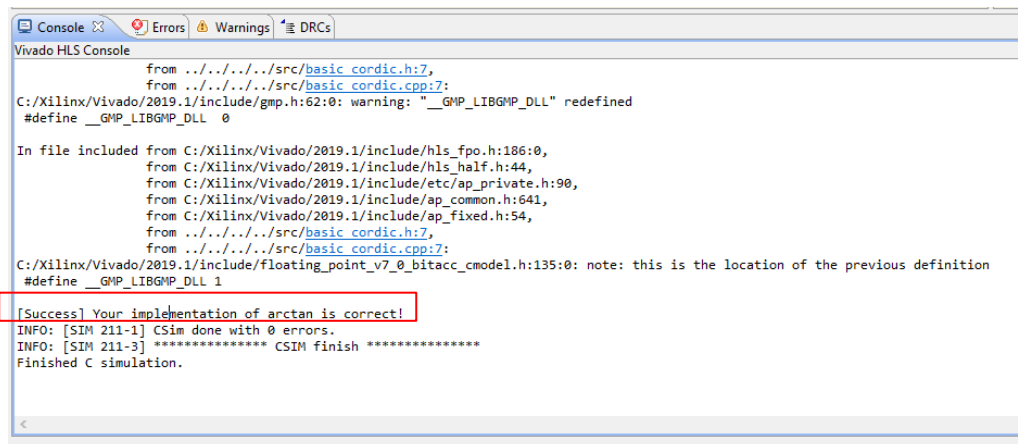


Figure 7: C++ verification done.

Now we are ready to synthesize our C++ implementation to RTL! Please go through the following steps in Section 3.3.

## 3.3 Steps for C/RTL Co-simulation

Once you pass the C++ verification, we can start synthesizing the code to hardware language (RTL).

- Before we start the C++ Synthesis, the top function should be specified for HLS. Click on **“Project -> Project Settings”** and choose **“Synthesis”** on the left column. Browse the target function you want to synthesize to the RTL module. For example, for synthesizing your `sin_cos` function, please choose the corresponding `cordic_sin_cos` as the top function, as shown in Figure 8.
- Select **“Solution -> Run C Synthesis -> Active Solution”**. HLS will automatically launch the process and start to synthesize your code to RTL implementation. If everything works well, you should see a *Synthesis Report* pops up.

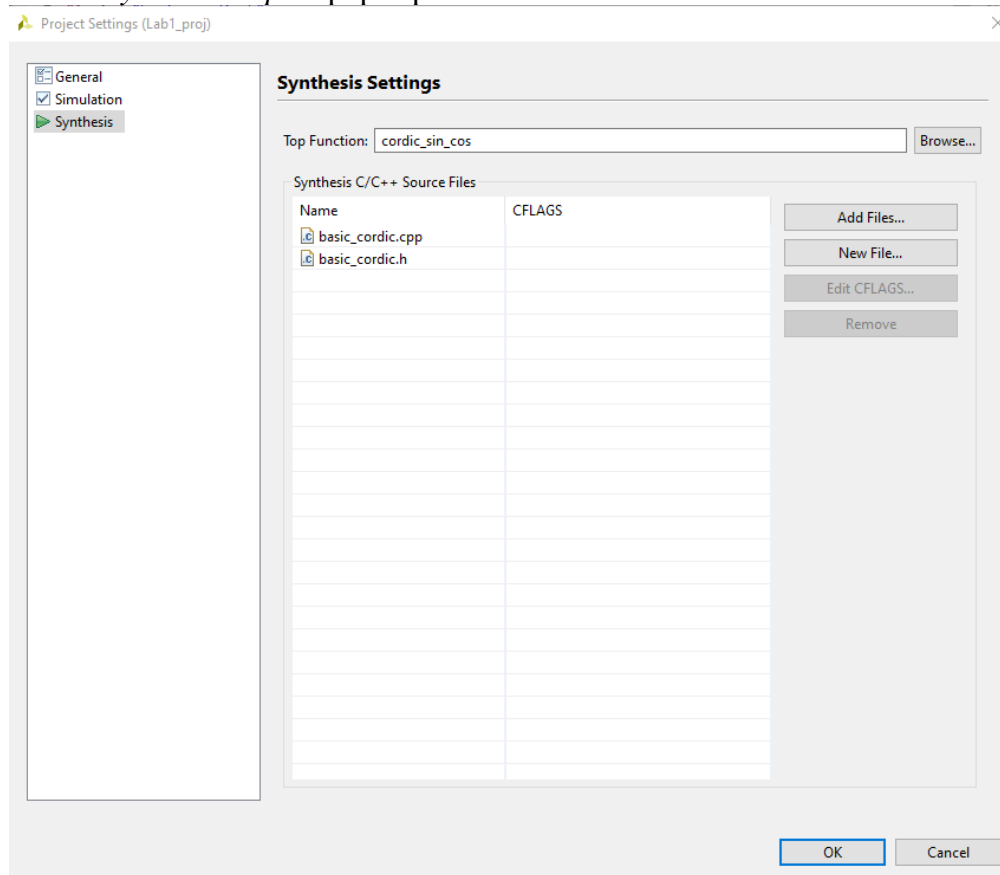



Figure 8: Select the top function.

