2025

CORDIC Algorithm

BY: SHEHAB ELDEEN KHALED MABROUK NOAH 2100422

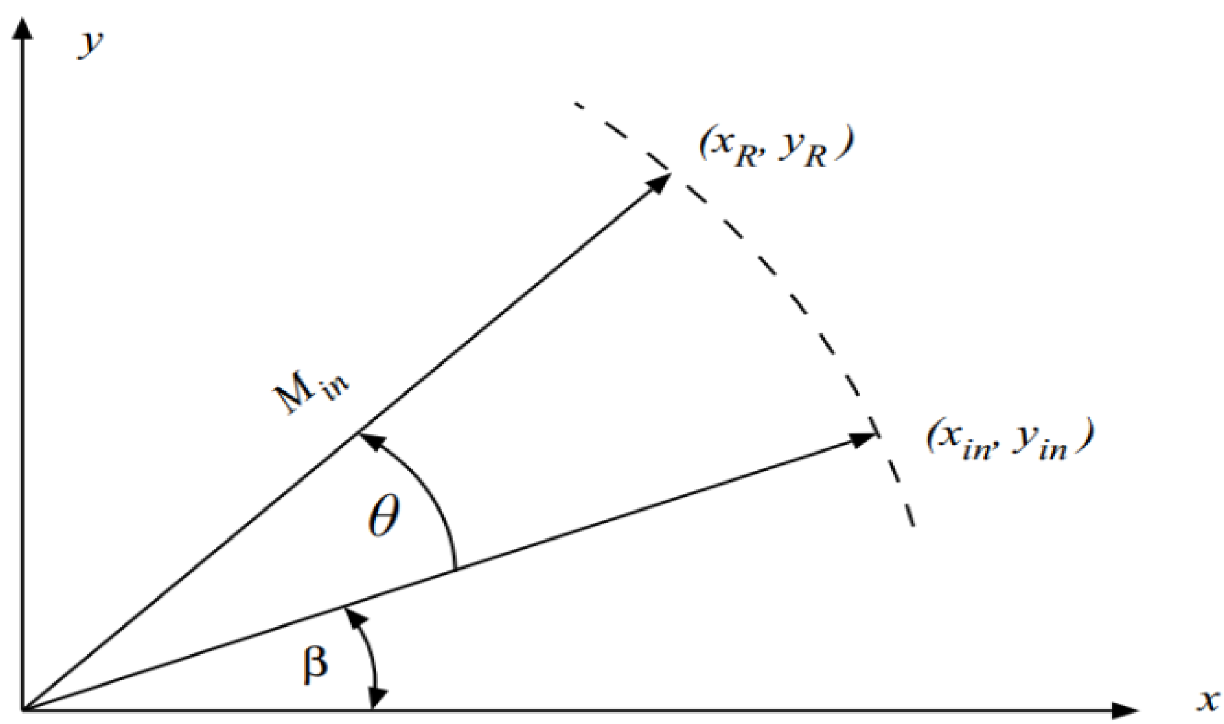
CHIPIONS | IEEE SSCS

Introduction:

The CORDIC **(Coordinate Rotation Digital Computer)** algorithm computes trigonometric (and other) functions using only **additions**, **subtractions**, **bit-shifts** and a **small lookup table of arctangents**. Because it avoids general multipliers, CORDIC is highly attractive in FPGA/ASIC implementations. In this project we implement a fixed-point CORDIC in Verilog to compute sine and cosine and provide a MATLAB fixed-point reference model for self-checking.

Project Idea:

To implement a hardware-efficient CORDIC engine to compute sine and cosine from an angle input expressed in fixed-point arithmetic format. The goal is a fully synthesizable Verilog design that accepts angles across an extended range **(−2π … +4π)** using a small lookup table and only shifters/adders — so it fits well on FPGAs where multipliers are costly. A quadrant mapper normalizes input angles into the CORDIC's convergence range and supplies sign flags, so the CORDIC core always works on a small angle.



