**Project 4 – Xilinx Vitis HLS CORDIC Module**

**Report for ELEC 522**

**College of Engineering and Computer Science**

**Rice University**

**Houston, TX**

**7th November, ‘2022**

**Karan Venaik**

**I. INTRODUCTION/PROJECT DESCRIPTION**

The objective of the project is to learn and execute the Xilinx System Generator Design flow of C to FPGA-compatible HDL generation and use the Zynq SoC Arm core to control the FPGA accelerator. This is achieved through the design of an optimized 16b, signed, fixed-point CORDIC circular module that calculates and returns the sine/cosine of a given angle. This CORDIC module will subsequently be used in the next project for QR Decomposition.

**II. THEORETICAL BACKGROUND**

**CORDIC Algorithm**

For the purposes of this project, the CORDIC algorithm is a technique to efficiently calculate the sines/cosines of a given angle ‘ɸ’. The algorithm starts by taking a unit vector with angle 0 (r, θ = 1, 0). The unit vector is rotated by some angle and compared against the required angle ‘ɸ’. If the updated angle ‘θ’ is less than the required angle ‘ɸ’, the rotation angle is halved, and the unit vector is rotated further. In contrast, if the angle ‘θ’ is greater than the required angle ‘ɸ’, the rotation angle is halved, and the unit vector is rotated in the opposite direction. The CORDIC algorithm is like the Binary Search algorithm where the search interval is repeatedly halved till the element is found. In the CORDIC algorithm, the angle of rotation is halved and appended to the angle ‘θ’ till it converges to the required angle ‘ɸ’. The cartesian coordinates of the resultant vector is the sine and cosine of the required angle.

Diagram

Description automatically generated

**Figure #1:** Graphical representation of unit vector rotation to reach target angle and subsequently derive the sine and cosine of target angle. The rotation angles decrease in value and gradually converge to the target over multiple iterations. (from Kastner, Ch. #3, Pg. #59)

The rotation can be accomplished by simple matrix multiplication shown as follows where ‘⍺’ is the rotation angle, x’/y’ are the rotated coordinates, and x/y are the original coordinates.

To simplify the matrix, we can convert the sine and cosine functions to their tangent equivalent.

Substituting the first format allows us to calculate the rotated vector solely in the form of tangents of the rotational angle. This approach is advantageous since we only need to store the tangents of the angle if we use lookup tables (as contrasted with the former expansion which implies, we must store cosines and tangents for every rotational angle.)

This result creates a new obstacle where calculating the rotated vector requires calculation of tangent of the rotation angle. If we create a method to calculate the tangent, we might as well directly calculate the sines/cosines. Additionally, other methods of calculating the tangent (look up tables, interpolation, etc.) require similar computation power which brings us back to square one.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Index | Tangent | | | 2^(-Index) | | |
| Angle (Degrees) | Angle (Radians) | Value | Value | Arctan (Radians) | Arctan (Degrees) |
| 0 | 90 | 1.570796327 | 1.63246E+16 | 1 | 0.785398163 | 45 |
| 1 | 45 | 0.785398163 | 1 | 0.5 | 0.463647609 | 26.56505118 |
| 2 | 22.5 | 0.392699082 | 0.414213562 | 0.25 | 0.244978663 | 14.03624347 |
| 3 | 11.25 | 0.196349541 | 0.198912367 | 0.125 | 0.124354995 | 7.125016349 |
| 4 | 5.625 | 0.09817477 | 0.098491403 | 0.0625 | 0.06241881 | 3.576334375 |
| 5 | 2.8125 | 0.049087385 | 0.04912685 | 0.03125 | 0.031239833 | 1.789910608 |
| 6 | 1.40625 | 0.024543693 | 0.024548622 | 0.015625 | 0.015623729 | 0.89517371 |
| 7 | 0.703125 | 0.012271846 | 0.012272462 | 0.0078125 | 0.007812341 | 0.447614171 |
| 8 | 0.3515625 | 0.006135923 | 0.006136 | 0.00390625 | 0.00390623 | 0.2238105 |
| 9 | 0.17578125 | 0.003067962 | 0.003067971 | 0.001953125 | 0.001953123 | 0.111905677 |

Table #1 shows iteration numbers, angles, and corresponding angles. As required by the algorithm, the angle must be reduced every iteration. However, it is not necessary for the rotation to be exactly halved every iteration to converge over time. In an effort to reduce computation, we could replace the tangent with another decreasing series. Thus, instead of pre-computing tangents of all rotational angles and creating a lookup table, we can multiply the original vectors with 2-N, since that amounts to simple right shifts. Table #1 also shows the decreasing series and the corresponding angles they rotate when supplied as input elements.

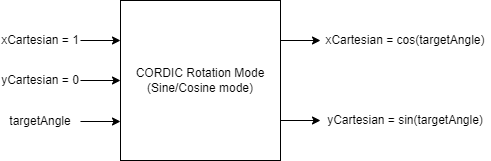
The conclusion obtained from the previous section can be used to design a CORDIC rotation module that exploits the right-shift behaviour to perform rotations efficiently and in an iterative manner. The equation for one rotation can be written as follows by substituting the tangent terms with right-shift/powers of two.

The above equation only gives the vector after a singular rotation. However, to obtain an accurate rotation of the input vector, multiple vector rotations must be conducted. The equivalent final rotation can be represented as,

Since the leading inverse square root term is dependent only on ‘n’ iterations, it can be simplified to a general expression and precomputed before the solution is deployed. Since the leading term is not multiplied in each iteration of the rotation, the rotated vector is “scaled” during the rotation. The general scaling factor is given below,

**Rotation Mode**

In “Rotation Mode”, the CORDIC module computes the sine and cosine of a given target angle. This is achieved by rotating a unit vector aligned with the “X”-axis (angle initially at zero) to the given target angle. The X- and Y-components of the final vector after rotation is completed are the cosine and sine of the target angle respectively (correct scaling is assumed). We can consider a black box as shown in fig. #2 to show the input-output relation of the CORDIC module in Rotation mode.

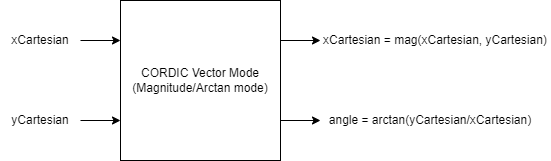


**Figure #2:** Black box representation of CORDIC Rotation Module (Sine/Cosine Mode) with the inputs streaming in from the left and the outputs streaming out to the right. The inputs (xCartesian, yCartesian) is set to (1, 0) for correct computation regardless of the user provided targetAngle. After rotation iterations are completed, outputs (xCartesian, yCartesian) transform to (cos(targetAngle), sin(targetAngle)).

Even though the CORDIC module has a unit vector as an input, the design must allow for different (xCartesian, yCartesian) values for a more generalized rotation with input target angle.

**Vector Mode**

In “Vector Mode”, the CORDIC module computes the arctangent and magnitude of a given vector. This is achieved by rotating a given vector to minimize the Y-component to zero (align the given vector with the “X”-axis). The X-component of the final vector after rotation is completed is the magnitude of the input vector. Similarly, the angle produced is arctan (yCartesian/xCartesian). We can consider a black box as shown in fig. #3 to show the input-output relation of the CORDIC module in Vector mode.



**Figure #3:** Black box representation of CORDIC Vector Module (Magnitude/Arctan Mode) with the inputs streaming in from the left and the outputs streaming out to the right. The inputs (xCartesian, yCartesian) are userset. After rotation iterations are completed, outputs (xCartesian, angle) transform to (mag(xCartesian, yCartesian), arctan(yCartesian/xCartesian)).

**III. CODE DESIGN**

**Vitis HLS**

The Vitis HLS design section comprises of three files:

1. cordic.cpp
2. cordic.h
3. cordicTest.cpp

Majority of the design is written in the cordic.cpp module which allows the end-user to select between Rotation and Vector mode to calculate the sine/cosine or the magnitude/arctan with given input data. The input parameters are shown in table. #2 and table. #3.

|  |  |  |
| --- | --- | --- |
| Data Type | Parameter Name | Rotation Mode |
| Boolean | cordicMode | FALSE |
| ap\_fixed<16, 3> | targetAnglePhi | Radian angle for sine/cosine calculation |
| ap\_fixed<16, 3> | xCartesian | Length of Unit Vector |
| ap\_fixed<16, 3> | yCartesian | Set to 0 for correct sine/cosine calculation |
| ap\_fixed<16, 3> | &arctan | Unused (Junk/Can be ignored) |
| ap\_fixed<16, 3> | &s | Calculated sin(targetAnglePhi) |
| ap\_fixed<16, 3> | &c | Calculated cos(targetAnglePhi) |

**Table #2:** Input/Output Parameters for Vitis HLS CORDIC Module in Rotation Mode

|  |  |  |
| --- | --- | --- |
| Data Type | Parameter Name | Vector Mode |
| Boolean | cordicMode | TRUE |
| ap\_fixed<16, 3> | targetAnglePhi | Unused (set to 0) |
| ap\_fixed<16, 3> | xCartesian | Cartesian X-coordinate to calculate arctan(y/x) |
| ap\_fixed<16, 3> | yCartesian | Cartesian Y-coordinate to calculate arctan(y/x) |
| ap\_fixed<16, 3> | &arctan | Calculated arctan(y/x) |
| ap\_fixed<16, 3> | &s | Unused (Junk/Can be ignored) |
| ap\_fixed<16, 3> | &c | Calculated magnitude(x, y) |

**Table #3:** Input/Output Parameters for Vitis HLS CORDIC Module in Vector Mode

To successfully achieve rotation of the vectors and keep track of the angle the vector is rotated through, the module contains three angles,

1. targetAnglePhi – The target angle for which we must calculate sine and cosine. We want to reach/"binary search" for this angle iteratively through successively smaller rotations
2. cumulativeAngleTheta – The angle by which we are rotating every iteration. Angle for a given iteration "i" can be computed by arctan(2^-i).
3. currentRotationAngleAlpha – The cumulativeAngle we have obtained using iterative rotations. We aim to make this equivalent/very close to the target angle Phi.

The rotation direction is decided using minimization functions that change depending on the CORDIC mode. As described earlier, the rotation mode tries to minimize the angle between the unit vector and the target angle. Conversely, the vector mode minimizes the Y-coordinate to align the input vector with the “x”-axis. The two minimization functions are as follows,

1. Rotation mode – minimizationFunction = cumulativeAngleTheta - targetAnglePhi
2. Vector mode – minimizationFunction = yCartesian

The final scaling is completed using hardwired right shifts and accumulates as described in the lecture with Dr. Cavallaro to avoid using multipliers. The scaled values are passed back to the test bench using the pointers passed in the function definition.

