ELEC522 – Fall 2022

Project 4: Using VitisHLS to implement a CORDIC module on Zynq

By

Hsuan-You (Shaun) Lin

Rice University, Houston, TX

Nov. 10, 2022

Supervised by

Dr. Joseph R. Cavallaro

Contents

List of Figures

Write C/C++ code for the CORDIC circular mode module using Vitis HLS:

C++ code for CORDIC circular mode:

Here I modify the CORDIC_Kastner_Book_Example C++ code for the CORDIC module, I designed a 16 bit signed fixed-point input in the header file, and implemented the sine, cosine, and inverse tangent functions in the main cpp file. Also a testbench for the CORDIC circular mode is provided in the *Figure 4*, as you can see I choose 45 and -75 degree for theta angle, and the result as shown in *Figure 5*, from the output result that was calculated correct, which means this system is not only work in 0~180 degree, it also work in -180~0 degree.

(a) Screen capture of CORDIC circular mode main cpp code – part 1.

```
cordic.cpp ✕   cordic.h   cordic_testbench.cpp   cordic_csim.log   compare reports

#include <iostream>
#include "cordic.h"

static THETA_TYPE cordic_phase[NUM_ITERATIONS]={0.78539816339744828000,0.46364760900080609000,0.24497866312686414
                                    0.00097656218955931946,0.00048828121119489829,0.00024414062014936177,0.0001220
                                    0.00000095367431640596,0.00000047683715820309,0.00000023841857910156,0.0000001
                                    0.00000000093132257462,0.00000000046566128731,0.00000000023283064365,0.0000000
                                    0.00000000000090949470,0.00000000000045474735,0.00000000000022737368,0.0000000
                                    0.00000000000000088818,0.00000000000000044409,0.00000000000000022204,0.0000000
                                    0.00000000000000000087,0.00000000000000000043,0.00000000000000000022,0.0000000

static char scaling_factors[NUM_FACTORS]={1, 2, 5, 10, 16, 19, 22};

void cordic(bool modes, THETA_TYPE x_in, THETA_TYPE y_in, THETA_TYPE theta_in, THETA_TYPE &x_out, THETA_TYPE &y_ou
{
    x_out, y_out, theta_out = 0;
    THETA_TYPE xi, yi, angle, phase;
    THETA_TYPE theta = 0;
    THETA_TYPE x_shift, y_shift = 0;
    char Quadrant = 0;

/*----------------------------------------------------------------------------------------------------
cordic function:
    Input:
        THETA_TYPE x_in       :  input vector x value
        THETA_TYPE y_in       :  input vector y value
        THETA_TYPE theta_in   :  input theta angle (should be in range [-PI, PI])
        THETA_TYPE &x_out     :  output x value after cordic
        THETA_TYPE &y_out     :  output y value after cordic
        THETA_TYPE &theta_out:  output theta after cordic
        bool modes            :  if models = 0 -> COS_SIN mode / if models = 1 -> ARCTAN mode"
----------------------------------------------------------------------------------------------------*/

//======================Choose modes==============================
// models = 0 -> COS_SIN mode
    if(modes == 0){
        if(theta_in < -pi/2){
            angle = theta_in + pi;
            Quadrant = 2;
        }
        else if(theta_in > pi/2){
            angle = theta_in - pi;
            Quadrant = 1;
        }
        else{
            angle = theta_in;
            Quadrant = 0;
        }
        xi = x_in;
        yi = y_in;
    }

// models = 1 -> ARCTAN mode
    if(modes == 1){
        if (x_in >=0){
            xi = x_in;
            yi = y_in;
            Quadrant = 0;
        }
        if(x_in < 0){
            if(y_in > 0){
                xi = y_in;
                yi = -x_in;
                Quadrant = 3;
            }
            else{
                xi = -y_in;
                yi = x_in;
                Quadrant = 4;
            }
        }
    }
```

Figure 1.

(b) Screen capture of CORDIC circular mode main cpp code – part 1.

```cpp

//===================Rotate and Scale============================
Rotation_function:
    for (int i = 0; i < NUM_ITERATIONS; i++){
        //2^(-i)
        phase = cordic_phase[i];
        x_shift = (yi >> i);
        y_shift = (xi >> i);
        if((modes == 0 && theta < angle) || (modes == 1 && yi < 0)){
            xi = xi - x_shift;
            yi = yi + y_shift;
            theta = theta + phase;
        }
        else{
            xi = xi + x_shift;
            yi = yi - y_shift;
            theta = theta - phase;
        }
    }

Scale_function:
    for (int j = 0; j < NUM_FACTORS; j++){
        x_shift = (xi >> scaling_factors[j]);
        y_shift = (yi >> scaling_factors[j]);
        if(j == 0 || j == 2){
            xi = xi - x_shift;
        }
        else{
            xi = xi + x_shift;
        }
        if((j == 0 || j == 2) && (modes == 0)){
            yi = yi - y_shift;
        }
        else{
            yi = yi + y_shift;
        }
    }

//======================save the result==========================
    if(modes == 0){
        if(Quadrant){
            x_out = -xi;
            y_out = -yi;
        }
        else{
            x_out = xi;
            y_out = yi;
        }
        theta_out = 0;
    }

    else{
        x_out = xi;
        y_out = 0;
        theta_out = -theta;
        if(Quadrant == 3){
            theta_out = theta_out + pi/2;
        }
        if(Quadrant == 4){
            theta_out = theta_out - pi/2;
        }
    }
    done = 1;
}

```

Figure 2.

(c) Screen capture of CORDIC circular mode header code.

cordic.cpp cordic.h cordic_testbench.cpp cordic_csim.log compare reports

```cpp
#ifndef CORDIC_H
#define CORDIC_H

#include "ap_fixed.h"

typedef unsigned int UINTYPE_12;
typedef ap_fixed<16,4> THETA_TYPE;
const int NUM_ITERATIONS = 64;
const int NUM_FACTORS = 7;

static THETA_TYPE pi = 3.141592653589793;

/*-----------------------------------------------------------------------------------------
cordic function:
    Input:
        THETA_TYPE x_in       :  input vector x value
        THETA_TYPE y_in       :  input vector y value
        THETA_TYPE theta_in   :  input theta angle (should be in range [-PI, PI])
        THETA_TYPE &x_out     :  output x value after cordic
        THETA_TYPE &y_out     :  output y value after cordic
        THETA_TYPE &theta_out:  output theta after cordic
        bool modes            :  if models = 0 -> COS_SIN mode / if models = 1 -> ARCTAN mode"
-----------------------------------------------------------------------------------------*/
void cordic(bool modes, THETA_TYPE x_in, THETA_TYPE y_in, THETA_TYPE theta_in, THETA_TYPE &x_out, THETA_TYPE &y_out, THETA_TYPE &theta_out, short done);

#endif

```

Figure 3.

(d) Screen capture of CORDIC circular mode testbench code.

cordic.cpp | cordic.h | cordic_testbench.cpp | compare reports | cordic_csim.log

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "cordic.h"

using namespace std;

#define PI 3.1415926

int main(int argc, char **argv)
{
    THETA_TYPE x_in, y_in, theta_in, x_out, y_out, theta_out, len_out;
    float cos_z, sin_z, x_sw, y_sw, len_sw;
    bool modes = 0;
    short done, degree;

    cout << "--- Start of Testbench ---" << endl;

// ---------Test 45 degree---------
    degree = 45;
    x_in = 1;
    y_in = 0;
    theta_in = (degree * pi/180);
    cos_z = cos((float)theta_in);
    sin_z = sin((float)theta_in);
    x_sw = cos_z * (float)x_in - sin_z * (float)y_in;
    y_sw = sin_z * (float)x_in + cos_z * (float)y_in;
    len_sw = sqrt(float(pow(x_sw, 2) + pow(y_sw, 2)));

    // models = 0 -> COS_SIN mode
    modes = 0;
    printf("Expected cordic COS = %f, SIN = %f\n", (float)x_sw, (float)y_sw);
    cordic(modes, x_in, y_in, theta_in, x_out, y_out, theta_out, done);
    printf("After cordic COS_out = %f, SIN_out = %f\n\n", (float)x_out, (float)y_out);

    // models = 1 -> ARCTAN mode
    modes = 1;
    printf("Expected cordic length = %f, theta(in degree) = %f\n", (float)len_sw, (float)degree);
    cordic(modes, x_sw, y_sw, 0, len_out, y_out, theta_out, done);
    printf("After cordic length_out = %f, theta_out(in degree) = %f\n\n", (float)len_out, (float)theta_out/PI*180);

// ---------Test -75 degree----------
    degree = -75;
    x_in = 1;
    y_in = 0;
    theta_in = (degree * pi/180);
    cos_z = cos((float)theta_in);
    sin_z = sin((float)theta_in);
    x_sw = cos_z * (float)x_in - sin_z * (float)y_in;
    y_sw = sin_z * (float)x_in + cos_z * (float)y_in;
    len_sw = sqrt(float(pow(x_sw, 2) + pow(y_sw, 2)));