- ***RTL Design:***

1) Quadrature Logic Mapper Module:

```
module quad_logic (
    input signed [31:0] angle_in,              // Q5.27 to take the range [-2*pi to 4*pi]
    output reg signed [31:0] angle_cordic_out, // Q5.27 mapped to [-pi/2, +pi/2]
    output reg sin_sign, cos_sign             // 1 means negative
);
// Q5.27 constants
localparam signed [31:0] PI = 32'h1921FB54;           // pi
localparam signed [31:0] PI_2 = 32'h0C90FDAA;         // pi/2
localparam signed [31:0] TWO_PI = 32'h3243F6A9;       // 2*pi
localparam signed [31:0] THREE_PI_2 = 32'h25B2F8FE;   // 3*pi/2
reg signed [31:0] angle_norm;
always @(*) begin
    // Normalize the input angle to be in [0:2*pi]
    if (angle_in < 0) begin
        angle_norm = angle_in + TWO_PI;
    end
    else if (angle_in >= TWO_PI) begin
        angle_norm = angle_in - TWO_PI;
    end
    else begin
        angle_norm = angle_in;
    end
    // Map the normalized angle to be in [-pi/2:pi/2]
    if (angle_norm <= PI_2) begin  // 1st quad (0 to 90°)
        angle_cordic_out = angle_norm;
        sin_sign = 1'b0;  // sin positive
        cos_sign = 1'b0;  // cos positive
    end
    else if (angle_norm <= PI) begin  // 2nd quad (90° to 180°)
        angle_cordic_out = PI - angle_norm;
        sin_sign = 1'b0;  // sin positive
        cos_sign = 1'b1;  // cos negative
    end
    else if (angle_norm <= THREE_PI_2) begin  // 3rd quad (180° to 270°)
        angle_cordic_out = angle_norm - PI;
        sin_sign = 1'b1;  // sin negative
        cos_sign = 1'b1;  // cos negative
    end
    else begin  // 4th quad (270° to 360°)
        angle_cordic_out = TWO_PI - angle_norm;
        sin_sign = 1'b1;  // sin negative
        cos_sign = 1'b0;  // cos positive
    end
end
endmodule
```

2) CORDIC Algorithm Module:

```
module CORDIC #(parameter WIDTH = 16 , parameter ITERATIONS = 32) (
    input clk,                                // System Clock
    input signed [WIDTH-1:0] x_start,         // Q1.15 (signed) ==> 1-bit for sign , and 15-
bits for fraction
    input signed [WIDTH-1:0] y_start,         // Q1.15 (signed) ==> 1-bit for sign , and 15-
bits for fraction
    input signed [31:0] angle_after_map,      // Q5.27 (signed) 1-bit for sign ,4-bits for
integer part, and 27-bits for fraction part to take the range [-2*pi to 4*pi]
    input sin_sign, cos_sign,                 // 1 ==> negative
    output reg signed [WIDTH-1:0] sine,       // Q1.15 (signed) ==> 1-bit for sign , and 15-
bits for fraction
    output reg signed [WIDTH-1:0] cosine      // Q1.15 (signed) ==> 1-bit for sign , and 15-
bits for fraction
);

// Use larger internal registers to maintain precision
reg signed [31:0] x [0:ITERATIONS];
reg signed [31:0] y [0:ITERATIONS];
reg signed [31:0] z [0:ITERATIONS];

integer i;

// arctan table Q5.27
localparam signed [31:0] atan_table [0:31] = '{
    32'h06487ED5, // i=0,  atan=0.785398 rad = 45.000000 deg
    32'h03B58CE1, // i=1,  atan=0.463648 rad = 26.565051 deg
    32'h01F5B760, // i=2,  atan=0.244979 rad = 14.036243 deg