The test bench (cordicTest.cpp) allows for the user to test both modes by calling the CORDIC module (void cordic(…)) in the cordic.cpp file. Tables. #4 and #5 show the test input data.

|  |  |  |
| --- | --- | --- |
| Data Type | Parameter Name | Input Value |
| Boolean | cordicMode | FALSE |
| ap\_fixed<16, 3> | targetAnglePhi | Radian Angle swept between (0, 1.57) |
| ap\_fixed<16, 3> | xCartesian | 1 |
| ap\_fixed<16, 3> | yCartesian | 0 |

**Table #4:** Input parameters to test Vitis HLS CORDIC Module in Rotation Mode

|  |  |  |
| --- | --- | --- |
| Data Type | Parameter Name | Vector Mode |
| Boolean | cordicMode | TRUE |
| ap\_fixed<16, 3> | targetAnglePhi | 0 |
| ap\_fixed<16, 3> | xCartesian | X-cartesian = (0, 2, 0.1) |
| ap\_fixed<16, 3> | yCartesian | Y-cartesian = (0, 2, 0.1) |

**Table #5:** Input parameters to test Vitis HLS CORDIC Module in Vector Mode

**IV. RESULTS**

**Vitis HLS**

**Code Listing for Cordic.cpp**

***#include*** *"cordic.h"*

***#include*** *<iostream>*

***using******namespace*** *std;*

*//Function Name: cordic() - CORDIC block that calculates the sine/cosine of an input angle (rotation mode) or the magnitude/arctangent of an input vector (vector mode)*

*//bool cordicMode: Selects between "Rotation Mode" (false/0/low), or "Vector Mode" (true/1/high)*

*//*

*//THETA\_TYPE targetAnglePhi:*

*//(i) "Rotation Mode" - The angle (radian) who's sine and cosine the CORDIC module calculates.*

*//(ii) "Vector Mode" - Unused (must be set to 0 (zero)).*

*//*

*//COS\_SIN\_TYPE xCartesian:*

*//(i) "Rotation Mode" - The length of unit vector as aligned with X-axis or the Cartesian X-coordinate (must be set to 1 (one)).*

*//(ii) "Vector Mode" - Cartesian X-coordinate to calculate arctan(y/x) and sqrt(x^2 + y^2).*

*//*

*//COS\_SIN\_TYPE yCartesian:*

*//(i) "Rotation Mode" - The length of unit vector as aligned with Y-axis or the Cartesian Y-coordinate (must be set to 0 (zero)).*

*//(ii) "Vector Mode" - Cartesian Y-coordinate to calculate arctan(y/x) and sqrt(x^2 + y^2).*

*//*

*//COS\_SIN\_TYPE &arctan:*

*//(i) "Rotation Mode" - Unused (Junk/Can be ignored).*

*//(ii) "Vector Mode" - Value of calculated arctan(y/x) as available to the test function.*

*//*

*//COS\_SIN\_TYPE &s:*

*//(i) "Rotation Mode" - Value of calculated sine(targetAnglePhi) as available to the test function.*

*//(ii) "Vector Mode" - Unused (Junk/Can be ignored).*

*//*

*//COS\_SIN\_TYPE &c:*

*//(i) "Rotation Mode" - Value of calculated cosine(targetAnglePhi) as available to the test function.*

*//(ii) "Vector Mode" - Value of calculated sqrt(x^2 + y^2) as available to the test function.*

***void******cordic****(****bool*** *cordicMode,* ***THETA\_TYPE*** *targetAnglePhi,* ***COS\_SIN\_TYPE*** *xCartesian,* ***COS\_SIN\_TYPE*** *yCartesian,* ***COS\_SIN\_TYPE*** *&arctan,* ***COS\_SIN\_TYPE*** *&s,* ***COS\_SIN\_TYPE*** *&c)*

*{*

***#pragma*** *HLS INTERFACE mode=s\_axilite port=****return***

***#pragma*** *HLS INTERFACE mode=s\_axilite port=c*

***#pragma*** *HLS INTERFACE mode=s\_axilite port=s*

***#pragma*** *HLS INTERFACE mode=s\_axilite port=arctan*

***#pragma*** *HLS INTERFACE mode=s\_axilite port=yCartesian*

***#pragma*** *HLS INTERFACE mode=s\_axilite port=xCartesian*

***#pragma*** *HLS INTERFACE mode=s\_axilite port=targetAnglePhi*

***#pragma*** *HLS INTERFACE mode=s\_axilite port=cordicMode*

*//In "Rotation Mode", there are three angles, namely*

*//(i) Phi - The target angle for which we must calculate sine and cosine. We want to reach/"binary search" for this angle iteratively through successively smaller rotations*

*//(ii) Alpha - The angle by which we are rotating every iteration. Angle for a given iteration "i" can be computed by arctan(2^-i).*

*//(iii) Theta - The cumulativeAngle we have obtained using iterative rotations. We aim to make this equivalent/very close to the target angle Phi.*

***THETA\_TYPE*** *cumulativeAngleTheta = 0;*

***THETA\_TYPE*** *currentRotationAngleAlpha = 0;*

*//Minimization function allows us to direct the rotation for each iteration. We can change minimization function as necessary.*

*// "Rotation Mode" - Difference between cumulativeAngleTheta and the targetAnglePhi (How far away are we from the angle we need to be at?)*

*// "Vector Mode" - The Y component of vector to be rotated/Cartesian Y-coordinate*

***THETA\_TYPE*** *minimizationFunction = 0;*

***if****(!cordicMode)*

*{*

*minimizationFunction = cumulativeAngleTheta - targetAnglePhi;*

*}*

***else***

*{*

*minimizationFunction = yCartesian;*

*}*

*//Total number of iterations conducted to reach convergence*

***int*** *totalIterations = 20;*

***#pragma*** *HLS PIPELINE II = 1*

*//Starting rotation iteration process*

***for*** *(****int*** *currentIteration = 0; currentIteration < totalIterations; currentIteration++)*

*{*

*//Right Shifting the cartesian X- and Y-coordinates to provide efficient rotation instead of multiplication*

***COS\_SIN\_TYPE*** *cos\_shift = xCartesian >> currentIteration;*

***COS\_SIN\_TYPE*** *sin\_shift = yCartesian >> currentIteration;*

*//Extract the rotation angle corresponding to the iteration from the precomputed/provided lookup table*