    // models = 0 -> COS_SIN mode
    modes = 0;
    printf("Expected cordic COS = %f, SIN = %f\n", (float)x_sw, (float)y_sw);
    cordic(modes, x_in, y_in, theta_in, x_out, y_out, theta_out, done);
    printf("After cordic COS_out = %f, SIN_out = %f\n\n", (float)x_out, (float)y_out);

    // models = 1 -> ARCTAN mode
    modes = 1;
    printf("Expected cordic length = %f, theta(in degree) = %f\n", (float)len_sw, (float)degree);
    cordic(modes, x_sw, y_sw, 0, len_out, y_out, theta_out, done);
    printf("After cordic length_out = %f, theta_out(in degree) = %f\n\n", (float)len_out, (float)theta_out/PI*180);

    cout << "--- End of Testbench ---" << endl;

    return 0;
}
```

Figure 4.

(e) Screen capture of CORDIC circular mode testbench co-simulation result.

cordic.cpp | cordic.h | cordic_testbench.cpp | compare reports | cordic_csim.log

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../../fpga_hls/cordic_testbench.cpp in debug mode
   Generating csim.exe
--- Start of Testbench ---
Expected cordic COS = 0.707278, SIN = 0.706936
After cordic COS_out = 0.705566, SIN_out = 0.705566

Expected cordic length = 1.000000, theta(in degree) = 45.000000
After cordic length_out = 0.998291, theta_out(in degree) = 45.000128

Expected cordic COS = 0.258973, SIN = -0.965885
After cordic COS_out = 0.252197, SIN_out = -0.964844

Expected cordic length = 1.000000, theta(in degree) = -75.000000
After cordic length_out = 0.998291, theta_out(in degree) = -75.018865

--- End of Testbench ---
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

Figure 5.

Optimizations for CORDIC circular mode module in Vitis HLS:

Optimizations for CORDIC circular mode:

In this section, I built three different solutions to compare the different optimization methods, as you can see from Figure 6, there is no optimization for directive, and we synthesis the solution1 we got 82 latency and 1269 LUT, after that I tried to PIPELINE the cordic function, the system will run faster with 26 latency, but we got overall 5,723 LUT in solution2, which is not a good hardware resource costs, so I used AXILITE INTERFACE for each input signal, then we finally got 23 latency and 1361 LUT in solution3, we will need this for Vivado design flow.

Eventually, we got a better balance between system throughput and hardware resource costs from solution3.Also, my constraint configurations for each different optimizations we mentioned above as shown in *Figure 6*, *Figure 7*, and *Figure 8, and I also put the solution comparison table in Figure 9*.

(a) Screen capture of original Directive. (No optimization for Directive)

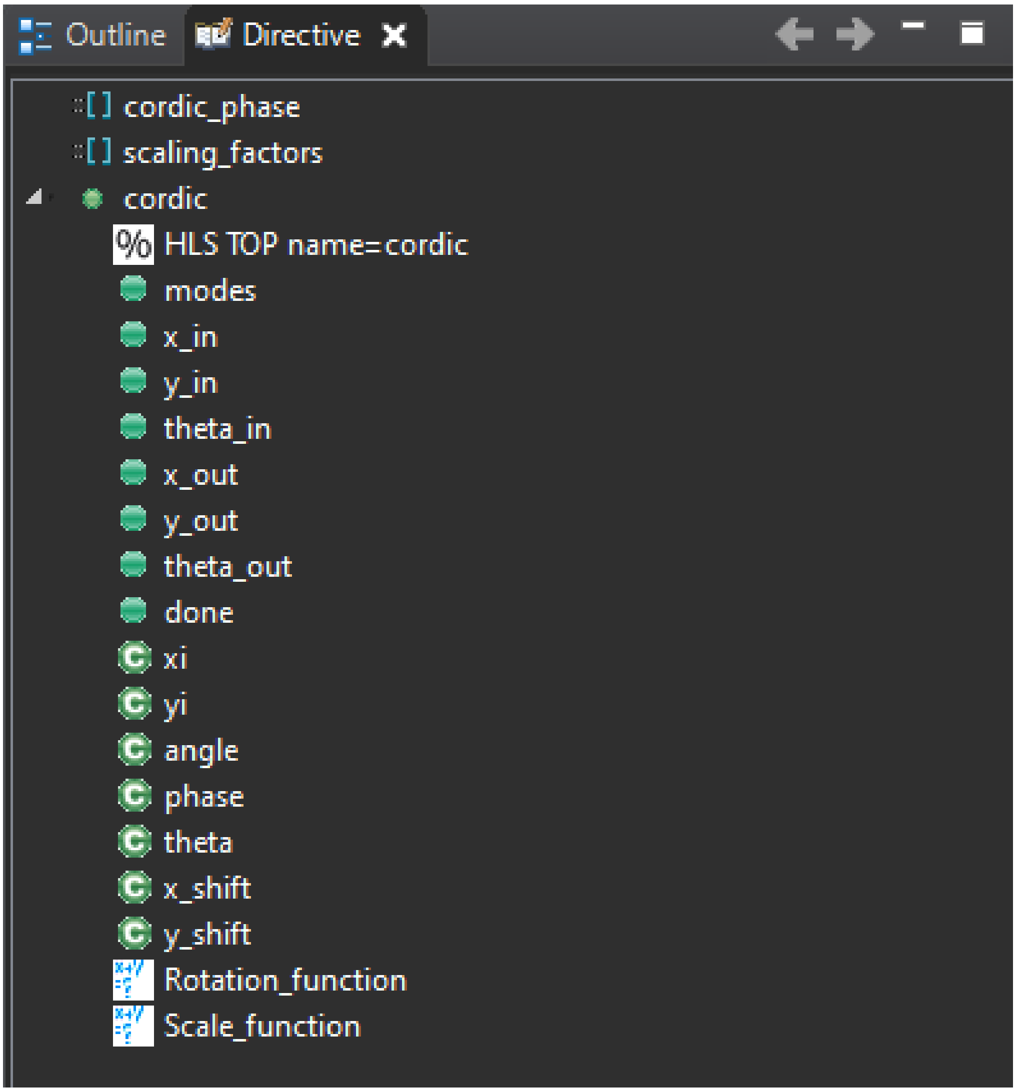


Figure 6.

(b) Screen capture of PIPELINE Directive.

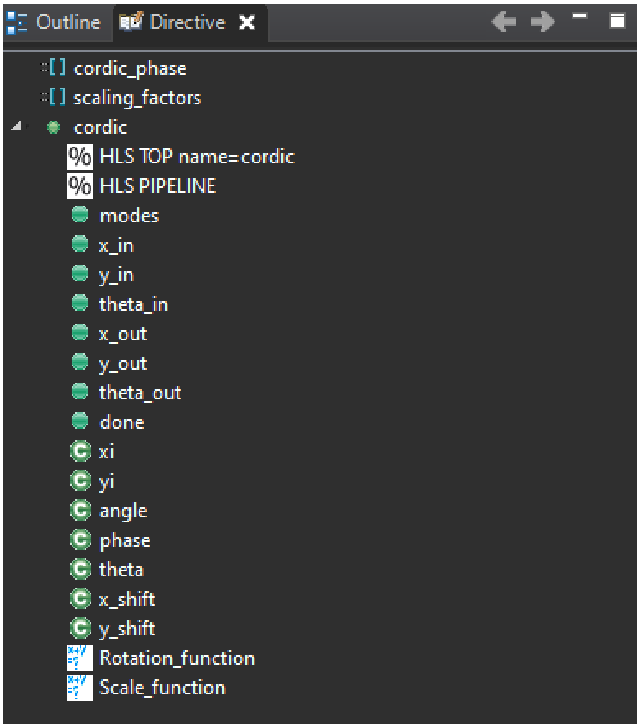


Figure 7.

(c) Screen capture of PIPELINE Directive.

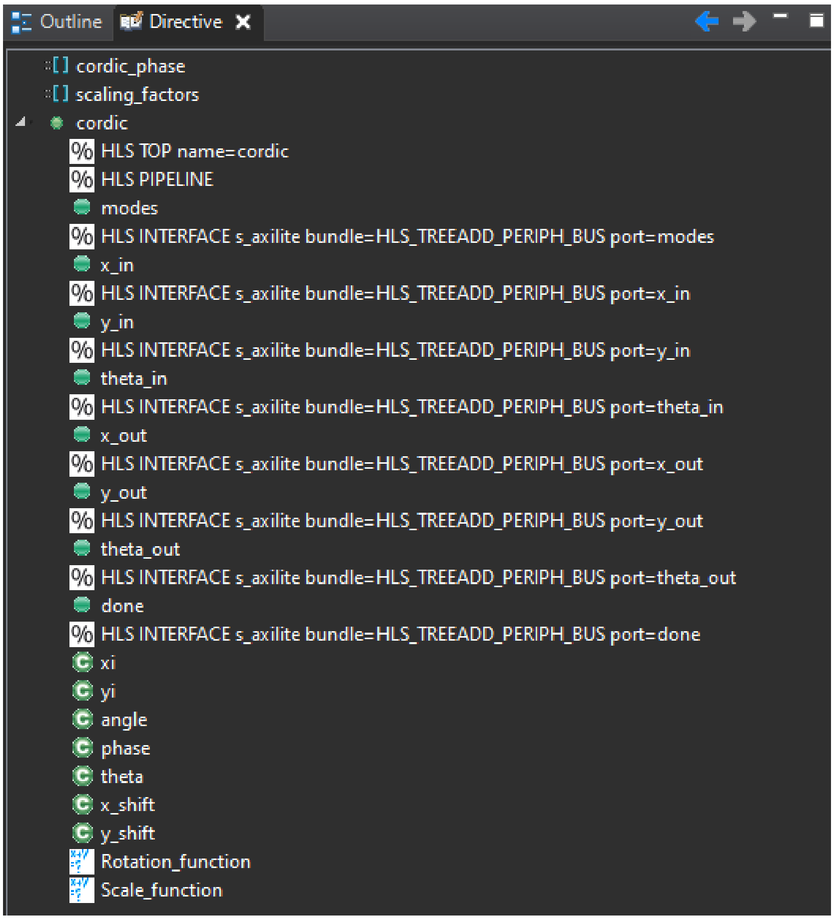


Figure 8.

(d) Screen capture of different solutions comparison table

Vitis HLS Report Comparison

All Compared Solutions

solution1-Default: xc7z020-clg484-1

solution2-Optimized: xc7z020-clg484-1

solution3-Pipeline: xc7z020-clg484-1

Performance Estimates

Timing

Clock		solution1-Default	solution2-Optimized	solution3-Pipeline
ap_clk	Target	10.00 ns	10.00 ns	10.00 ns
	Estimated	6.870 ns	7.267 ns	5.952 ns

Latency

		solution1-Default	solution2-Optimized	solution3-Pipeline
Latency (cycles)	min	82	26	23
	max	82	26	23
Latency (absolute)	min	0.820 us	0.260 us	0.230 us
	max	0.820 us	0.260 us	0.230 us
Interval (cycles)	min	83	1	1
	max	83	1	1

Utilization Estimates

	solution1-Default	solution2-Optimized	solution3-Pipeline
BRAM_18K	0	0	0
DSP	0	0	4
FF	317	2386	1585
LUT	1269	5723	1361
URAM	0	0	0

Figure 9.

Model Composer:

CORDIC circular mode system architecture in model composer:

In this section, I test the input data with CORDIC block and Vitis HLS block, input data generated from m_code, I set input angle theta_in from -180~180 degree, which means total 360 degree, from my HLS design, if we choose modes = 0, we will get COS_SIN mode results, if we choose modes = 1, it will be ARCTAN mode, then we need 360 times input data signal, so here I used timeseries function, then we can read the input data from workspace into each gateway.

In the *Figure 10*, upper block is Vitis HLS block, and lower block I used CORDIC block to verify the output results, scope output results as shown in *Figure 11* (COS_SIN mode) and *Figure 12* (ARCTAN mode), and I also exported the output data to workspace, here I created three different m_code to visualize the output results for each mode and conditions as shown in *Figure 13* (COS_SIN mode) and *Figure 14* (ARCTAN mode).

(a) Screen capture of my CORDIC Vitis HLS block with CORDIC block system architecture. (Model composer)

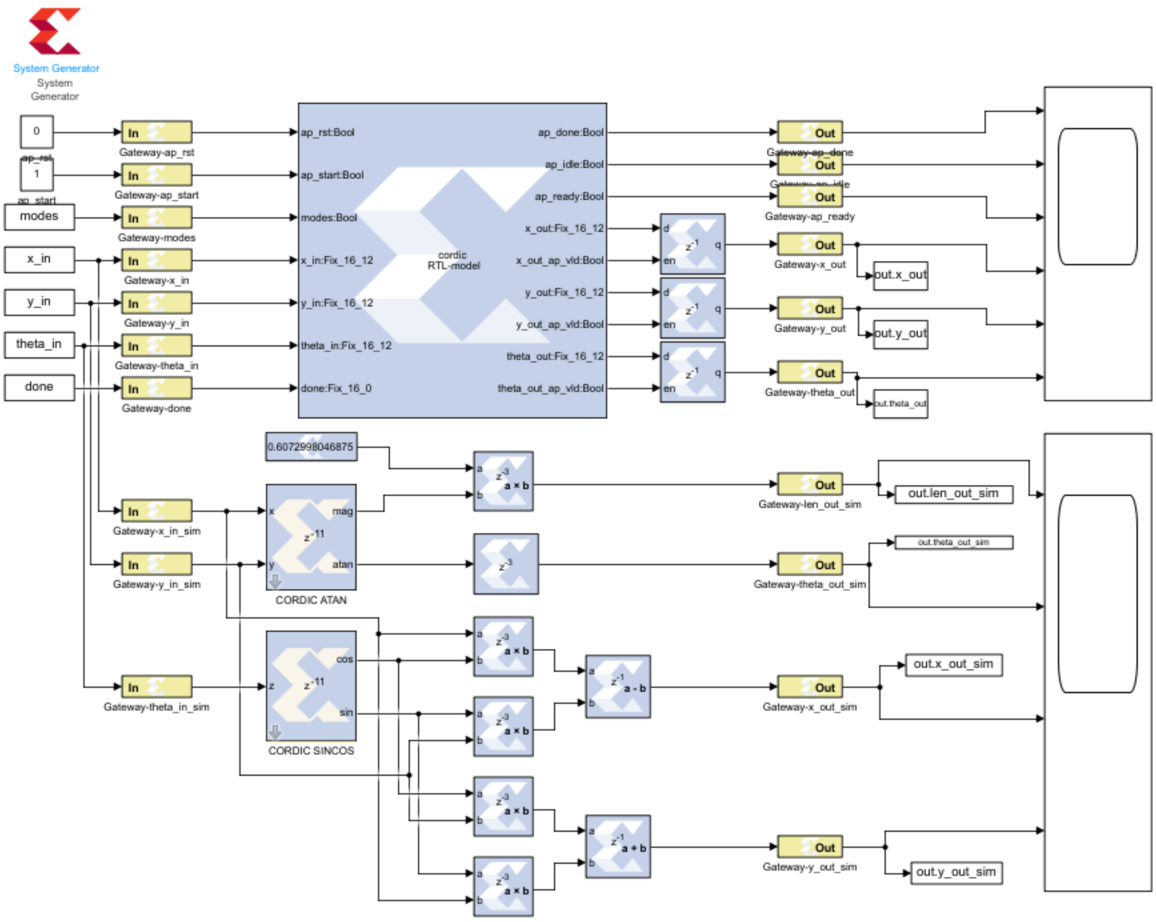


Figure 10.

(b) Screen capture of CORDIC Vitis HLS block and CORDIC block scope output results. (COS_SIN mode)

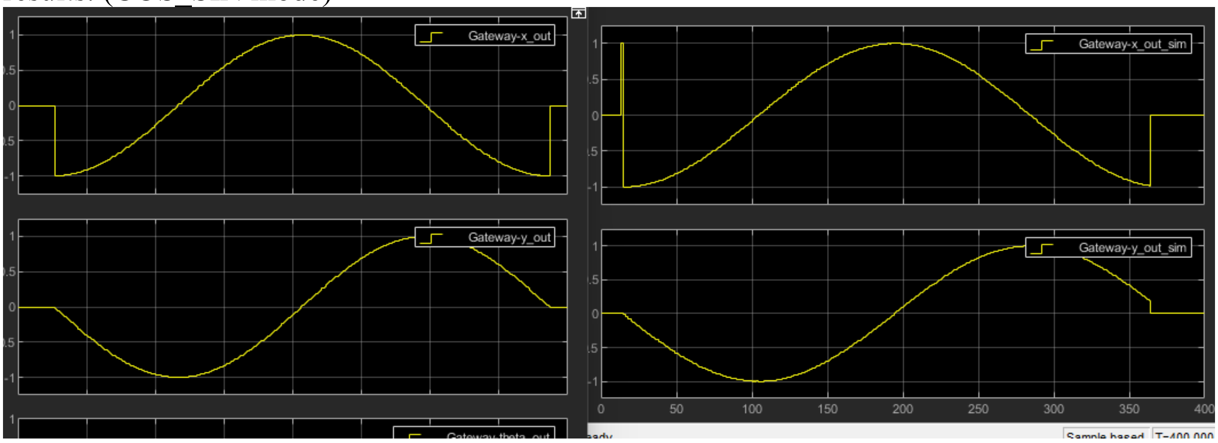


Figure 11.

(c) Screen capture of CORDIC Vitis HLS block and CORDIC block scope output results. (ARCTAN mode)

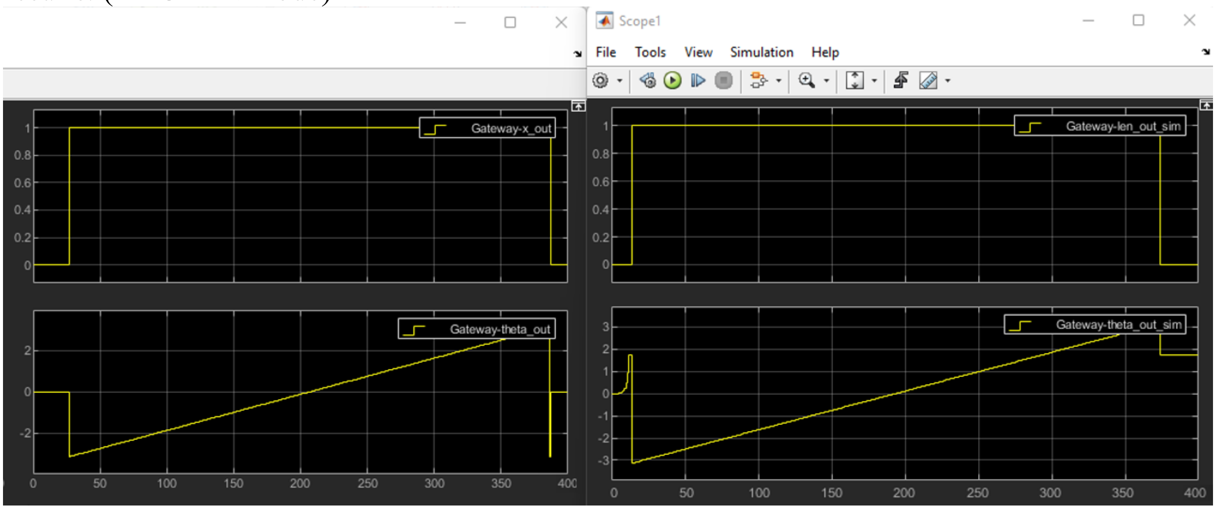


Figure 12.

(d) Screen capture of visualize CORDIC circular mode results. (COS_SIN mode)

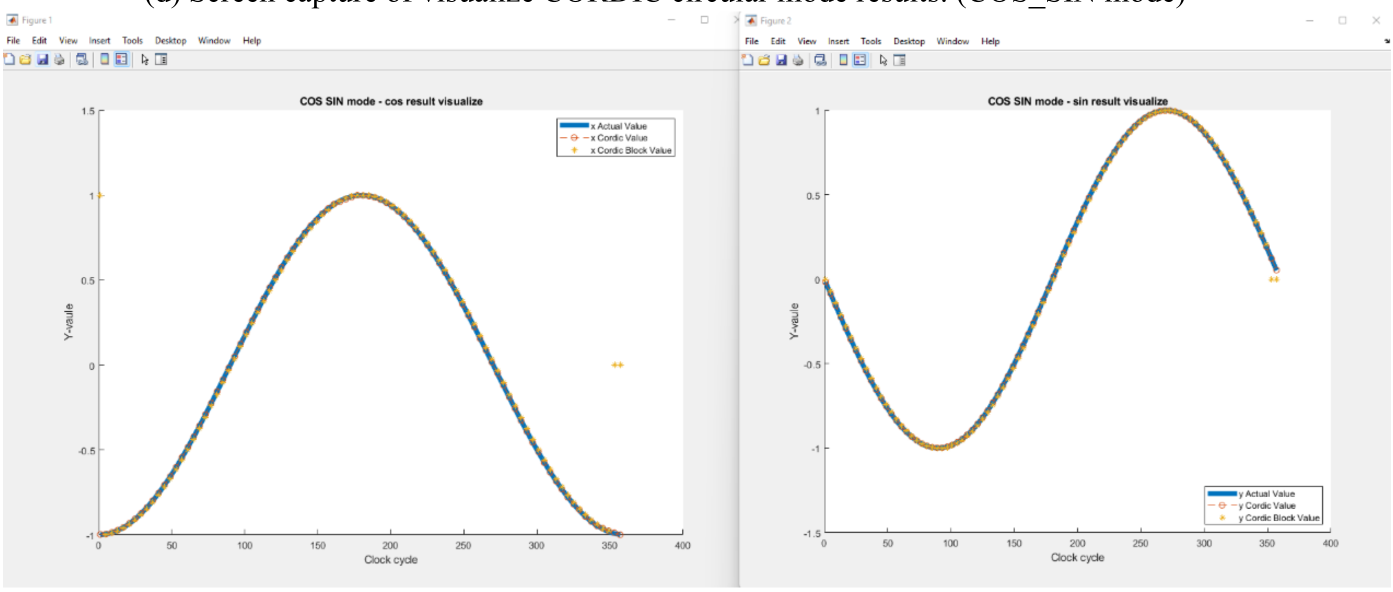


Figure 13.

(e) Screen capture of visualize CORDIC circular mode results. (ARCTAN mode)

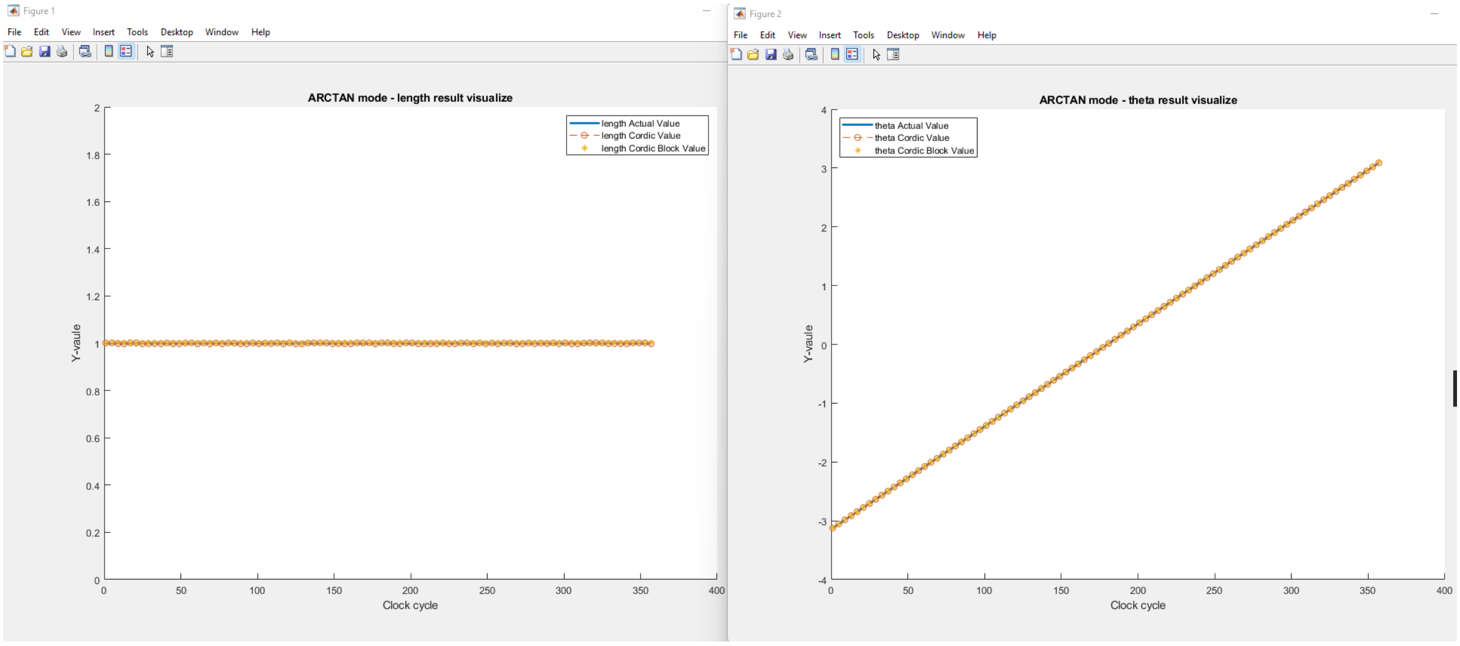
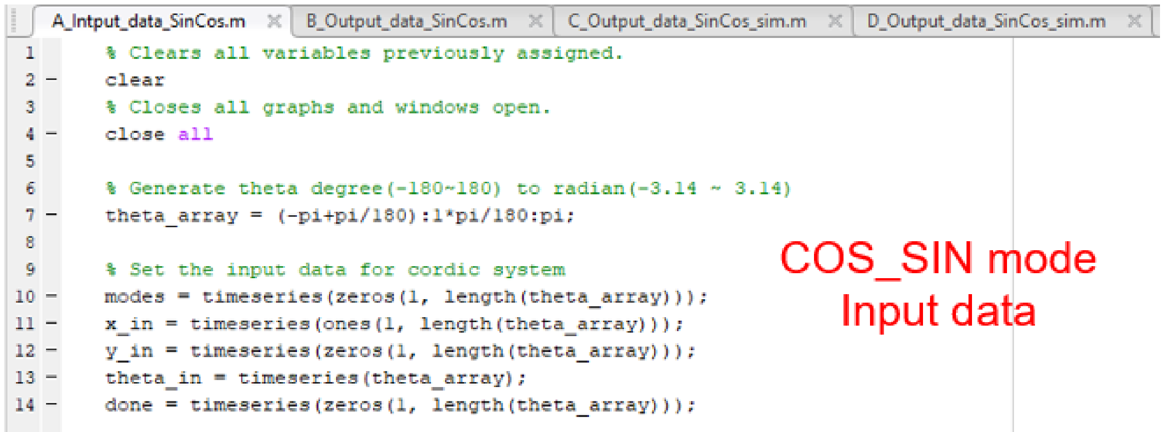


Figure 14.

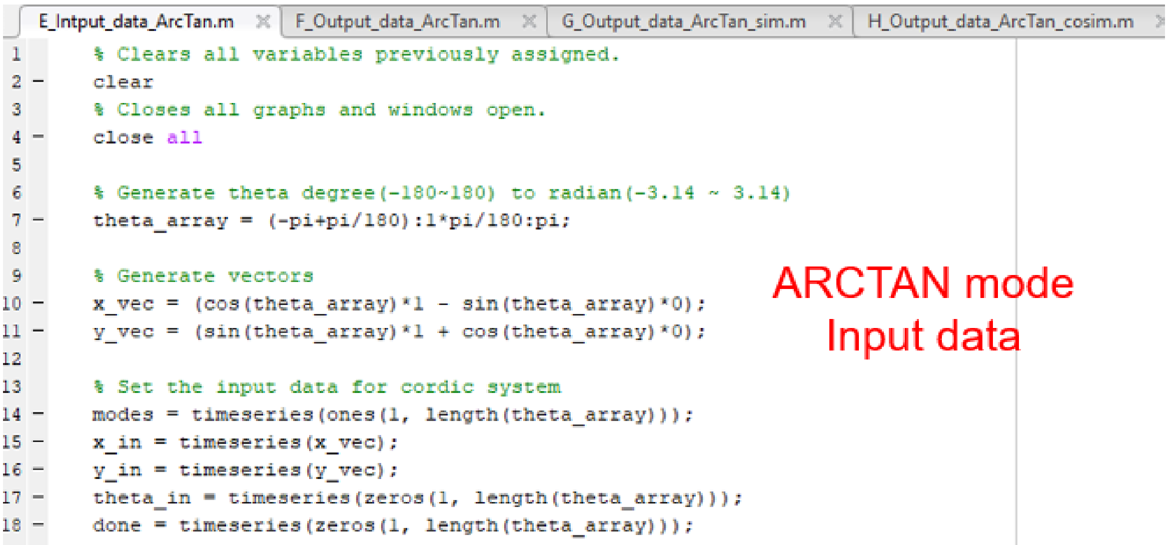
(f) Screen capture of input data m_code for COS_SIN mode and ARCTAN mode.