    32'h00FEADD5, // i=3,  atan=0.124355 rad =  7.125016 deg
    32'h007FD56F, // i=4,  atan=0.062419 rad =  3.576334 deg
    32'h003FFAAB, // i=5,  atan=0.031240 rad =  1.789911 deg
    32'h001FFF55, // i=6,  atan=0.015624 rad =  0.895174 deg
    32'h000FFFEB, // i=7,  atan=0.007812 rad =  0.447614 deg
    32'h0007FFFD, // i=8,  atan=0.003906 rad =  0.223811 deg
    32'h00040000, // i=9,  atan=0.001953 rad =  0.111906 deg
    32'h00020000, // i=10, atan=0.000977 rad =  0.055953 deg
    32'h00010000, // i=11, atan=0.000489 rad =  0.027976 deg
    32'h00008000, // i=12, atan=0.000244 rad =  0.013988 deg
    32'h00004000, // i=13, atan=0.000122 rad =  0.006994 deg
    32'h00002000, // i=14, atan=0.000061 rad =  0.003497 deg
    32'h00001000, // i=15, atan=0.000031 rad =  0.001749 deg
    32'h00000800, // i=16, atan=0.000015 rad =  0.000874 deg
    32'h00000400, // i=17, atan=0.000008 rad =  0.000437 deg
    32'h00000200, // i=18, atan=0.000004 rad =  0.000219 deg
    32'h00000100, // i=19, atan=0.000002 rad =  0.000109 deg
    32'h00000080, // i=20, atan=0.000001 rad =  0.000055 deg
    32'h00000040, // i=21, atan=0.000000 rad =  0.000027 deg
```

```verilog
    32'h00000020, // i=22, atan=0.000000 rad =  0.000014 deg
    32'h00000010, // i=23, atan=0.000000 rad =  0.000007 deg
    32'h00000008, // i=24, atan=0.000000 rad =  0.000003 deg
    32'h00000004, // i=25, atan=0.000000 rad =  0.000002 deg
    32'h00000002, // i=26, atan=0.000000 rad =  0.000001 deg
    32'h00000001, // i=27, atan=0.000000 rad =  0.000000 deg
    32'h00000000, // i=28, atan=0.000000 rad =  0.000000 deg
    32'h00000000, // i=29, atan=0.000000 rad =  0.000000 deg
    32'h00000000, // i=30, atan=0.000000 rad =  0.000000 deg
    32'h00000000  // i=31, atan=0.000000 rad =  0.000000 deg
};

always @(posedge clk) begin
    // Initialize with extended precision (convert Q1.15 to Q1.30 equivalent)
    // Sign extend and shift left by 15 bits to increase accuracy
    x[0] <= {{1{x_start[WIDTH-1]}}, x_start, 15'b0};
    y[0] <= {{1{y_start[WIDTH-1]}}, y_start, 15'b0};
    z[0] <= angle_after_map;

    // CORDIC iterations
    for (i = 0; i < ITERATIONS; i = i + 1) begin
        if (z[i][31] == 1'b1) begin     // negative angle
            x[i+1] <= x[i] + (y[i] >>> i);
            y[i+1] <= y[i] - (x[i] >>> i);
            z[i+1] <= z[i] + atan_table[i];
        end
        else begin                      // positive angle
            x[i+1] <= x[i] - (y[i] >>> i);
            y[i+1] <= y[i] + (x[i] >>> i);
            z[i+1] <= z[i] - atan_table[i];
        end
    end

    // Final output
    // Convert from extended precision back to Q1.15 by taking bits [30:15] ==> bit[30] is
the integer bit and rest is fraction
    if (sin_sign) begin
        sine <= -y[ITERATIONS][30:15];  // we doesn't take the bit[31] , as we handle the
sign separately
    end else begin
        sine <= y[ITERATIONS][30:15];
    end

    if (cos_sign) begin
        cosine <= -x[ITERATIONS][30:15];
    end else begin
        cosine <= x[ITERATIONS][30:15];
    end
end
endmodule
```