*currentRotationAngleAlpha = rotationAngles[currentIteration];*

*//Which way do we rotate?*

*//If the minimizationFunction is -ve,*

*//(i) "Rotation Mode" - Counterclockwise*

*//(ii) "Vector Mode" - Clockwise*

***if****(minimizationFunction < 0)*

*{*

*xCartesian = xCartesian - sin\_shift;*

*yCartesian = yCartesian + cos\_shift;*

***if*** *(!cordicMode)*

*{*

*//Update cumulativeAngleTheta that has been covered/rotated through*

*//Update Minimization function since we are getting closer*

*cumulativeAngleTheta = cumulativeAngleTheta + currentRotationAngleAlpha;*

*minimizationFunction = cumulativeAngleTheta - targetAnglePhi;*

*}*

***else******if*** *(cordicMode)*

*{*

*//Update minimizationFunction*

*cumulativeAngleTheta = cumulativeAngleTheta - currentRotationAngleAlpha;*

*minimizationFunction = yCartesian;*

*}*

*}*

*//Which way do we rotate?*

*//If the minimizationFunction is +ve,*

*//(i) "Rotation Mode" - Clockwise*

*//(ii) "Vector Mode" - Counterclockwise*

***else***

*{*

*xCartesian = xCartesian + sin\_shift;*

*yCartesian = yCartesian - cos\_shift;*

***if*** *(!cordicMode)*

*{*

*//Do SinCos*

*cumulativeAngleTheta = cumulativeAngleTheta - currentRotationAngleAlpha;*

*minimizationFunction = cumulativeAngleTheta - targetAnglePhi;*

*}*

***else******if*** *(cordicMode)*

*{*

*//Do arctan*

*cumulativeAngleTheta = cumulativeAngleTheta + currentRotationAngleAlpha;*

*minimizationFunction = yCartesian;*

*}*

*}*

*}*

*//Scaling Factor correction implementation as covered in lecture on 11/10*

*//Manually unrolled to ensure pipelining*

***COS\_SIN\_TYPE*** *xCartesianTemp = xCartesian;*

***COS\_SIN\_TYPE*** *yCartesianTemp = yCartesian;*

*xCartesianTemp = xCartesian - (xCartesian >> 1);*

*yCartesianTemp = yCartesian - (yCartesian >> 1);*

*xCartesian = xCartesianTemp;*

*yCartesian = yCartesianTemp;*

*xCartesianTemp = xCartesian + (xCartesian >> 2);*

*yCartesianTemp = yCartesian + (yCartesian >> 2);*

*xCartesian = xCartesianTemp;*

*yCartesian = yCartesianTemp;*

*xCartesianTemp = xCartesian - (xCartesian >> 5);*

*yCartesianTemp = yCartesian - (yCartesian >> 5);*

*xCartesian = xCartesianTemp;*

*yCartesian = yCartesianTemp;*

*xCartesianTemp = xCartesian + (xCartesian >> 10);*

*yCartesianTemp = yCartesian + (yCartesian >> 10);*

*xCartesian = xCartesianTemp;*

*yCartesian = yCartesianTemp;*

*xCartesianTemp = xCartesian + (xCartesian >> 16);*

*yCartesianTemp = yCartesian + (yCartesian >> 16);*

*xCartesian = xCartesianTemp;*

*yCartesian = yCartesianTemp;*

*xCartesianTemp = xCartesian + (xCartesian >> 19);*

*yCartesianTemp = yCartesian + (yCartesian >> 19);*

*xCartesian = xCartesianTemp;*

*yCartesian = yCartesianTemp;*

*xCartesianTemp = xCartesian + (xCartesian >> 22);*

*yCartesianTemp = yCartesian + (yCartesian >> 22);*

*xCartesian = xCartesianTemp;*

*yCartesian = yCartesianTemp;*

*s = yCartesian;*

*c = xCartesian;*

*arctan = cumulativeAngleTheta;*

*}*

**Code Listing for Cordic.h**

***#ifndef*** *CORDIC\_H*

***#define******CORDIC\_H***

***#include*** *"ap\_fixed.h"*

***typedef******unsigned******int******UINTYPE\_6****;*

***typedef******ap\_fixed****<16,3>* ***THETA\_TYPE****;*

***typedef******ap\_fixed****<16,3>* ***COS\_SIN\_TYPE****;*

*//typedef float THETA\_TYPE; //For testing. Please Ignore*

*//typedef float COS\_SIN\_TYPE; //For testing. Please Ignore*

***const******int*** *NUM\_ITERATIONS=32;*

***const******int*** *NUM\_DEGREE=90;*

*//For testing. Please Ignore*

*//static THETA\_TYPE cordic\_phase[64]={0.78539816339744828000,0.46364760900080609000,0.24497866312686414000,0.12435499454676144000,0.06241880999595735000,0.03123983343026827700,0.01562372862047683100,0.00781234106010111110,0.00390623013196697180,0.00195312251647881880,0.00097656218955931946,0.00048828121119489829,0.00024414062014936177,0.00012207031189367021,0.00006103515617420877,0.00003051757811552610,0.00001525878906131576,0.00000762939453110197,0.00000381469726560650,0.00000190734863281019,0.00000095367431640596,0.00000047683715820309,0.00000023841857910156,0.00000011920928955078,0.00000005960464477539,0.00000002980232238770,0.00000001490116119385,0.00000000745058059692,0.00000000372529029846,0.00000000186264514923,0.00000000093132257462,0.00000000046566128731,0.00000000023283064365,0.00000000011641532183,0.00000000005820766091,0.00000000002910383046,0.00000000001455191523,0.00000000000727595761,0.00000000000363797881,0.00000000000181898940,0.00000000000090949470,0.00000000000045474735,0.00000000000022737368,0.00000000000011368684,0.00000000000005684342,0.00000000000002842171,0.00000000000001421085,0.00000000000000710543,0.00000000000000355271,0.00000000000000177636,0.00000000000000088818,0.00000000000000044409,0.00000000000000022204,0.00000000000000011102,0.00000000000000005551,0.00000000000000002776,0.00000000000000001388,0.00000000000000000694,0.00000000000000000347,0.00000000000000000173,0.00000000000000000087,0.00000000000000000043,0.00000000000000000022,0.00000000000000000011};*

*//static THETA\_TYPE rotationAngles[25] = {45, 26.56505118, 14.03624347, 7.125016349, 3.576334375, 1.789910608, 0.89517371, 0.447614171, 0.2238105, 0.111905677, 0.055952892, 0.027976453, 0.013988227, 0.006994114, 0.003497057, 0.001748528, 0.000874264, 0.000437132, 0.000218566, 0.000109283};*

***static******THETA\_TYPE*** *rotationAngles[64]={0.78539816339744828000,0.46364760900080609000,0.24497866312686414000,0.12435499454676144000,0.06241880999595735000,0.03123983343026827700,0.01562372862047683100,0.00781234106010111110,0.00390623013196697180,0.00195312251647881880,0.00097656218955931946,0.00048828121119489829,0.00024414062014936177,0.00012207031189367021,0.00006103515617420877,0.00003051757811552610,0.00001525878906131576,0.00000762939453110197,0.00000381469726560650,0.00000190734863281019,0.00000095367431640596,0.00000047683715820309,0.00000023841857910156,0.00000011920928955078,0.00000005960464477539,0.00000002980232238770,0.00000001490116119385,0.00000000745058059692,0.00000000372529029846,0.00000000186264514923,0.00000000093132257462,0.00000000046566128731,0.00000000023283064365,0.00000000011641532183,0.00000000005820766091,0.00000000002910383046,0.00000000001455191523,0.00000000000727595761,0.00000000000363797881,0.00000000000181898940,0.00000000000090949470,0.00000000000045474735,0.00000000000022737368,0.00000000000011368684,0.00000000000005684342,0.00000000000002842171,0.00000000000001421085,0.00000000000000710543,0.00000000000000355271,0.00000000000000177636,0.00000000000000088818,0.00000000000000044409,0.00000000000000022204,0.00000000000000011102,0.00000000000000005551,0.00000000000000002776,0.00000000000000001388,0.00000000000000000694,0.00000000000000000347,0.00000000000000000173,0.00000000000000000087,0.00000000000000000043,0.00000000000000000022,0.00000000000000000011};*

***void******cordic****(****bool*** *cordicMode,* ***THETA\_TYPE*** *targetAnglePhi,* ***COS\_SIN\_TYPE*** *xCartesian,* ***COS\_SIN\_TYPE*** *yCartesian,* ***COS\_SIN\_TYPE*** *&arctan,* ***COS\_SIN\_TYPE*** *&s,* ***COS\_SIN\_TYPE*** *&c);*

***#endif***

**Code Listing for CordicTest.cpp**

*/\**

*This is traditional CORDIC computation of sine and cosine.*

*The current code is based on [FXT: cordic-circ-demo.cc]*

*Correctly calculates cos and sine between 0-90 degrees (0-100).*

*INPUT:*

*double theta: Input angle*

*long n: Number of iterations.*

*OUTPUT:*

*double &s: Reference to the sine part*

*double &c: Reference to the cos part*

*error\_sin= [abs(s-zs)/zs]\*100;*

*error\_cos= [abs(c-zc)/zc]\*100;*

*Total\_Error\_Sin = sum(error\_sin)*

*Total\_error\_Cos = sum(error\_cos)*

*\*/*

***#include*** *<math.h>*

***#include*** *"cordic.h"*

***#include*** *<stdio.h>*

***#include*** *<stdlib.h>*

***using******namespace*** *std;*

*//#define M\_PI 3.1415926536897932384626*

***double******abs\_double****(****double*** *var){*

***if*** *( var < 0)*

*var = -var;*

***return*** *var;*

*}*

***int******main****(****int*** *argc,* ***char*** *\*\*argv)*

*{*

*//FILE \*fp;*

***COS\_SIN\_TYPE*** *s; //sine*

***COS\_SIN\_TYPE*** *c; //cos*

***COS\_SIN\_TYPE*** *arctan;*

***COS\_SIN\_TYPE*** *xValue;*

***COS\_SIN\_TYPE*** *yValue;*

***THETA\_TYPE*** *radian; //radian versuin of degree*

***for****(****int*** *i=0; i<91; i = i+15)*

*{*

*radian = i\*M\_PI/180;*

*//cordic(int cordicMode, THETA\_TYPE targetAnglePhi, COS\_SIN\_TYPE xCartesian, COS\_SIN\_TYPE yCartesian, COS\_SIN\_TYPE &arctan, COS\_SIN\_TYPE &s, COS\_SIN\_TYPE &c)*

***cordic****(****false****, radian, 1, 0, arctan, s, c);*

***fprintf****(stdout, "degree=%d, radian=%f, cos=%f, sin=%f, arctan=%f\n", i, (****double****)radian, (****double****)c, (****double****)s, (****double****)arctan);*

*}*

***fprintf****(stdout, "\n\n\n");*

***for*** *(****int*** *y = 1; y<=3; y++)*

*{*

***for*** *(****int*** *x = 1; x<=3; x++)*

*{*

*xValue = x\*1.0;*

*yValue = y\*1.0;*

*//cordic(int cordicMode, THETA\_TYPE targetAnglePhi, COS\_SIN\_TYPE xCartesian, COS\_SIN\_TYPE yCartesian, COS\_SIN\_TYPE &arctan, COS\_SIN\_TYPE &s, COS\_SIN\_TYPE &c)*

***cordic****(****true****, 0, xValue, yValue, arctan, s, c);*

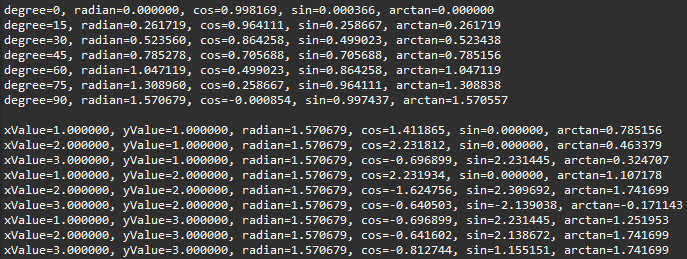
***fprintf****(stdout, "xValue=%f, yValue=%f, radian=%f, cos=%f, sin=%f, arctan=%f\n", (****double****)xValue, (****double****)yValue, (****double****)radian, (****double****)c, (****double****)s, (****double****)arctan);*

*}*

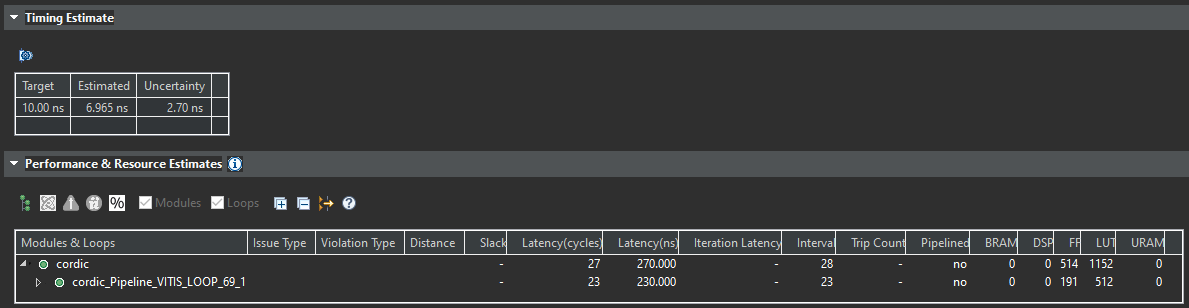
*}*

***return*** *0;*

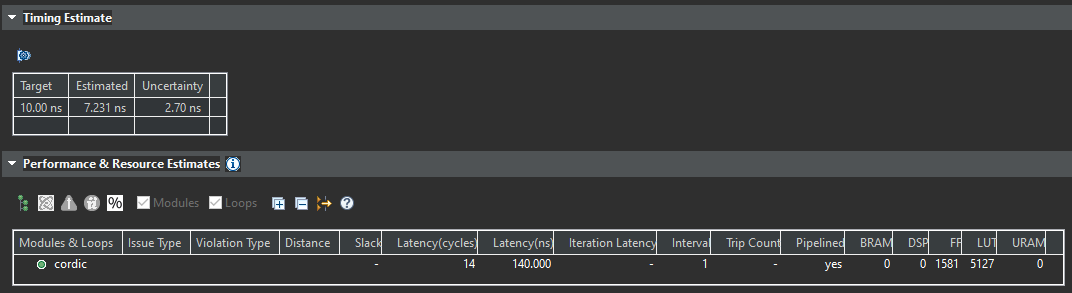
*}*



**Figure #4:** Output of Vitis HLS test bench in Rotation and Vector mode. As can be seen, we encounter overflow when the input xValue and yValue exceed the maximum that can be held in ap\_fixed<16,3>. However, when the data types are changed to ap\_fixed(116, 4> and higher, the overflow errors are cleared.