```
% Clears all variables previously assigned.
clear
% Closes all graphs and windows open.
close all

% Generate theta degree(-180~180) to radian(-3.14 ~ 3.14)
theta_array = (-pi+pi/180):1*pi/180:pi;

% Set the input data for cordic system
modes = timeseries(zeros(1, length(theta_array)));
x_in = timeseries(ones(1, length(theta_array)));
y_in = timeseries(zeros(1, length(theta_array)));
theta_in = timeseries(theta_array);
done = timeseries(zeros(1, length(theta_array)));
```




```
% Clears all variables previously assigned.
clear
% Closes all graphs and windows open.
close all

% Generate theta degree(-180~180) to radian(-3.14 ~ 3.14)
theta_array = (-pi+pi/180):1*pi/180:pi;

% Generate vectors
x_vec = (cos(theta_array)*1 - sin(theta_array)*0);
y_vec = (sin(theta_array)*1 + cos(theta_array)*0);

% Set the input data for cordic system
modes = timeseries(ones(1, length(theta_array)));
x_in = timeseries(x_vec);
y_in = timeseries(y_vec);
theta_in = timeseries(zeros(1, length(theta_array)));
done = timeseries(zeros(1, length(theta_array)));
```

Figure 15.

(g) Screen capture of output data m_code for COS_SIN mode.

A_Intput_data_SinCos.m | B_Output_data_SinCos.m | C_Output_data_SinCos_sim.m | D_Output_data_SinCos_sim.m

```
% Generate theta degree(-180~180) to radian(-3.14 ~ 3.14)
theta_array = (-pi+pi/180):1*pi/180:pi;

% Calculate true rotation for each angle
x_sw = cos(theta_array)*1 - sin(theta_array)*0;
y_sw = sin(theta_array)*1 + cos(theta_array)*0;

x_hw = out.x_out';
y_hw = out.y_out';

x_sim = out.x_out_sim';
y_sim = out.y_out_sim';

% Visualize the results
time = 1:4:(length(theta_array)); % 360 data are too dense to visualize, so here sampled to 90 data
figure()
hold on
plot(time, x_sw(1:4:(length(theta_array))),'LineWidth', 5, 'Color', '#0072BD');
plot(time, x_hw(28:4:(length(theta_array)+27)), '--o', 'Color', '#D95319');
plot(time, x_sim(15:4:(length(theta_array)+14)), '*', 'Color', '#EDB120');
hold off
legend('x Actual Value', 'x Cordic Value', 'x Cordic Block Value')
title('COS SIN mode - cos result visualize')
xlabel('Clock cycle')
ylabel('Y-vaule')

figure()
hold on
plot(time, y_sw(1:4:(length(theta_array))), 'LineWidth', 5, 'Color', '#0072BD');
plot(time, y_hw(28:4:(length(theta_array)+27)), '--o', 'Color', '#D95319');
plot(time, y_sim(15:4:(length(theta_array)+14)), '*', 'Color', '#EDB120');
hold off
legend('y Actual Value', 'y Cordic Value', 'y Cordic Block Value',...
    'Location', 'southeast')
title('COS SIN mode - sin result visualize')
xlabel('Clock cycle')
ylabel('Y-vaule')
```

Figure 16.

(h) Screen capture of output data m_code for ARCTAN mode.

E_Intput_data_ArcTan.m | F_Output_data_ArcTan.m | G_Output_data_ArcTan_sim.m | H_Output_data_ArcTan_cosim.m

```
% Generate theta degree(-180~180) to radian(-3.14 ~ 3.14)
theta_array = (-pi+pi/180):1*pi/180:pi;

% Generate vectors
x_vec = (cos(theta_array)*1 - sin(theta_array)*0);
y_vec = (sin(theta_array)*1 + cos(theta_array)*0);

len_hw = out.x_out';
theta_hw = out.theta_out';

len_sw = sqrt(x_vec.^2+y_vec.^2);
theta_sw = atan2(y_vec, x_vec);

len_sim = out.len_out_sim';
theta_sim = out.theta_out_sim';

% Visualize the results
time = 1:4:(length(theta_array)); % 360 data are too dense to visualize, so here sampled to 90 data
figure()
hold on
plot(time, len_sw(1:4:(length(theta_array))), 'LineWidth', 2, 'Color', '#0072BD');
plot(time, len_hw(28:4:(length(theta_array)+27)), '--o', 'Color', '#D95319');
plot(time, len_sim(15:4:(length(theta_array)+14)), '*', 'Color', '#EDB120');
hold off
legend('length Actual Value', 'length Cordic Value', 'length Cordic Block Value')
title('ARCTAN mode - length result visualize')
xlabel('Clock cycle')
ylabel('Y-vaule')
xlim([0 400])
ylim([0 2])