3) <u>Top Module:</u>

```
module CORDIC_top #(parameter WIDTH = 16 , parameter ITERATIONS = 32) (
    input clk,
    input signed [WIDTH-1:0] x_start,
    input signed [WIDTH-1:0] y_start,
    input signed [31:0] angle_in,
    output signed [WIDTH-1:0] sine,
    output signed [WIDTH-1:0] cosine
);

wire signed [31:0] angle_cordic_out;
wire sin_sign, cos_sign;

quad_logic quad_logic_DUT(
    .angle_in(angle_in),
    .angle_cordic_out(angle_cordic_out),
    .sin_sign(sin_sign),
    .cos_sign(cos_sign)
);

CORDIC #(.WIDTH(WIDTH), .ITERATIONS(ITERATIONS)) CORDIC_DUT(
    .clk(clk),
    .x_start(x_start),
    .y_start(y_start),
    .angle_after_map(angle_cordic_out),
    .sin_sign(sin_sign),
    .cos_sign(cos_sign),
    .sine(sine),
    .cosine(cosine)
);

endmodule
```

- ## ***Self-Checking Testbench:***

```
module CORDIC_tb ();
    // Signal Declaration
    parameter WIDTH = 16;
    reg clk;
    reg signed [WIDTH-1:0] x_start , y_start;  // Q1.15 (signed) ==> 1bit for sign , and rest for fractional
    reg signed [31:0] angle_in;                // Q5.27 (signed) 1bit for sign ,4bits for integer part, and the rest for fraction part
    wire signed [WIDTH-1:0] sine , cosine;     // Q1.15 (signed) represent sin and cos [-1:1] ==> 1bit for sign , and rest for fractional
    real sine_real, cosine_real;
    real angle_rad, angle_deg;

    localparam signed [WIDTH-1:0] Kn = 16'h4DBA; // Q1.15 (signed) Kn = 0.6072529351

    // Q5.27 constants for angle generation
    localparam signed [31:0] PI = 32'h1921FB54;           // pi
    localparam signed [31:0] TWO_PI = 32'h3243F6A9;       // 2*pi
    localparam signed [31:0] NEG_TWO_PI = 32'hCDBC0958;   // -2*pi
    localparam signed [31:0] FOUR_PI = 32'h6487ED52;      // 4*pi
    integer i = 0;
    integer pos_angles [0:18];   // to store random angles

    // File handling variables
    integer matlab_file;
    integer file_status;
    integer test_case_count = 0;

    // MATLAB reference results storage and compare it with verilog results
    real matlab_sine_real [0:50];  // Store up to 50 test cases
    real matlab_cosine_real [0:50];

    // Error calculation variables
    real sine_error, cosine_error;
    real sine_error_percent, cosine_error_percent;
    real max_sine_error = 0, max_cosine_error = 0;

    // module instantiation
    CORDIC_top DUT_top(clk,x_start,y_start,angle_in,sine,cosine);

    // Clock generation
    initial begin
        clk = 0;
        forever begin
            #2 clk = ~clk;
        end
    end
```

```
    // Read MATLAB outputs file
    initial begin
        matlab_file = $fopen("MATLAB_outputs.txt", "r");
        if (matlab_file == 0) begin
            $display("Error: Could not open MATLAB_outputs.txt file!");
            $stop;
        end

        $display("\nReading MATLAB reference results");

        // Read all test cases from file
        test_case_count = 0;
        while (!$feof(matlab_file) && test_case_count < 50) begin
            file_status = $fscanf(matlab_file, "%f %f",
                matlab_sine_real[test_case_count],
                matlab_cosine_real[test_case_count]);

            if (file_status == 2) begin // Successfully read 2 values
                test_case_count = test_case_count + 1;
            end
        end
        $fclose(matlab_file);
        $display("Read %0d test cases from MATLAB_outputs.txt", test_case_count);
    end

    // Task to run a test case and compare with MATLAB
    task run_test_case;
        input [31:0] angle;
        input integer matlab_index;
        begin
            angle_in = angle;
            #150; // Wait for CORDIC computation

            // Calculate errors
            sine_error = sine_real - matlab_sine_real[matlab_index];
            cosine_error = cosine_real - matlab_cosine_real[matlab_index];

            // Calculate error percentage
            if (matlab_sine_real[matlab_index] != 0) begin
                sine_error_percent = (sine_error / matlab_sine_real[matlab_index]) * 100;
            end else begin
                sine_error_percent = (sine_error != 0) ? 999.9 : 0.0;
            end

            if (matlab_cosine_real[matlab_index] != 0) begin
                cosine_error_percent = (cosine_error / matlab_cosine_real[matlab_index]) *
100;
            end else begin
                cosine_error_percent = (cosine_error != 0) ? 999.9 : 0.0;
            end
```

```
            // Update error statistics
            if (sine_error_percent < 0) sine_error_percent = -sine_error_percent;
            if (cosine_error_percent < 0) cosine_error_percent = -cosine_error_percent;

            if (sine_error_percent > max_sine_error) max_sine_error = sine_error_percent;
            if (cosine_error_percent > max_cosine_error) max_cosine_error =
cosine_error_percent;

            $display("%4t %10.1f %10.6f  %12h %10.6f  %10.6f  %11h %11h %10.2f%%
%10.2f%%",
                     $time, angle_deg, angle_rad, angle_in,
                     sine_real, cosine_real, sine, cosine,
                     sine_error_percent, cosine_error_percent);
        end
    endtask

    //Stimulus generation
    initial begin
        x_start = Kn;
        y_start = 0; // always zero

        // Wait for file to be read
        #20;

        $display("\n================================= CORDIC Algorithm Testbench
==========================================\n");