**Figure #5:** Vitis HLS synthesis results for CORDIC module without any optimization pragmas. This serves as a baseline for our optimization comparisons.



**Figure #6:** Vitis HLS synthesis results for CORDIC module with optimized pipeflining. As can be seen, latency is substantially lowered (nearly halved) with increased resource utilization (~3x – 5x increase).

**Architecture Optimization**

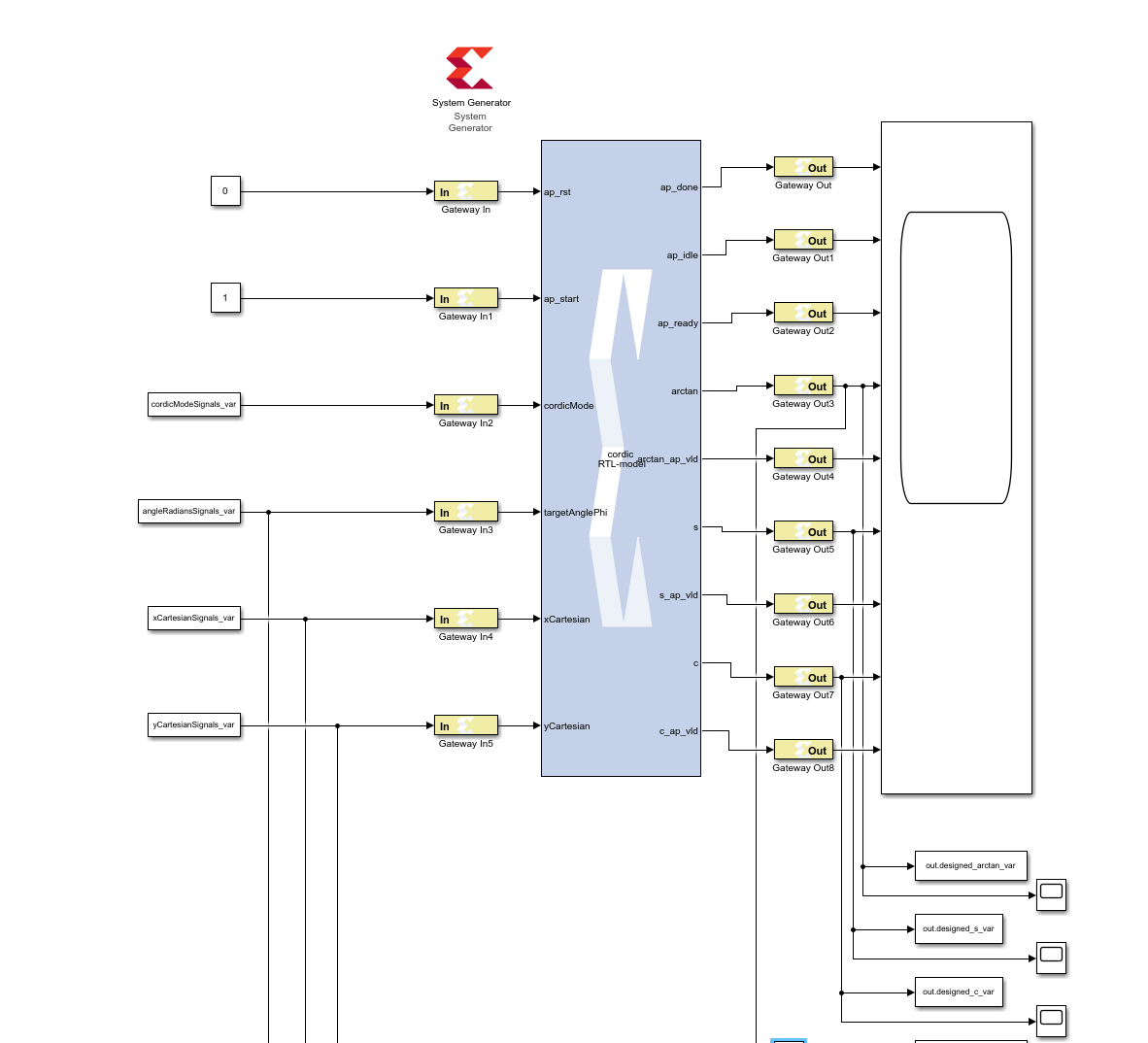
As visible in my code listing, I have four (4) “if” checks to correctly rotate the input matrix. The four conditions are as follows,

1. Minimization function < 0 and Rotation mode – Rotate counterclockwise
2. Minimization function < 0 and Vector mode – Rotate clockwise
3. Minimization function >= 0 and Rotation mode – Rotate clockwise
4. Minimization function >= 0 and Vector mode – Rotate counterclockwise

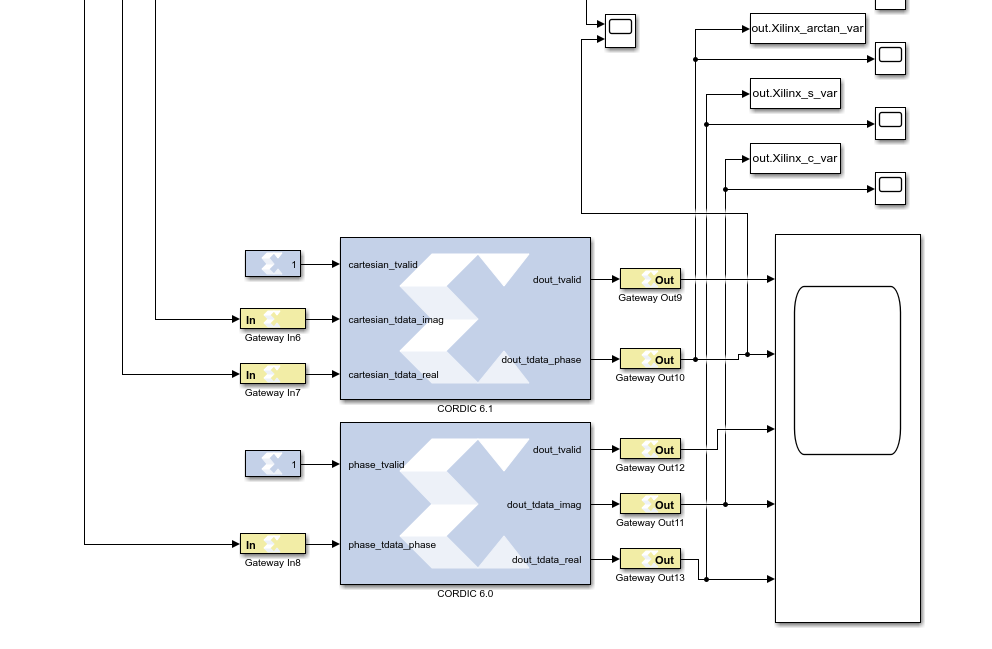
On presenting the conditions, there is an intuition that the program could be better optimized in just two (2) “if” checks to allow for lower resource utilization. However, due to choosing the incorrect minimization functions, I could not condense the program.

As shown in the previous photos, pipelining worsened the timing characteristics but greatly improved latency (halved) at the expense of substantially increase resource utilization. This was not further explored due to time constraints.