figure()
hold on
plot(time, theta_sw(1:4:(length(theta_array))), 'LineWidth', 2, 'Color', '#0072BD');
plot(time, theta_hw(28:4:(length(theta_array)+27)), '--o', 'Color', '#D95319');
plot(time, theta_sim(15:4:(length(theta_array)+14)), '*', 'Color', '#EDB120');
hold off
legend('theta Actual Value', 'theta Cordic Value', 'theta Cordic Block Value',...
    'Location', 'northwest')
title('ARCTAN mode - theta result visualize')
xlabel('Clock cycle')
ylabel('Y-vaule')
```

Figure 17.

Model Composer: Hardware co-simulation: (SysGen)

Co-Simulation CORDIC circular mode in model composer:

After successfully generate the system in JTAG compilation mode, I connect the input and output into cordic_cosim block, to simulate the system in Zedboard as shown in *Figure 18*, and *Figure 19* is the resources analyzer result, I also show the scope output in *Figure 20* (COS_SIN mode) and *Figure 21* (ARCTAN mode), visualized the output data from workspace in *Figure 22* (COS_SIN mode) and *Figure 23* (ARCTAN mode).

(a) Screen capture of co-simulation CORDIC circular mode system architecture (Model composer)

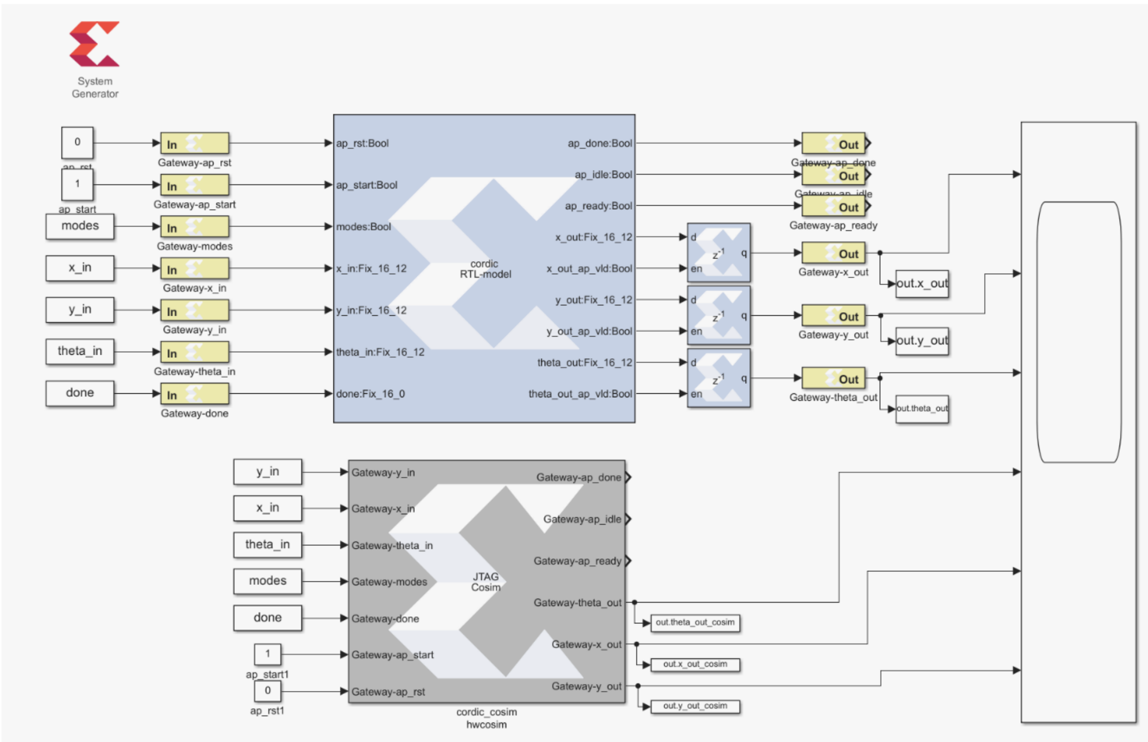


Figure 18.

(b) Screen capture of co-simulation resources analyzer.

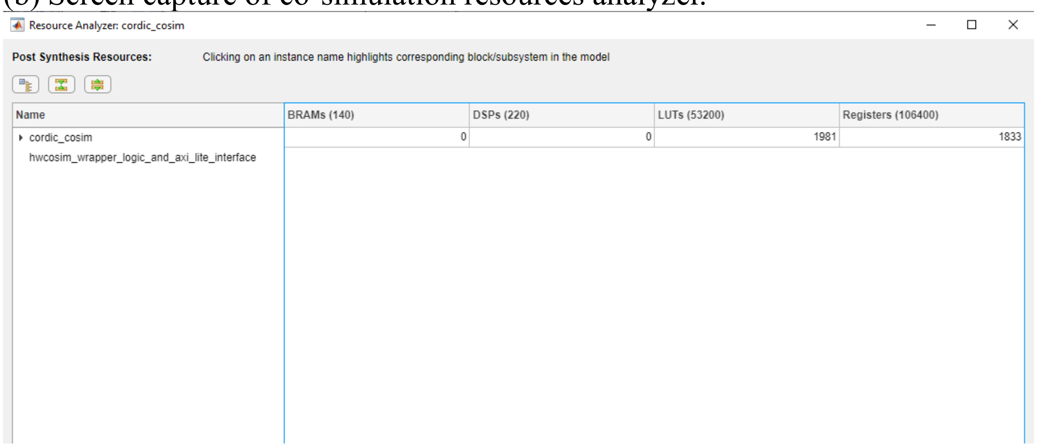


Figure 19.

(c) Screen capture of CORDIC Vitis HLS block and CORDIC co-sim block scope output results. (COS_SIN mode)

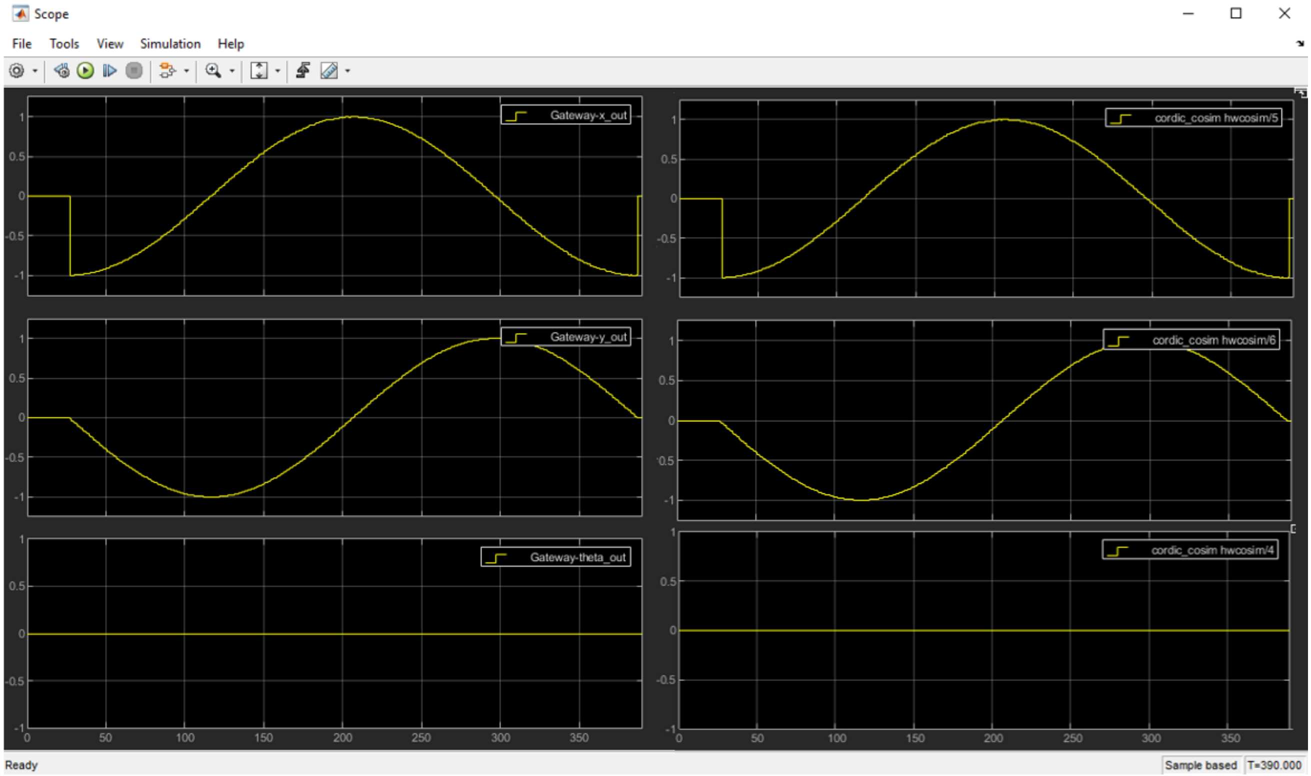


Figure 20.

(d) Screen capture of CORDIC Vitis HLS block and CORDIC co-sim block scope output results. (ARCTAN mode)

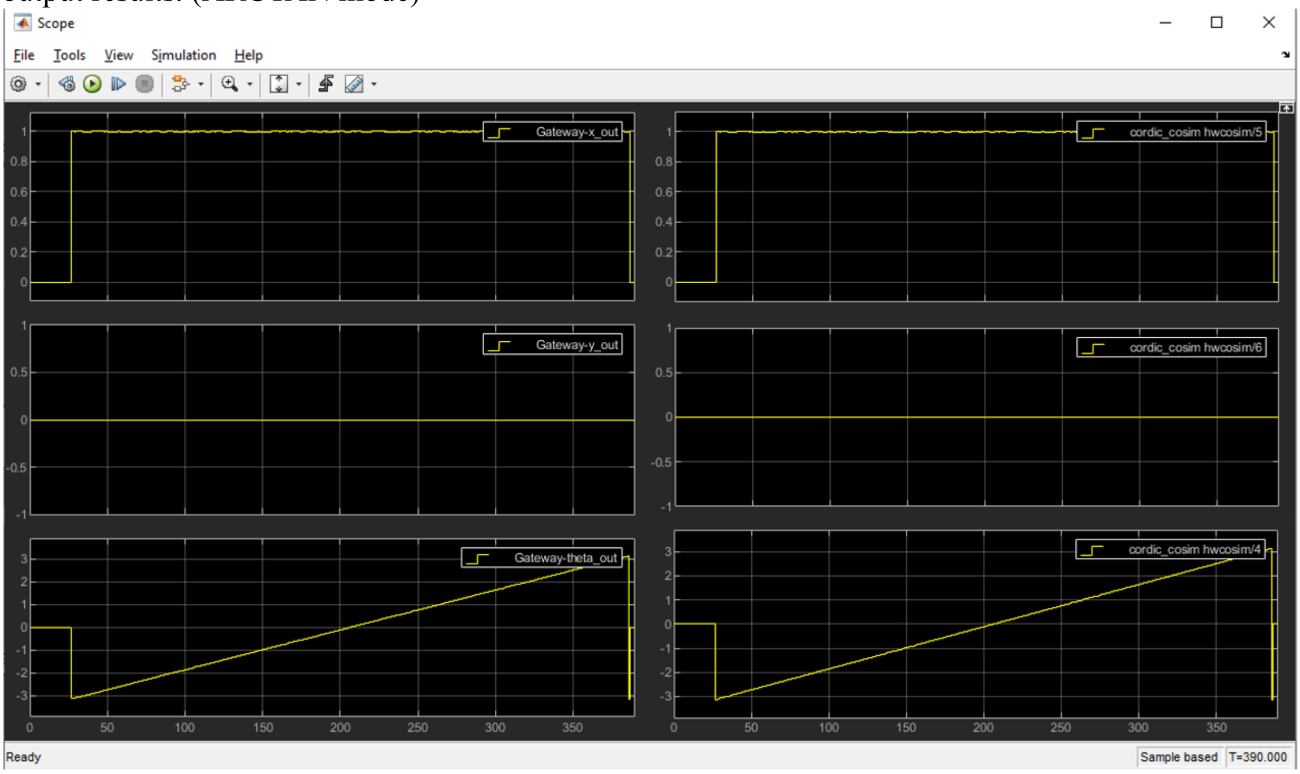


Figure 21.

(e) Screen capture of visualize CORDIC circular mode results. (COS_SIN mode)

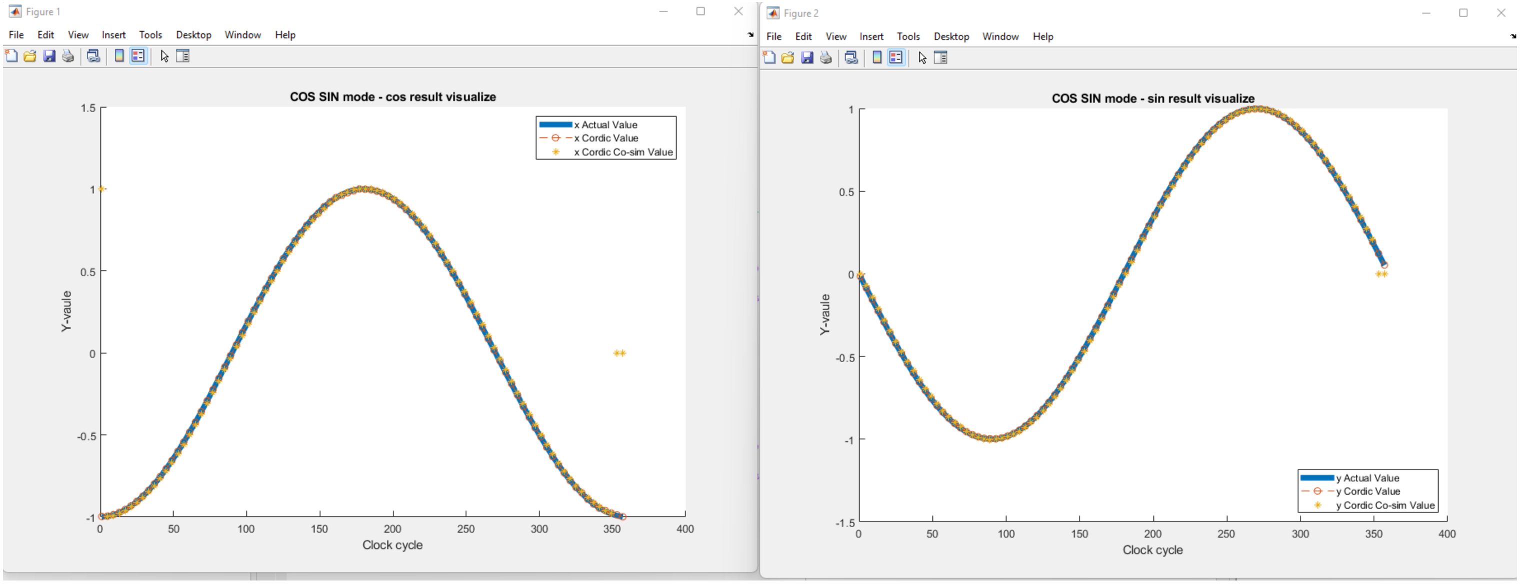


Figure 22.

(f) Screen capture of visualize CORDIC circular mode results. (ARCTAN mode)