        $display("----------------------------------------- Corner Cases -----------------
-------------------------\n");
        $display("Time Angle(Deg) Angle(Rad)  Angle(Q5.27)
Sine(Real)  Cos(Real)   Sine(Q1.15) Cos(Q1.15) Sine_Err(%%) Cos_Err(%%)");
        $display("---- ---------- ----------  ------------ ----------  ----------  -------
---- ----------- ---------- ----------\n");

        // Initialize error statistics
        max_sine_error = 0;
        max_cosine_error = 0;

        // ------------------------------------Corner Cases-------------------------------
-----------------
        // Test cases must match the order in your MATLAB_outputs.txt file
        run_test_case(32'h00000000, 0);  // 0 deg
        run_test_case(32'h0C90FDAA, 1);  // 90 deg
        run_test_case(32'h1921FB54, 2);  // 180 deg
        run_test_case(32'h25B2F8FF, 3);  // 270 deg
        run_test_case(32'h3243F6A9, 4);  // 360 deg

        // ---------------------------------------------Overflow and Underflow------------
-----------------------------------
```

```
        $display("----------------------------------------- Overflow and Underflow -------
-----------------------------------\n");
        $display("Time Angle(Deg) Angle(Rad)  Angle(Q5.27)
Sine(Real)  Cos(Real)   Sine(Q1.15) Cos(Q1.15) Sine_Err(%%) Cos_Err(%%)");
        $display("---- ---------- ----------  ------------ ----------  ----------  -------
---- ----------- ---------- ----------\n");

        run_test_case(32'h3ED4F453, 5);  // 450 deg
        run_test_case(32'hEF3EADC8, 6);  // -120 deg
        // -------------------------------------Boundary conditions [-2*pi to 4*pi]-------
----------------------------------------
        $display("----------------------------------------- Boundary conditions ----------
--------------------------------\n");
        $display("Time Angle(Deg) Angle(Rad)  Angle(Q5.27)
Sine(Real)  Cos(Real)   Sine(Q1.15) Cos(Q1.15) Sine_Err(%%) Cos_Err(%%)");
        $display("---- ---------- ----------  ------------ ----------  ----------  -------
---- ----------- ---------- ----------\n");

        run_test_case(32'h6487ED51, 7);  // 720 deg
        run_test_case(32'hCDBC0957, 8);  // -360 deg

        // ----------------------------------------------Random Positive Angles-----------
-------------------------------------------
        $display("\n------------------------------------ Random Positive Angles ----------
----------------------------------------\n");
        $display("Time Angle(Deg) Angle(Rad)  Angle(Q5.27)
Sine(Real)  Cos(Real)   Sine(Q1.15) Cos(Q1.15) Sine_Err(%%) Cos_Err(%%)");
        $display("---- ---------- ----------  ------------ ----------  ----------  -------
---- ----------- ---------- ----------\n");

        // Random angles starting from index 9 in MATLAB file
        pos_angles[0] = 32'h0431FBD4;   // 30 deg
        pos_angles[1] = 32'h052A8A6E;   // 37 deg
        pos_angles[2] = 32'h06487ED5;   // 45 deg
        pos_angles[3] = 32'h0860A91D;   // 60 deg
        pos_angles[4] = 32'h0E3DEC4A;   // 102 deg
        pos_angles[5] = 32'h10C15239;   // 120 deg
        pos_angles[6] = 32'h12D97C80;   // 135 deg
        pos_angles[7] = 32'h14F1A6C7;   // 150 deg
        pos_angles[8] = 32'h1A876CD9;   // 190 deg
        pos_angles[9] = 32'h1D524FE3;   // 210 deg
        pos_angles[10] = 32'h1F6A7A2A;  // 225 deg
        pos_angles[11] = 32'h22E815F5;  // 250 deg
        pos_angles[12] = 32'h29E34D8D;  // 300 deg
        pos_angles[13] = 32'h2BFB77D4;  // 315 deg
        pos_angles[14] = 32'h2F7913A0;  // 340 deg
        // Floating point angles
        pos_angles[15] = 32'h05AB3868;   // 40.6 deg
        pos_angles[16] = 32'h0E01285A;   // 100.3 deg
        pos_angles[17] = 32'h1EC60DA0;   // 220.4 deg
```

```
        pos_angles[18] = 32'h30F78A87;   // 350.7 deg

        for (i = 0; i < 19; i = i + 1) begin
            run_test_case(pos_angles[i], 9+i);
        end

        // ----------------------------------------------Random Negative Angles-----------
-------------------------------------------
        $display("\n------------------------------------ Random Negative Angles ----------
----------------------------------------\n");
        $display("Time Angle(Deg) Angle(Rad)  Angle(Q5.27)
Sine(Real)  Cos(Real)   Sine(Q1.15) Cos(Q1.15) Sine_Err(%%) Cos_Err(%%)");
        $display("---- ---------- ----------  ------------ ----------  ----------  -------
---- ----------- ---------- ----------\n");

        for (i = 0; i < 19; i = i + 1) begin
            run_test_case(-pos_angles[i], 28+i);
        end

        $display("\n================================= ERROR STATISTICS
==========================================\n");
        $display("Maximum Sine Error:     %0.2f%%", max_sine_error);
        $display("Maximum Cosine Error:   %0.2f%%", max_cosine_error);
        $display("====================================== End of Testing
=================================================\n");
        $stop;
    end

    // generate real values of angle in rad and degrees
    always @(angle_in) begin
        angle_rad = $itor($signed(angle_in)) / (1.0 * (1<<27));
        angle_deg = angle_rad * (180.0 / 3.141592653589793);
    end

    // generate real values of sine and cosine
    always @(sine or cosine) begin
        sine_real = $itor($signed(sine)) / 32768.0;
        cosine_real = $itor($signed(cosine)) / 32768.0;
    end

endmodule
```