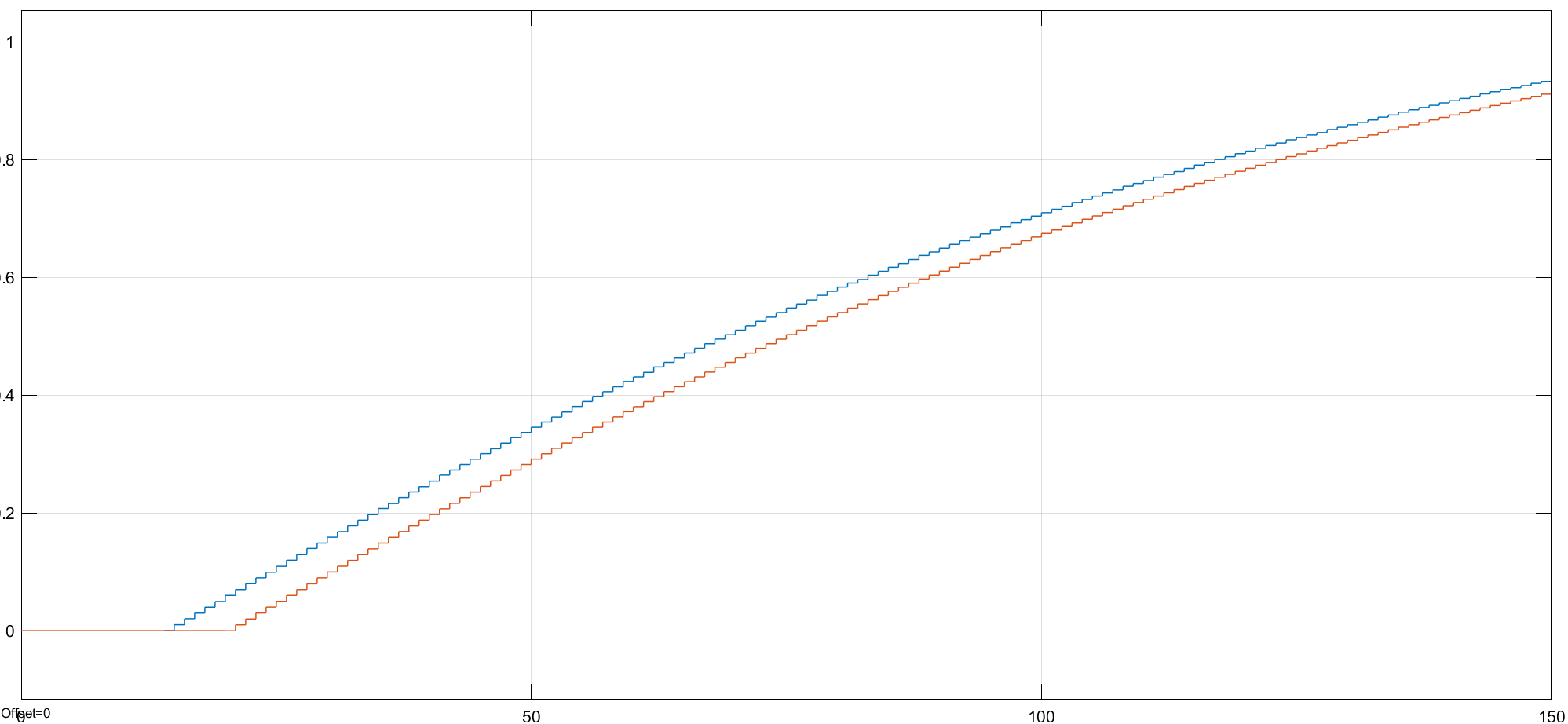
**Model Composer**

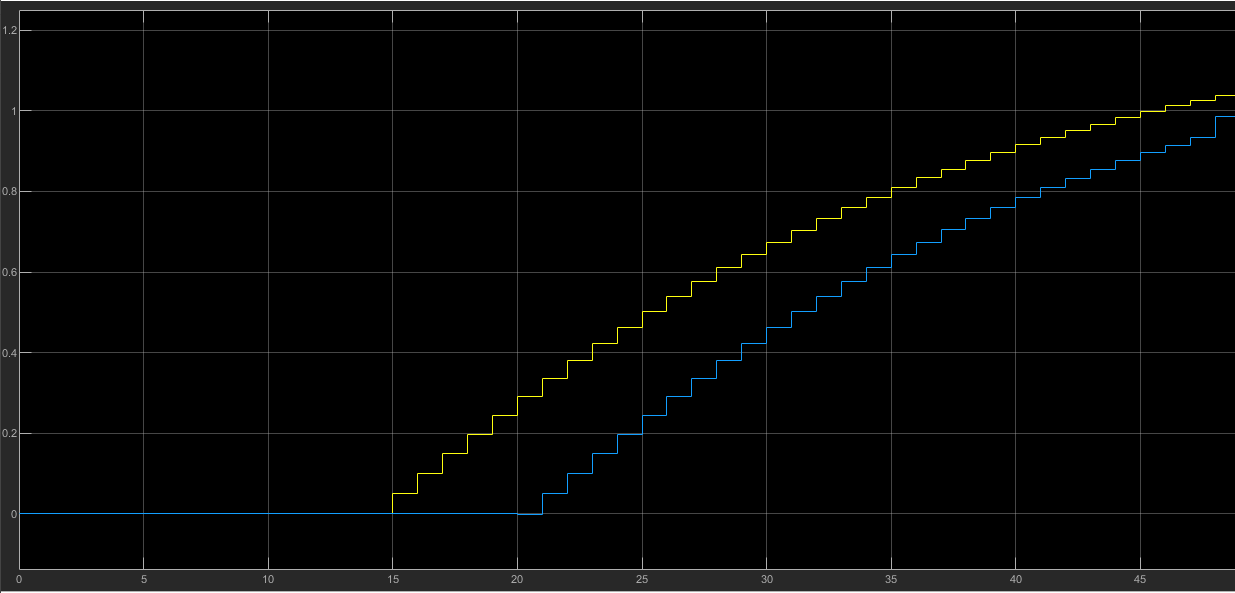


**Figure #7:** Model composer implementation of CORDIC Module created in VitisHLS.

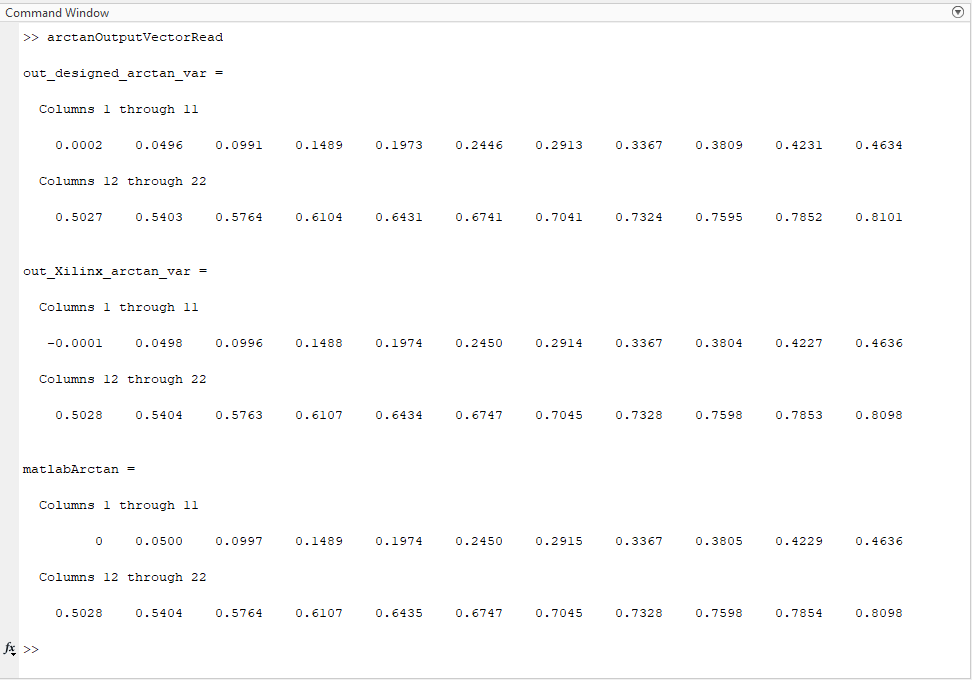


**Figure #8:** Model composer implementation of Xilinx designed, In-Built CORDIC Module for comparison.

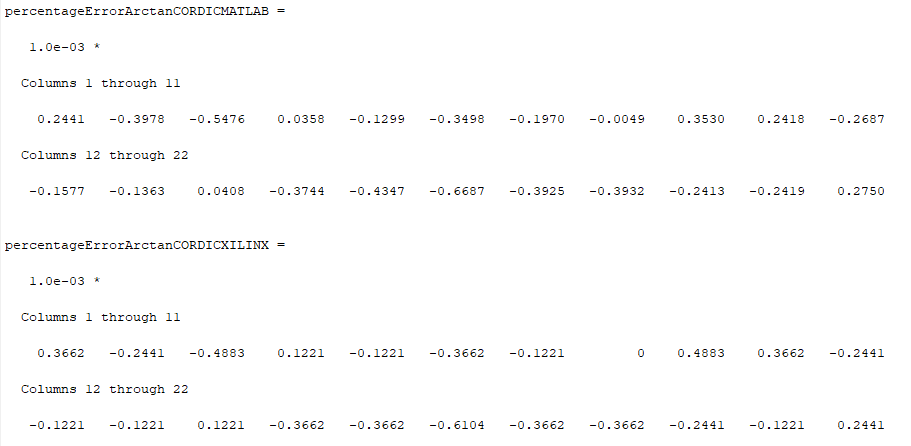




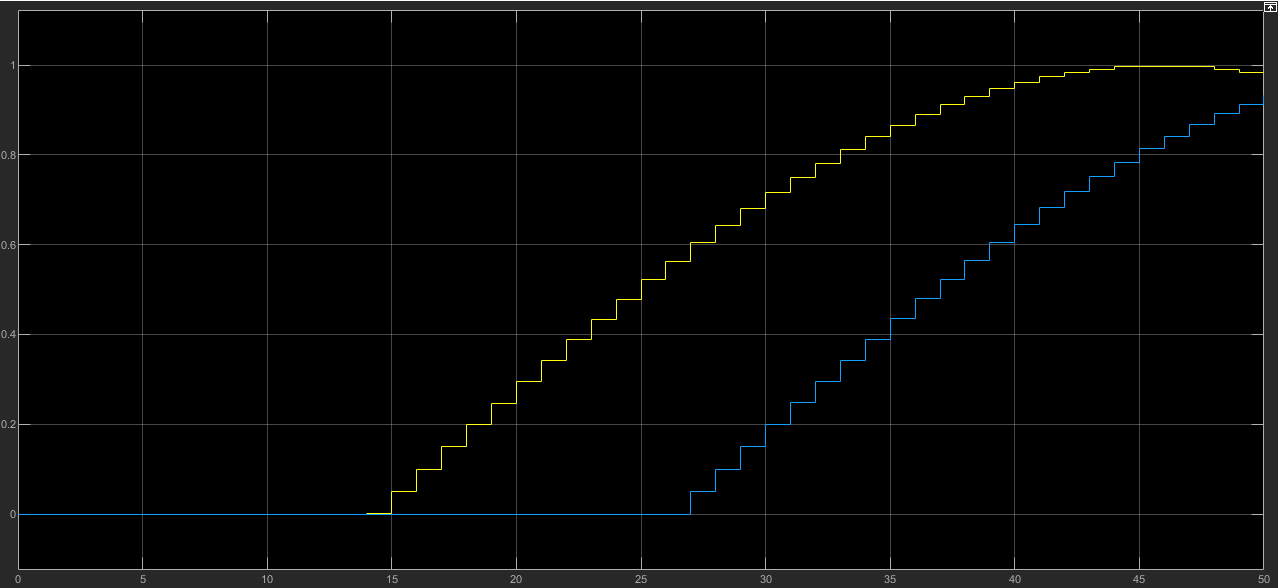
**Figure #9:** Model composer scope comparing arctan(y/x) input/output results between Vitis HLS designed CORDIC module (yellow) and Xilinx designed, In-Built CORDIC Module (blue). As can be seen, the results are in line with each other, but module designed by me provides output after 14 cycles.