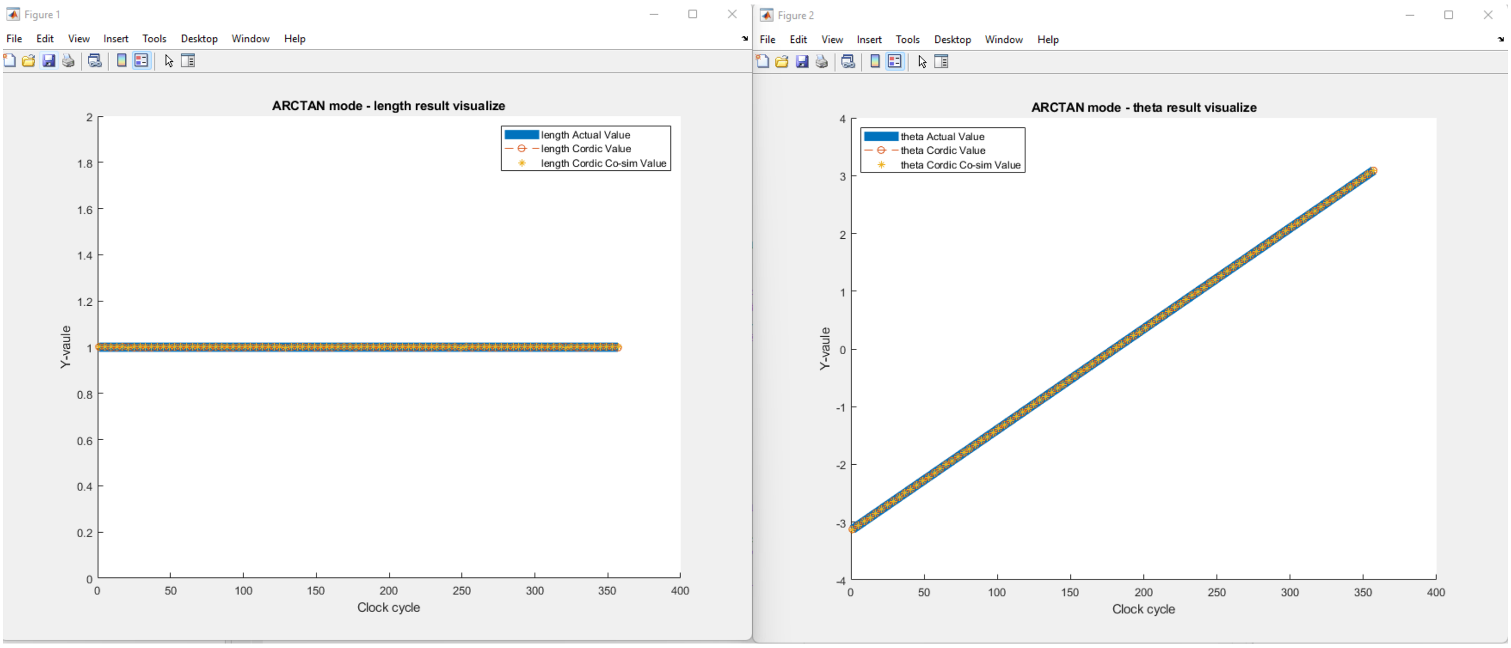


Figure 23.

Vivado:

CORDIC circular mode Vivado block design:

In this section, basically we just import the RTL IP design which exported from Vitis HLS, and when we export Vitis HLS we need to include AXILITE interfaces as mentioned in solution3, and integrated the IP package with the ARM core in Vivado as shown in Figure 24, then verified and wrapped the design, after that synthesis the design and run implementation, finally generated bitstream, exported the hardware to Vitis IDE. Also the *Figure 25* is my cordic design utilization and timing results table.

(a) Screen capture of Vivado block design including cordic IP block and ARM core.

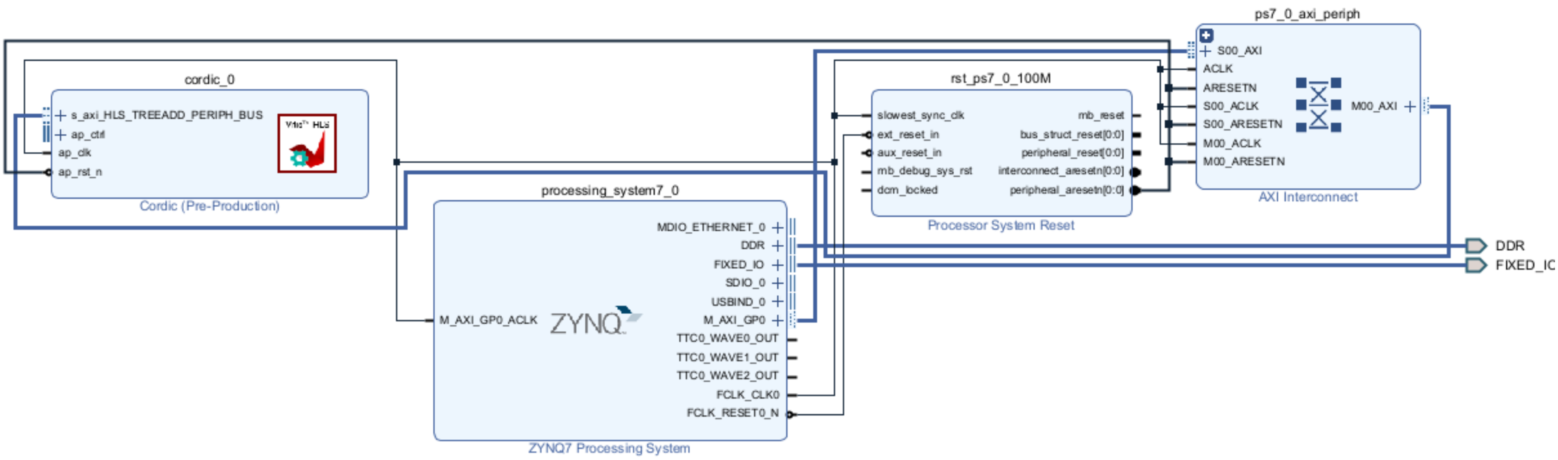


Figure 24.

(b) Screen capture of cordic's Vivado synthesis and implementation utilization and timing results.

Tcl Console | Messages | Log | Reports | Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	Methodology	RQA Score	QoR Suggestions	LUT	FF	BRAM	URAM	DSP
synth_1 (active)	**constrs_1**	**synth_design Complete!**												**0**	**0**	**0**	**0**	**0**
impl_1	constrs_1	write_bitstream Complete!	4.112	0.000	0.044	0.000		0.000	1.705	0				441	522	0	0	0
Out-of-Context Module Runs																		
design_1		Submodule Runs Complete																

Figure 25.

Vitis IDE: ARM program control

CORDIC circular mode ARM program control in Vitis IDE:

In this section, I modify the main.cc file from Project_4_tree_fixed_HLS_Vitis code and include the xcordic.h header file which generated from Vivado, and successfully build the program to ARM and connected to Zedboard serial port, then debug the program, and final result as shown in *Figure 28*.

(a) Screen capture of Vitis IDE main.cc ARM code.