- Now we are ready to perform the behavior simulation of the generated RTL implementation. Please select **“Solution -> Run C/RTL Co-simulation”**. Set the simulator to **Vivado Simulator**, and make sure to set **“all”** for Dump Trace as shown in Figure 9.
- Once the co-simulation is done, you should find a *Cosimulation Report* that pops up to the window. Make sure the status of the Verilog result is *Pass*!
- Now the behavior simulation of your module is ready. Please find **“Open Wave Viewer...”** on

the top bar (  ) and click it. If the icon does not appear on the bar, go back to step c and make sure the configuration is correct.

f) Vivado will then launch its simulators after a few minutes. Your results should be the same as shown in Figure 10 and Figure 11. Note that we can convert the hexadecimal results to decimal values. Right click on the value and choose “**Radix -> Real Settings**”. Set the configuration to be the same as Figure 12.

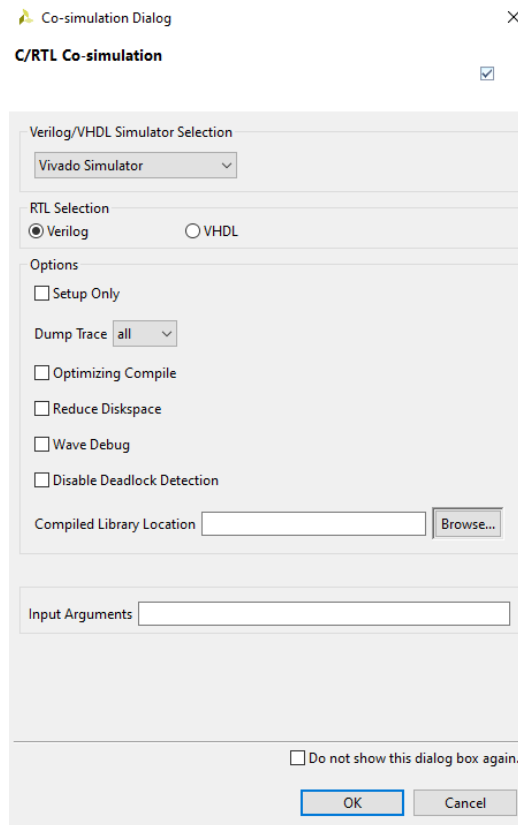


Figure 9: Configuration for C/RTL Co-simulation.

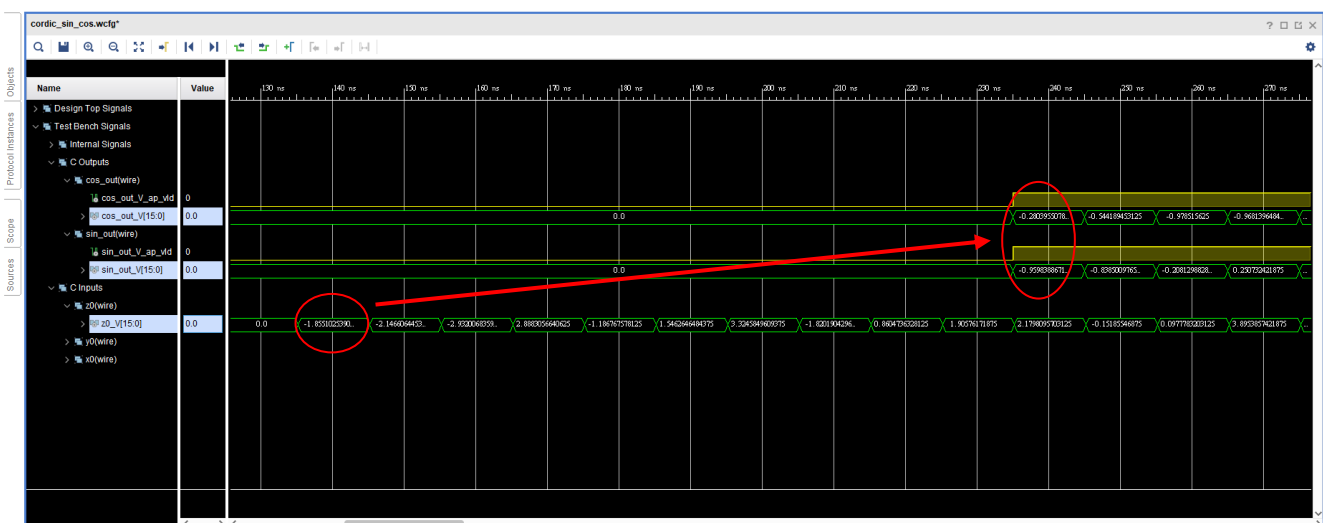


Figure 10: Behavior simulation of CORDIC-based sine and cosine.