- ***MATLAB Reference Model:***

```
function CORDIC_model()
    % CORDIC MATLAB Reference Model

    fprintf('=== CORDIC MATLAB Reference Model ===\n');

    % 1) Corner cases (degrees)
    corner_cases = [0, 90, 180, 270, 360];

    % 2) Overflow and Underflow (degrees)
    overflow_cases = [450, -120];

    % 3) Boundary conditions (degrees)
    boundary_cases = [720, -360];

    % 4) Random tests positive (degrees)
    random_positive =
[30,37,45,60,102,120,135,150,190,210,225,250,300,315,340,40.6,
100.3,220.4,350.7];

    % 5) Random tests negative (degrees)
    random_negative = [-30,-37,-45,-60,-102,-120,-135,-150,-
190,-210,-225,-250,-300,-315,-340,-40.6,-100.3,-220.4,-350.7];

    % Combine all
    all_angles = [corner_cases, overflow_cases,
boundary_cases, random_positive, random_negative];

    % Create file to put results in it
    filename = 'MATLAB_outputs.txt';
    fileID = fopen(filename, 'w');
    if fileID == -1
        if ispc
            docs_folder = fullfile(getenv('USERPROFILE'),
'Documents');
        else
            docs_folder = fullfile(getenv('HOME'),
'Documents');
        end
        filename = fullfile(docs_folder,
'MATLAB_outputs.txt');
        fileID = fopen(filename, 'w');
    end

    if fileID == -1
```

```
        fprintf('File can not be created\n');
    else
        % File opened successfully - proceed with normal 
writing
        fprintf('File created successfully: %s\n', filename);
        
        % Display detailed results on screen AND write to file
        fprintf('\n1) CORNER CASES (0,90,180,270,360)\n');
        fprintf('Angle_deg Sine_Hex Cosine_Hex Sine_Real 
Cosine_Real\n');
        for i = 1:length(corner_cases)
            angle_deg = corner_cases(i);
            [sine, cosine, sine_real, cosine_real, sine_hex, 
cosine_hex] = process_angle(angle_deg);
            fprintf('%9.1f %8s %10s %9.6f %10.6f\n', 
angle_deg, sine_hex, cosine_hex, sine_real, cosine_real);
            fprintf(fileID, '%.6f %.6f\n', sine_real, 
cosine_real);
        end
        
        fprintf('\n2) OVERFLOW AND UNDERFLOW (450,-120)\n');
        fprintf('Angle_deg Sine_Hex Cosine_Hex Sine_Real 
Cosine_Real\n');
        for i = 1:length(overflow_cases)
            angle_deg = overflow_cases(i);
            [sine, cosine, sine_real, cosine_real, sine_hex, 
cosine_hex] = process_angle(angle_deg);
            fprintf('%9.1f %8s %10s %9.6f %10.6f\n', 
angle_deg, sine_hex, cosine_hex, sine_real, cosine_real);
            fprintf(fileID, '%.6f %.6f\n', sine_real, 
cosine_real);
        end
        
        fprintf('\n3) BOUNDARY CONDITIONS (720,-360)\n');
        fprintf('Angle_deg Sine_Hex Cosine_Hex Sine_Real 
Cosine_Real\n');
        for i = 1:length(boundary_cases)
            angle_deg = boundary_cases(i);
            [sine, cosine, sine_real, cosine_real, sine_hex, 
cosine_hex] = process_angle(angle_deg);
            fprintf('%9.1f %8s %10s %9.6f %10.6f\n', 
angle_deg, sine_hex, cosine_hex, sine_real, cosine_real);
            fprintf(fileID, '%.6f %.6f\n', sine_real, 
cosine_real);
        end
```

```
        fprintf('\n4) RANDOM TESTS POSITIVE\n');
        fprintf('Angle_deg Sine_Hex Cosine_Hex Sine_Real
Cosine_Real\n');
        for i = 1:length(random_positive)
            angle_deg = random_positive(i);
            [sine, cosine, sine_real, cosine_real, sine_hex,
cosine_hex] = process_angle(angle_deg);
            fprintf('%9.1f %8s %10s %9.6f %10.6f\n',
angle_deg, sine_hex, cosine_hex, sine_real, cosine_real);
            fprintf(fileID, '%.6f %.6f\n', sine_real,
cosine_real);
        end

        fprintf('\n5) RANDOM TESTS NEGATIVE\n');
        fprintf('Angle_deg Sine_Hex Cosine_Hex Sine_Real
Cosine_Real\n');
        for i = 1:length(random_negative)
            angle_deg = random_negative(i);
            [sine, cosine, sine_real, cosine_real, sine_hex,
cosine_hex] = process_angle(angle_deg);
            fprintf('%9.1f %8s %10s %9.6f %10.6f\n',
angle_deg, sine_hex, cosine_hex, sine_real, cosine_real);