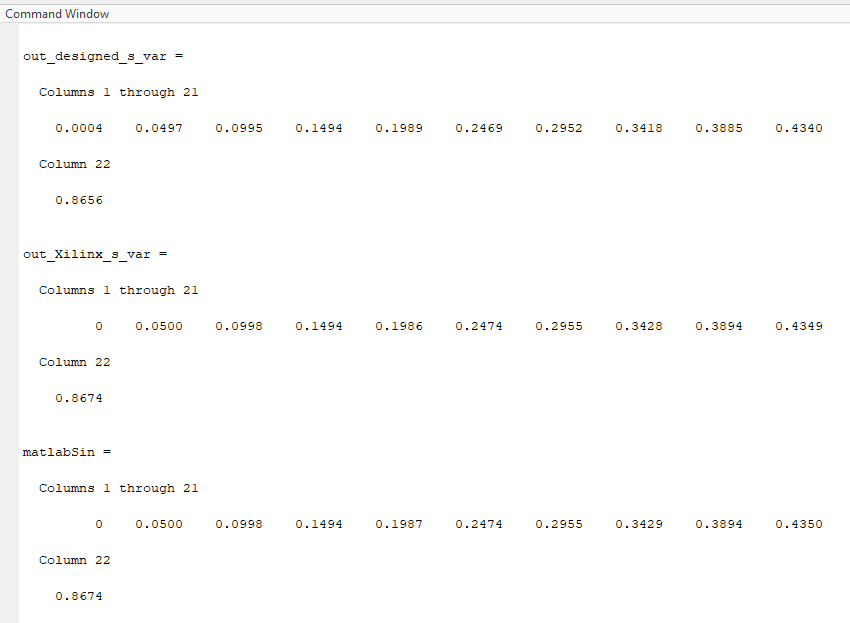
**Figure #10:** MATLAB Comparing arctangent outputs with sweep inputs. There are three matrices – arctangent obtained from the designed CORDIC module, arctan from inbuilt Xilinx CORDIC Module, and arctan generated by MATLAB’s inbuilt arctangent function.

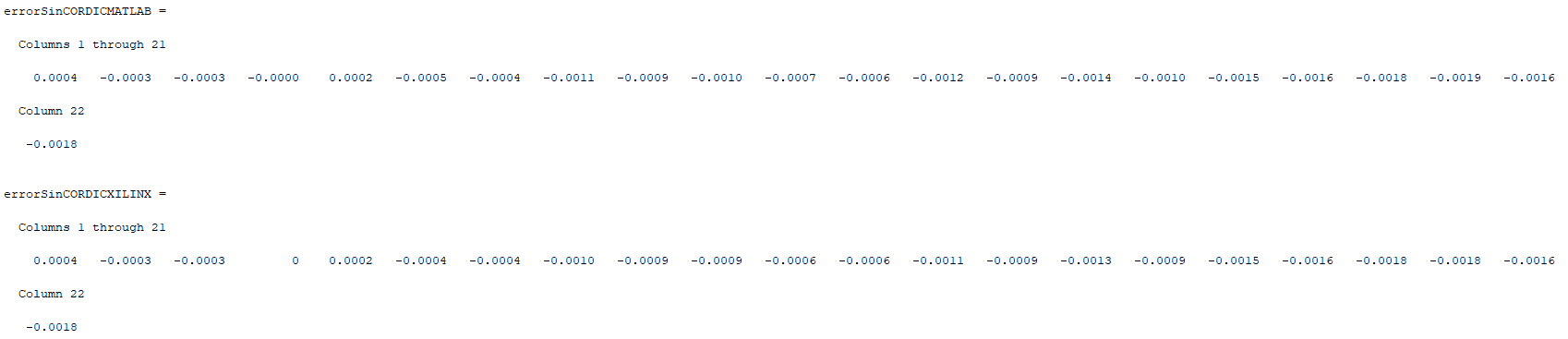


**Figure #11:** MATLAB Result comparing the arctangent results with two reference results. The first is the difference between the designed CORDIC module, and the inbuilt Xilinx CORDIC module. The second is the difference between the designed CORDIC Module and MATLAB’s Inbuilt arctangent function.

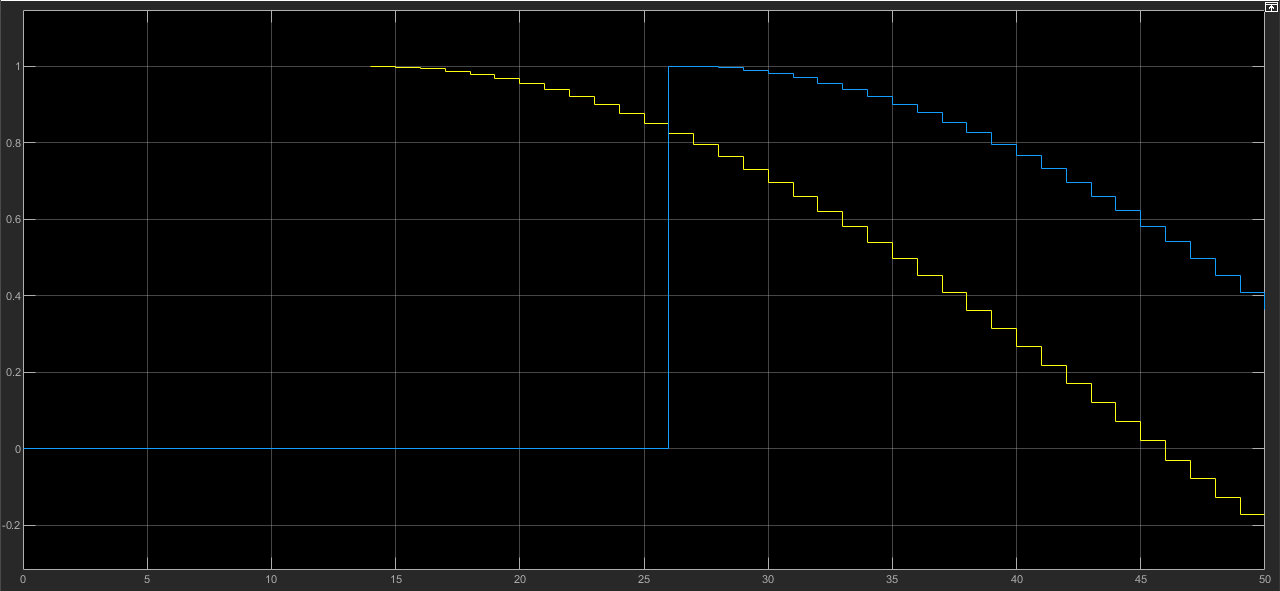


**Figure #12:** Model composer scope comparing sin(radian) input/output results between Vitis HLS designed CORDIC module (yellow) and Xilinx designed, In-Built CORDIC Module (blue). As can be seen, the results are in line with each other, but module designed by me provides output after 14 cycles.

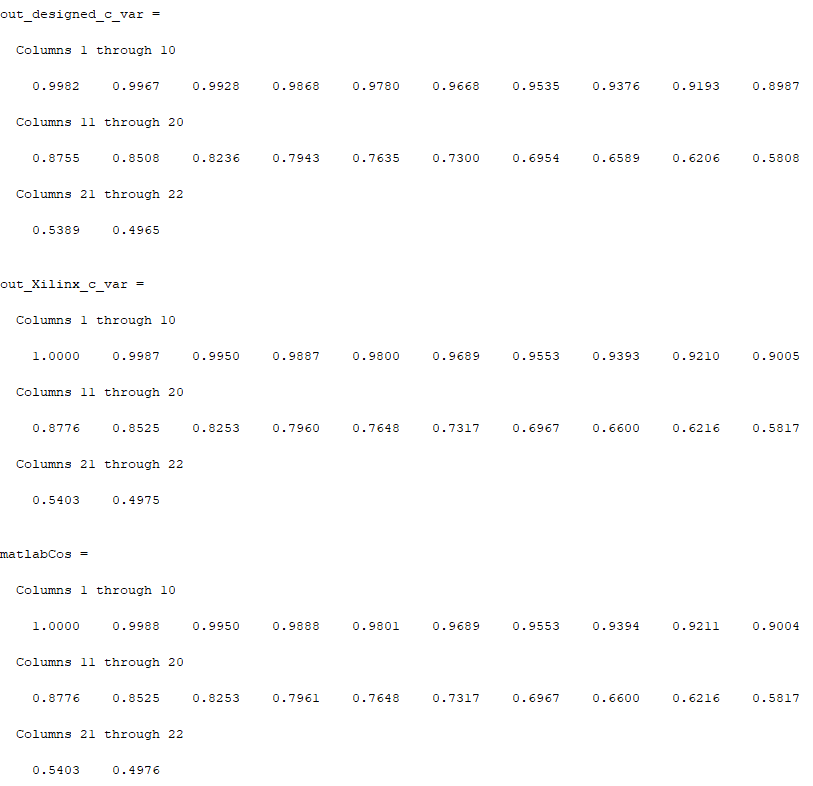
**Figure #13:** MATLAB Comparing sin outputs with sweep inputs. There are three matrices – sin obtained from the designed CORDIC module, sin from inbuilt Xilinx CORDIC Module, and sin generated by MATLAB’s inbuilt sin function.



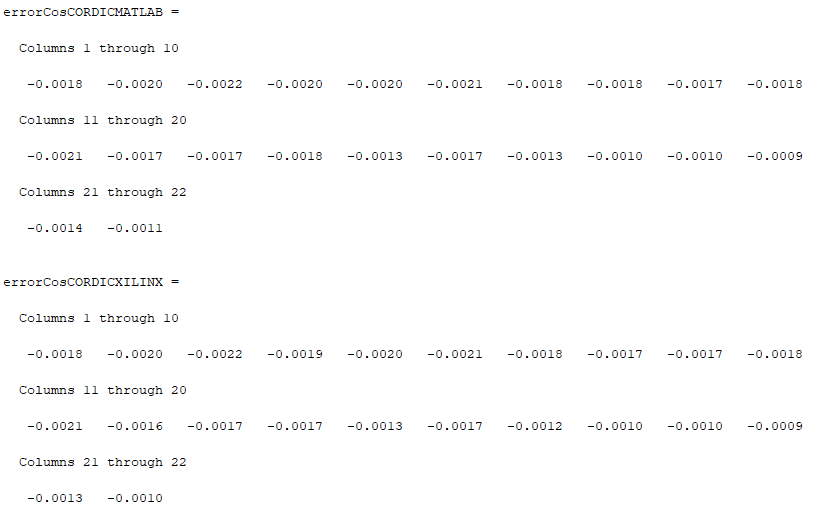
**Figure #14:** MATLAB Result comparing the sin results with two reference results. The first is the difference between the designed CORDIC module, and the inbuilt Xilinx CORDIC module. The second is the difference between the designed CORDIC Module and MATLAB’s Inbuilt sin function.