```
#include <cmath>
#include <iostream>
#include "xcordic.h"
#include "ap_fixed.h"
#include "cordic.h"
XCordic Cordic;
#include "xparameters.h"
#include "xtime_l.h"
#include "xscugic.h"

using namespace std;

typedef ap_fixed<16,4> FIXED_TYPE;

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));
}
#define pi 3.1415926

int main()
{
    cout << "--- Start of the Program ---" << endl;

    FIXED_TYPE modes_in = 1;
    FIXED_TYPE x_in = 1.0;
    FIXED_TYPE y_in = 0.0;
    FIXED_TYPE theta_in = (THETA_TYPE) (45.0 * pi / 180 ); //test for 45 degree

    unsigned int modes_in_u32, x_u32, y_u32, theta_u32, x_out_u32, y_out_u32, theta_out_u32;
    int done_out = 0;
    THETA_TYPE x_out, y_out, theta_out;

// models = 0 -> COS_SIN mode
    modes_in_u32 = 0;
    x_u32 = get_int_reinterpret(x_in);
    y_u32 = get_int_reinterpret(y_in);
    theta_u32 = get_int_reinterpret(theta_in);
    cout << "Initialized for Software simulation: " << "modes=" << modes_in_u32 << ", " << "x=" << x_u32 << ", " << "y=" << y_u32 << ", " << "theta=" << theta_u32  << endl;
    XCordic_Initialize(&Cordic, 0);
    XCordic_Set_modes(&Cordic, modes_in_u32);
    XCordic_Set_x_in(&Cordic, x_u32);
    XCordic_Set_y_in(&Cordic, y_u32);
    XCordic_Set_theta_in(&Cordic, theta_u32);
    XCordic_Start(&Cordic);
    while (!XCordic_IsReady(&Cordic));
    done_out = XCordic_Get_done(&Cordic);
    x_out_u32 = XCordic_Get_x_out(&Cordic);
    y_out_u32 = XCordic_Get_y_out(&Cordic);
    theta_out_u32 = XCordic_Get_theta_out(&Cordic);
    cout << "Done signal from ARM hardware = " << done_out << endl;
    x_out = get_fixed_reinterpret(x_out_u32);
    y_out = get_fixed_reinterpret(y_out_u32);
    cout << "Hardware result after ARM calculation (COS_SIN modes): " << "x_out =" << x_out_u << ", " << "y_out=" << y_out_u << ", " << "theta_out =" << theta_out_u32 << endl;

//-----------------------------------------------------------------------------------------------------------------

    FIXED_TYPE modes_in = 1;
    FIXED_TYPE x_in = 0.258973;
    FIXED_TYPE y_in = -0.965885;
    FIXED_TYPE theta_in = (THETA_TYPE) (-75.0 * pi / 180 ); //test for -75 degree

// models = 1 -> ARCTAN mode
    modes_in_u32 = 1;
    x_u32 = get_int_reinterpret(x_in);
    y_u32 = get_int_reinterpret(y_in);
    theta_u32 = get_int_reinterpret(theta_in);
    cout << "Initialized for Software simulation: " << "modes=" << modes_in_u32 << ", " << "x=" << x_u32 << ", " << "y=" << y_u32 << ", " << "theta=" << theta_u32  << endl;
    XCordic_Initialize(&Cordic, 0);
    XCordic_Set_modes_r(&Cordic, modes_in_u32);
    XCordic_Set_x_in(&Cordic, x_u32);
    XCordic_Set_y_in(&Cordic, y_u32);
    XCordic_Set_theta_in(&Cordic, theta_u32);
    XCordic_Start(&Cordic);
    while (!XCordic_IsReady(&Cordic));
    done_out = XCordic_Get_done(&Cordic);
    x_out_u32 = XCordic_Get_x_out(&Cordic);
    y_out_u32 = XCordic_Get_y_out(&Cordic);
    theta_out_u32 = XCordic_Get_theta_out(&Cordic);
    cout << "Done signal from ARM hardware = " << done_out << endl;
    theta_out = get_fixed_reinterpret(theta_out_u32);
    cout << "Hardware result after ARM calculation (ARCTAN modes): " << "x_out =" << x_out_u << ", " << "y_out=" << y_out_u << ", " << "theta_out =" << theta_out_u32 << endl;

    cout << "--- End of the Program ---" << endl;

    return 0;
}
```

Figure 26.

(b) Screen capture of successful build the program.

Console ⊠ | Problems | Vitis Log | Guidance

Build Console [project4_v5_vitis, Debug]

```
'Finished building: ../src/main.cc'
' '
'Building target: project4_v5_vitis.elf'
'Invoking: ARM v7 g++ linker'
arm-none-eabi-g++ -mcpu=cortex-a9 -mfpu=vfpv3 -mfloat-abi=hard -Wl,-build-id=none -specs=Xilinx
'Finished building target: project4_v5_vitis.elf'
' '
'Invoking: ARM v7 Print Size'
arm-none-eabi-size project4_v5_vitis.elf  |tee "project4_v5_vitis.elf.size"
   text    data     bss     dec     hex filename
 655304    2920   29160  687384   a7d18 project4_v5_vitis.elf
'Finished building: project4_v5_vitis.elf.size'
' '

10:40:43 Build Finished (took 2s.719ms)
```

Figure 27.

(c) Screen capture of successfully connect to Zedboard and debug the program.

Console | Vitis Serial Terminal ⊠ | Executables | Debug Shell | Vitis Log | Problems | Debugger Console

Connected to: Serial (COM4, 115200, 0, 8)

```
Already connected to port: COM4--- Start of the Program ---

Initialized for Software simulation: modes=0, x= 1, y=0, theta=0.785156

Done signal from ARM hardware = 1
                                          <Expected x= 0.707278, y = 0.706936>
Hardware result after ARM calculation (COS_SIN modes):x_out =0.705566, y_out=0.705566, theta_out =0

Initialized for Software simulation: modes=1, x=0.258973, y=-0.965885, theta=0

Done signal from ARM hardware = 1
                                          <Expected length = 1, theta = -1.308838>
Hardware result after ARM calculation (ARCTAN modes): x_out =0.998291, y=-0.965885, theta_out =-1.30933

--- End of the Program ---                 X_out = length
```

Figure 28.