            fprintf(fileID, '%.6f %.6f\n', sine_real,
cosine_real);
        end
        fclose(fileID);
    end
end

function [sine, cosine, sine_real, cosine_real, sine_hex,
cosine_hex] = process_angle(angle_deg)

    [sine, cosine] = cordic_algorithm(angle_deg);

    % Convert to real values
    sine_real = double(sine) / 32768;
    cosine_real = double(cosine) / 32768;

    % Convert to hexa
    sine_hex = dec2hex(typecast(sine, 'uint16'), 4);
    cosine_hex = dec2hex(typecast(cosine, 'uint16'), 4);
end

function [sine, cosine] = cordic_algorithm(angle_deg)
```

```
    % Fixed-point formats
    Q5_27_SCALE = 2^27;  % 32-bit angle: 5 integer + 27
fractional

    % Some Q5.27 constants
    PI = int32(hex2dec('1921FB54'));      % pi in Q5.27
    PI_2 = int32(hex2dec('0C90FDAA'));    % pi/2 in Q5.27
    TWO_PI = int32(hex2dec('3243F6A9')); % 2*pi in Q5.27
    Kn = int16(hex2dec('4DBA'));          % Scaling factor in
Q1.15

    % Convert angle to Q5.27
    angle_rad = deg2rad(angle_deg);
    angle_in = int32(angle_rad * Q5_27_SCALE);

    % Quadrant mapping
    if angle_in < 0
        angle_norm = angle_in + TWO_PI;
    elseif angle_in >= TWO_PI
        angle_norm = angle_in - TWO_PI;
    else
        angle_norm = angle_in;
    end

    % Determine quadrant and adjust angle
    if angle_norm <= PI_2           % 1st quadrant
        angle_cordic = angle_norm;
        sin_sign = 0;   % sin positive
        cos_sign = 0;   % cos positive
    elseif angle_norm <= PI         % 2nd quadrant
        angle_cordic = PI - angle_norm;
        sin_sign = 0;   % sin positive
        cos_sign = 1;   % cos negative
    elseif angle_norm <= PI+PI_2    % 3rd quadrant
        angle_cordic = angle_norm - PI;
        sin_sign = 1;   % sin negative
        cos_sign = 1;   % cos negative
    else                            % 4th quadrant
        angle_cordic = TWO_PI - angle_norm;
        sin_sign = 1;   % sin negative
        cos_sign = 0;   % cos positive
    end

    % atan table (Q5.27)
    atan_table = int32([
```

```
        hex2dec('06487ED5'), hex2dec('03B58CE1'),
hex2dec('01F5B760'), ...
        hex2dec('00FEADD5'), hex2dec('007FD56F'),
hex2dec('003FFAAB'), ...
        hex2dec('001FFF55'), hex2dec('000FFFEB'),
hex2dec('0007FFFD'), ...
        hex2dec('00040000'), hex2dec('00020000'),
hex2dec('00010000'), ...
        hex2dec('00008000'), hex2dec('00004000'),
hex2dec('00002000'), ...
        hex2dec('00001000'), hex2dec('00000800'),
hex2dec('00000400'), ...
        hex2dec('00000200'), hex2dec('00000100'),
hex2dec('00000080'), ...
        hex2dec('00000040'), hex2dec('00000020'),
hex2dec('00000010'), ...
        hex2dec('00000008'), hex2dec('00000004'),
hex2dec('00000002'), ...
        hex2dec('00000001'), hex2dec('00000000'),
hex2dec('00000000'), ...
        hex2dec('00000000'), hex2dec('00000000')
    ]);

    x_start = Kn;  % Q1.15
    y_start = int16(0);

    % Initialize with extended precision (convert Q1.15 to
Q1.30 equivalent)
    % Sign extend and shift left by 15 bits
    x = int32(x_start) * int32(32768);  % Convert Q1.15 to
Q1.30
    y = int32(y_start) * int32(32768);  % Convert Q1.15 to
Q1.30
    z = angle_cordic;

    % CORDIC iterations (32)
    for i = 1:32
        if z < 0
            x_next = x + bitsra(y, i-1);
            y_next = y - bitsra(x, i-1);
            z_next = z + atan_table(i);
        else
            x_next = x - bitsra(y, i-1);
            y_next = y + bitsra(x, i-1);
            z_next = z - atan_table(i);
```

```
        end
        x = x_next;
        y = y_next;
        z = z_next;
    end

    % Convert back to Q1.15 (take bits [30:15])
    sine_temp = int16(bitsra(y, 15));    % bit shift right
arithmetic by 15 bits
    cosine_temp = int16(bitsra(x, 15));

    % Apply quadrant correction
    if sin_sign
        sine = -sine_temp;
    else
        sine = sine_temp;
    end

    if cos_sign
        cosine = -cosine_temp;
    else
        cosine = cosine_temp;
    end
end
```