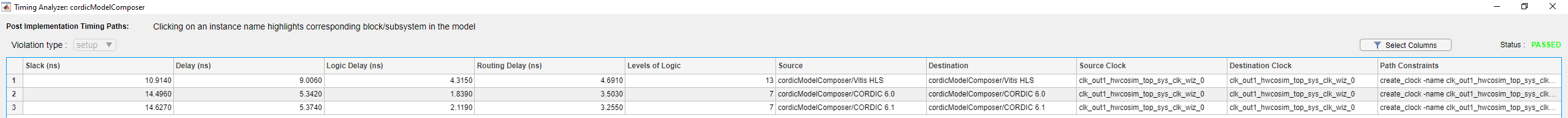


**Figure #15:** Model composer scope comparing cos(radian) input/output results between Vitis HLS designed CORDIC module (yellow) and Xilinx designed, In-Built CORDIC Module (blue). As can be seen, the results are in line with each other, but module designed by me provides output after 14 cycles.

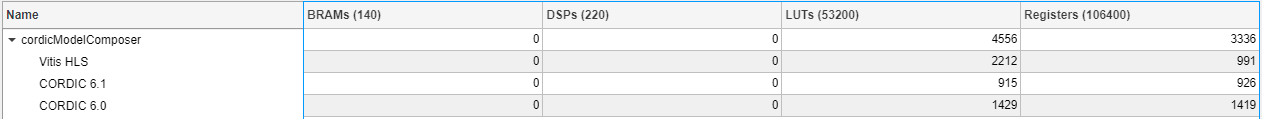


**Figure #16:** MATLAB Comparing cos outputs with sweep inputs. There are three matrices – cos obtained from the designed CORDIC module, cos from inbuilt Xilinx CORDIC Module, and cos generated by MATLAB’s inbuilt cos function.

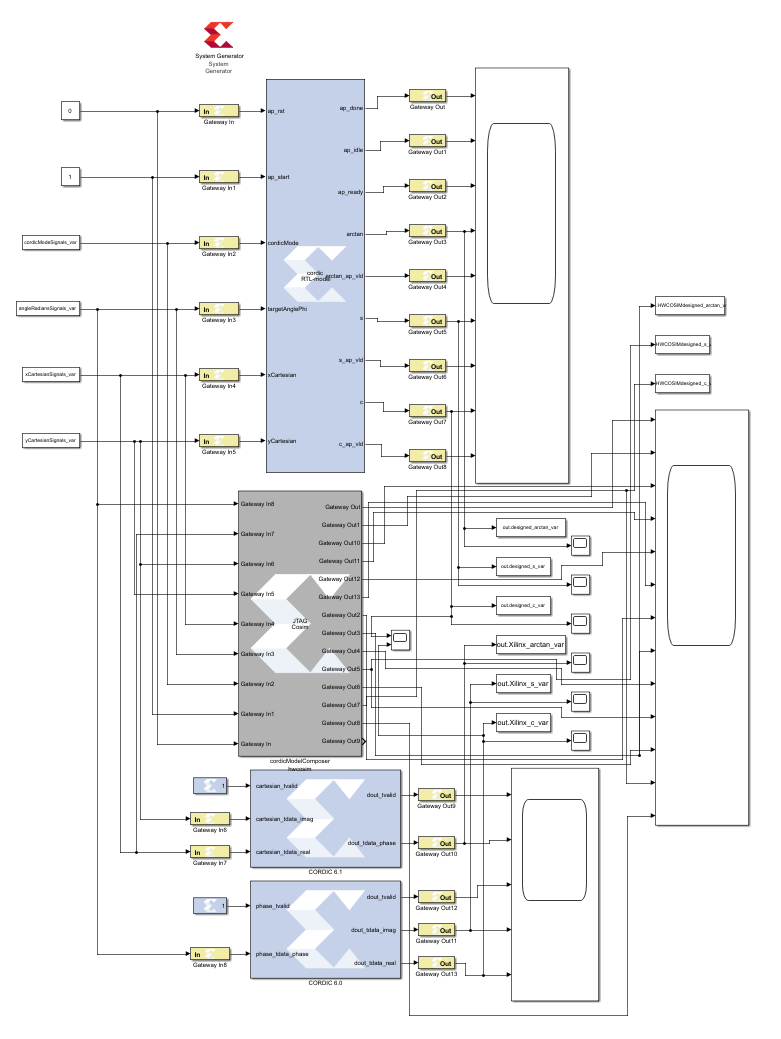
**Figure #17:** MATLAB Result comparing the cos results with two reference results. The first is the difference between the designed CORDIC module, and the inbuilt Xilinx CORDIC module. The second is the difference between the designed CORDIC Module and MATLAB’s Inbuilt sin function.



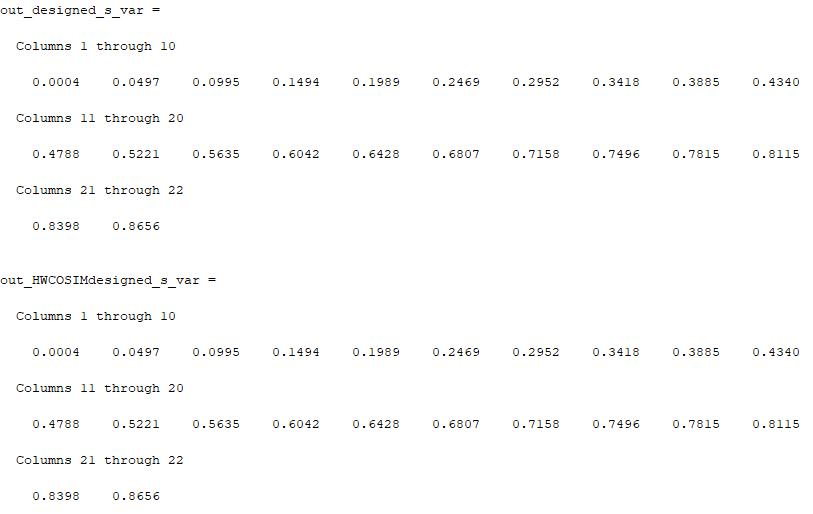
**Figure #18:** Model Composer Post Implementation Timing Results



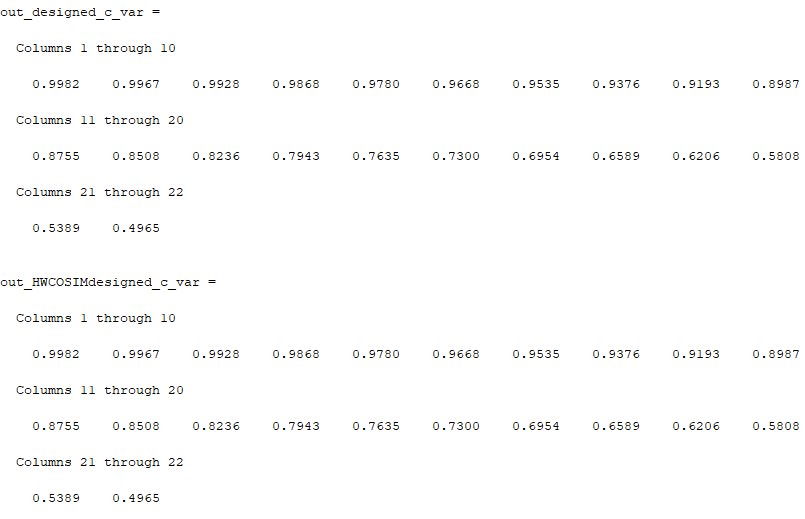
**Figure #19:** Model Composer Post Implementation Resource Utilization Results. The Vitis HLS design uses lesser components than both MATLAB CORDIC Modules combined.

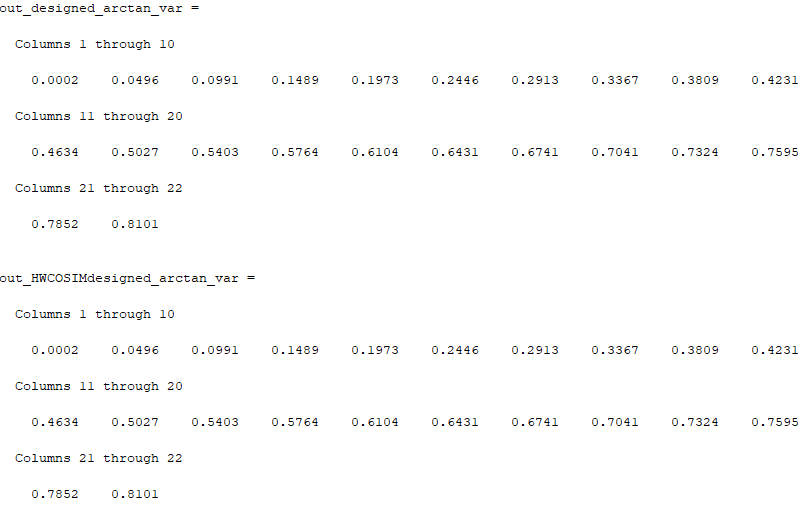


**Figure #20:** Model Composer Hardware Cosim Block and wiring.

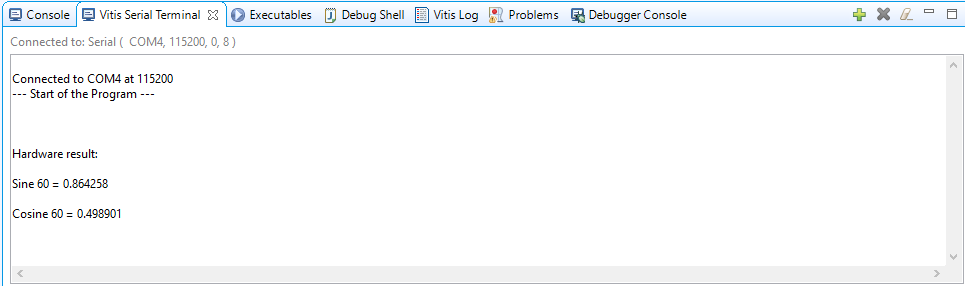


**Figure #21:** Model Composer Hardware Cosim results for sin sweep.

**Figure #22:** Model Composer Hardware Cosim results for cos sweep.



**Figure #23:** Model Composer Hardware Cosim results for arctan sweep.

**Vivado/ARM Core**

**Code Listing for main.cc**

*//*

*// C++ Application - main.cc for addertree - this version called t\_fixed*

*//*

*// For other applications, your project name will replace the t\_fixed and T\_fixed used below.*

*// The variables in your VitisHLS file, for example "a" will be part of a function name to write to*