# ECEN 489/689 – Lab 1: CORDIC Circuit Design using Vivado HLS

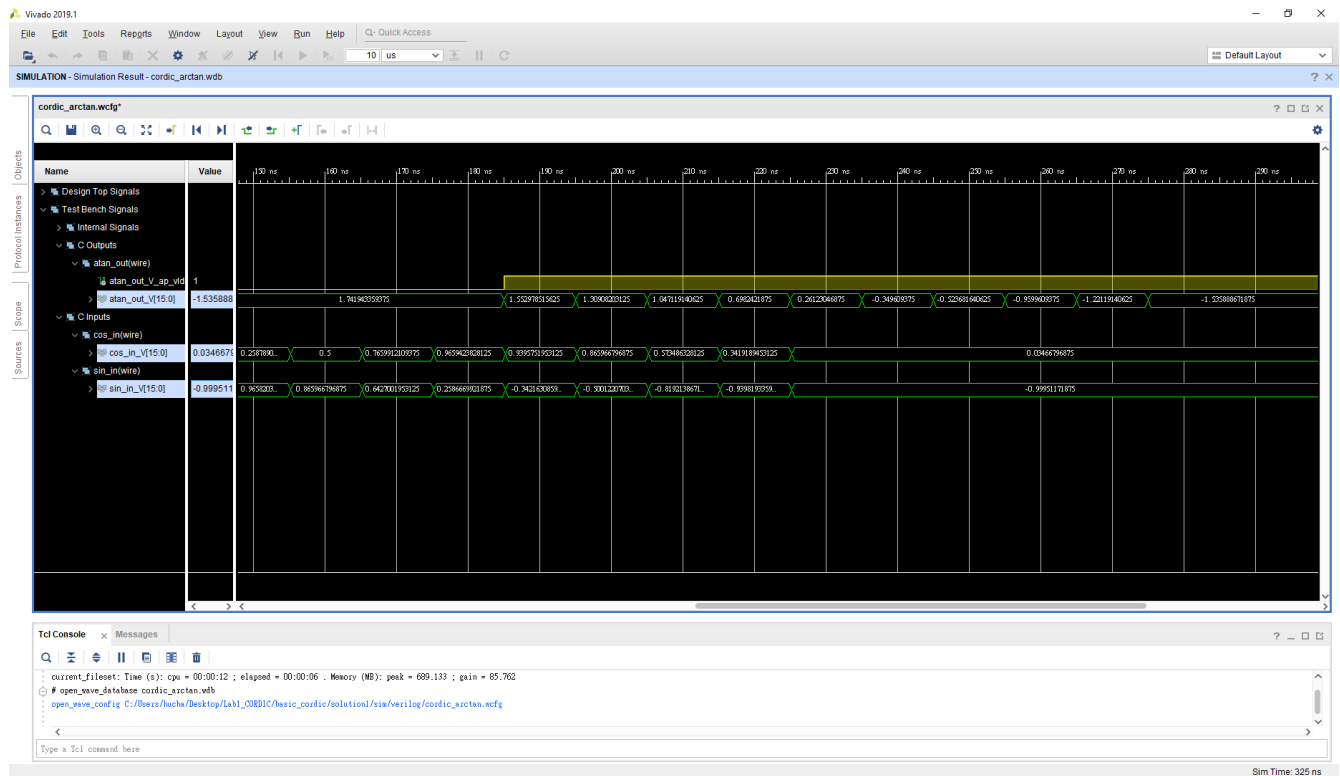


Figure 11: Behavior simulation of CORDIC-based  $\tan^{-1}$ .

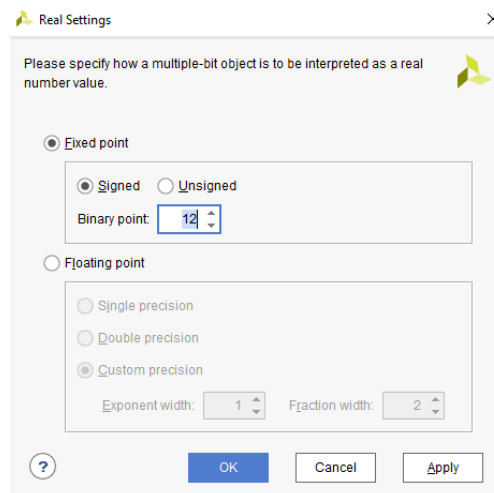


Figure 12: Convert hexadecimal to decimal values.

## 4 Pre-lab submission

1. Please only submit 1 PDF file, containing the following items:
  - a. Screenshots of your behavior simulation of CORDIC-based sine and cosine (the one like

Figure 10).

- b. A few words explaining the results in the screenshots.
2. Please name the PDF file as “Lab#\_PreLab\_Section#\_LastName\_FirstName.pdf”.
3. Please email the PDF file to the TA with the subject as “ECEN 489 Lab Report”.

### 5 Post-lab Submission

1. Please only submit 1 zip file, containing all the code files you used in this lab, and 1 PDF report. Name it as “Lab#\_PostLab\_Section#\_LastName\_FirstName.zip”.
2. In the PDF report, please include the following items:
  - a. Screenshots of the behavioral simulations for CORDIC-based  $\tan^{-1}$ .
  - b. A few words explaining the results in the screenshots.
3. Please email the zip file to the TA with the subject as “ECEN 489 Lab Report”.