*// the accelerator from the ARM core as "XT\_fixed\_Set\_a" below.*

*//*

*// This version is for VitisHLS design exported directly to IP Creator for Vivado*

*// This version assumes that the VitisHLS code uses the AXI pragma for I/O*

*// This version assumes that the ap\_fixed data type is used for integer/fraction.*

*// This version uses ap\_fixed<16,3, AP\_RND, AP\_SAT> for a 16 bit signed word with 3 integer bits.*

*// standard .h for c++ math and I/O*

***#include*** *<cmath>*

***#include*** *<iostream>*

*// .h file based on your original .hls design input*

***#include*** *"cordic.h"*

*// .h file from export to IP Creator from VitisHLS. This file is in the repository added to Vitis*

*// It will be copied from the drivers folder to the local in my example at*

*// C:\ELEC522\cavallar\Tree\_Fixed\_HLS\_Vitis\Vitis\design\_1\_wrapper\ps7\_cortex9\_0\standalone\_ps7\_c\include*

***#include*** *"xcordic.h"*

*// Add the include path C:\Xilinx\Vitis\_HLS\2021.1\include to Vitis project properties*

*// Add this, right click on the project name to access C/C++ Build Settings then C/C++ General*

*// Finally add path to Paths and Symbols, include directory*

*// For the c++ file for the fixed point data type*

***#include*** *"ap\_fixed.h"*

*// IMPORTANT HLS t\_fixed HW instance - Needed pointer - see the xt\_fixed.h file*

*XCordic Cordic;*

*// The Board Support Package that Vitis builds when you start a project should have these files*

*// Memory map file from*

*// C:\ELEC522\cavallar\Tree\_Fixed\_HLS\_Vitis\Vitis\design\_1\_wrapper\ps7\_cortexa9\_0\standalone\_ps7\_cortexa9\_0\bsp\ps7\_cortexa9\_0\include\*

***#include*** *"xparameters.h"*

*// The Xilinx time function file for measuring ARM core cycles*

***#include*** *"xtime\_l.h"*

*// Xilinx function for interrupt controller - may not be used in polling example.*

***#include*** *"xscugic.h"*

*// Used by C++ for the print command cout*

***using******namespace*** *std;*

*// Define our fixed point type used here to match the PL accelerator*

***typedef*** *ap\_fixed<16,3, AP\_RND, AP\_SAT> FIXED\_TYPE;*

*// The AXI interface sends only "raw bits" so we need to convert standard data types we use to*

*// match the ap\_fixed data type in the PL accelerator.*

*// Using C++ functions per - RJ Cunningham*

***int******get\_int\_reinterpret****(FIXED\_TYPE x) {*

***return*** *\*(****reinterpret\_cast****<****short*** *\*>(&x));*

*}*

*FIXED\_TYPE* ***get\_fixed\_reinterpret****(****int*** *x) {*

***return*** *\*(****reinterpret\_cast****<FIXED\_TYPE \*>(&x));*

*}*

*// actual MAIN control program running on ARM processor PS*

***int******main****()*

*{*

*cout << "--- Start of the Program ---" <<* ***endl****;*

*// values for simulation and testing using C++ format with ap\_fixed*

***bool*** *cordicMode;*

*THETA\_TYPE radian; //radian versuin of degree*

*COS\_SIN\_TYPE xValue;*

*COS\_SIN\_TYPE yValue;*

*COS\_SIN\_TYPE arctan\_out;*

*COS\_SIN\_TYPE s\_out; //sine*

*COS\_SIN\_TYPE c\_out; //cos*

*//Test the cordic in sin/cos mode (rotation mode)*

*cordicMode =* ***false****;*

*xValue = 1;*

*yValue = 0;*

*//Defining unsigned int for AXI transfer*

***unsigned******int*** *cordicMode\_u32;*

***unsigned******int*** *radian\_u32;*

***unsigned******int*** *xValue\_u32;*

***unsigned******int*** *yValue\_u32;*

***unsigned******int*** *arctan\_out\_u32;*

***unsigned******int*** *s\_out\_u32;*

***unsigned******int*** *c\_out\_u32;*

*//Reinterpreting for AXI transfer*

*cordicMode\_u32 = get\_int\_reinterpret(cordicMode);*

*xValue\_u32 = get\_int\_reinterpret(xValue);*

*yValue\_u32 = get\_int\_reinterpret(yValue);*

*XCordic\_Initialize(&Cordic, 0);*

*XCordic\_Set\_cordicMode(&Cordic, cordicMode\_u32);*

*XCordic\_Set\_xCartesian(&Cordic, xValue\_u32);*

*XCordic\_Set\_yCartesian(&Cordic, yValue\_u32);*

***int*** *i = 60;*

*radian = i\*M\_PI/180;*

*radian\_u32 = get\_int\_reinterpret(radian);*

*XCordic\_Set\_targetAnglePhi(&Cordic, radian\_u32);*

*XCordic\_Start(&Cordic);*

***do***

*{*

*arctan\_out\_u32 = XCordic\_Get\_arctan(&Cordic);*

*s\_out\_u32 = XCordic\_Get\_s(&Cordic);*

*c\_out\_u32 = XCordic\_Get\_c(&Cordic);*

*}*

***while*** *(!XCordic\_IsReady(&Cordic));*

*arctan\_out = get\_fixed\_reinterpret(arctan\_out\_u32);*

*s\_out = get\_fixed\_reinterpret(s\_out\_u32);*

*c\_out = get\_fixed\_reinterpret(c\_out\_u32);*

*cout << "\nHardware result: \nSine 60 = " << s\_out << "\nCosine 60 = " << c\_out <<* ***endl****;*

***return*** *0;*

*}*

**Code Listing for cordic.h**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Device: All

//Design Name: t\_fixed

//Purpose:

// This is the header for the treeadd.cpp design.

//Reference:

//Revision History:

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**#ifndef** CORDIC\_H

**#define** CORDIC\_H

**#include** "ap\_fixed.h"

**typedef** **unsigned** **int** UINTYPE\_6;

**typedef** ap\_fixed<16,3, *AP\_RND*, *AP\_SAT*> THETA\_TYPE;

**typedef** ap\_fixed<16,3, *AP\_RND*, *AP\_SAT*> COS\_SIN\_TYPE;

//typedef float THETA\_TYPE; //For testing. Please Ignore

//typedef float COS\_SIN\_TYPE; //For testing. Please Ignore

**const** **int** NUM\_ITERATIONS=32;

**const** **int** NUM\_DEGREE=90;

//For testing. Please Ignore

//static THETA\_TYPE cordic\_phase[64]={0.78539816339744828000,0.46364760900080609000,0.24497866312686414000,0.12435499454676144000,0.06241880999595735000,0.03123983343026827700,0.01562372862047683100,0.00781234106010111110,0.00390623013196697180,0.00195312251647881880,0.00097656218955931946,0.00048828121119489829,0.00024414062014936177,0.00012207031189367021,0.00006103515617420877,0.00003051757811552610,0.00001525878906131576,0.00000762939453110197,0.00000381469726560650,0.00000190734863281019,0.00000095367431640596,0.00000047683715820309,0.00000023841857910156,0.00000011920928955078,0.00000005960464477539,0.00000002980232238770,0.00000001490116119385,0.00000000745058059692,0.00000000372529029846,0.00000000186264514923,0.00000000093132257462,0.00000000046566128731,0.00000000023283064365,0.00000000011641532183,0.00000000005820766091,0.00000000002910383046,0.00000000001455191523,0.00000000000727595761,0.00000000000363797881,0.00000000000181898940,0.00000000000090949470,0.00000000000045474735,0.00000000000022737368,0.00000000000011368684,0.00000000000005684342,0.00000000000002842171,0.00000000000001421085,0.00000000000000710543,0.00000000000000355271,0.00000000000000177636,0.00000000000000088818,0.00000000000000044409,0.00000000000000022204,0.00000000000000011102,0.00000000000000005551,0.00000000000000002776,0.00000000000000001388,0.00000000000000000694,0.00000000000000000347,0.00000000000000000173,0.00000000000000000087,0.00000000000000000043,0.00000000000000000022,0.00000000000000000011};

//static THETA\_TYPE rotationAngles[25] = {45, 26.56505118, 14.03624347, 7.125016349, 3.576334375, 1.789910608, 0.89517371, 0.447614171, 0.2238105, 0.111905677, 0.055952892, 0.027976453, 0.013988227, 0.006994114, 0.003497057, 0.001748528, 0.000874264, 0.000437132, 0.000218566, 0.000109283};

**static** THETA\_TYPE rotationAngles[64]={0.78539816339744828000,0.46364760900080609000,0.24497866312686414000,0.12435499454676144000,0.06241880999595735000,0.03123983343026827700,0.01562372862047683100,0.00781234106010111110,0.00390623013196697180,0.00195312251647881880,0.00097656218955931946,0.00048828121119489829,0.00024414062014936177,0.00012207031189367021,0.00006103515617420877,0.00003051757811552610,0.00001525878906131576,0.00000762939453110197,0.00000381469726560650,0.00000190734863281019,0.00000095367431640596,0.00000047683715820309,0.00000023841857910156,0.00000011920928955078,0.00000005960464477539,0.00000002980232238770,0.00000001490116119385,0.00000000745058059692,0.00000000372529029846,0.00000000186264514923,0.00000000093132257462,0.00000000046566128731,0.00000000023283064365,0.00000000011641532183,0.00000000005820766091,0.00000000002910383046,0.00000000001455191523,0.00000000000727595761,0.00000000000363797881,0.00000000000181898940,0.00000000000090949470,0.00000000000045474735,0.00000000000022737368,0.00000000000011368684,0.00000000000005684342,0.00000000000002842171,0.00000000000001421085,0.00000000000000710543,0.00000000000000355271,0.00000000000000177636,0.00000000000000088818,0.00000000000000044409,0.00000000000000022204,0.00000000000000011102,0.00000000000000005551,0.00000000000000002776,0.00000000000000001388,0.00000000000000000694,0.00000000000000000347,0.00000000000000000173,0.00000000000000000087,0.00000000000000000043,0.00000000000000000022,0.00000000000000000011};

**void** **cordic**(**bool** cordicMode, THETA\_TYPE targetAnglePhi, COS\_SIN\_TYPE xCartesian, COS\_SIN\_TYPE yCartesian, COS\_SIN\_TYPE &arctan, COS\_SIN\_TYPE &s, COS\_SIN\_TYPE &c);

**#endif**