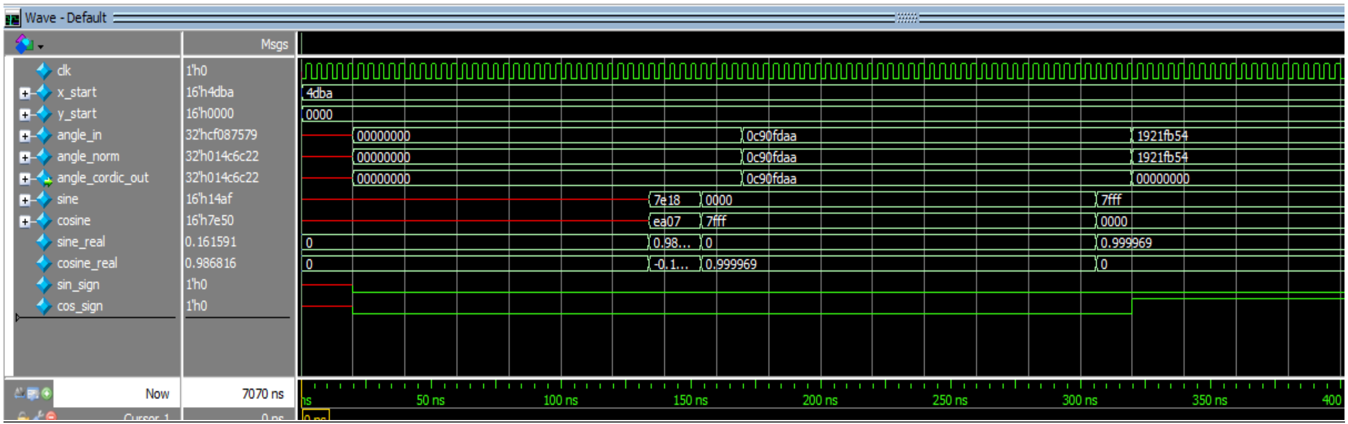
- ***Do file:***

```
vlib work
vlog CORDIC.v quad_logic.v CORDIC_top.v CORDIC_tb.v
vsim -voptargs=+acc work.CORDIC_tb
add wave *
add wave /CORDIC_tb/DUT_top/sin_sign
add wave /CORDIC_tb/DUT_top/cos_sign
add wave /CORDIC_tb/DUT_top/quad_logic_DUT/angle_norm
add wave /CORDIC_tb/DUT_top/quad_logic_DUT/angle_cordic_out
run -all
#quit -sim
```

- ## *Waveform Snippets:*

1) Corner Cases:

⇨ Waveform:



⇨ MATLAB Results:

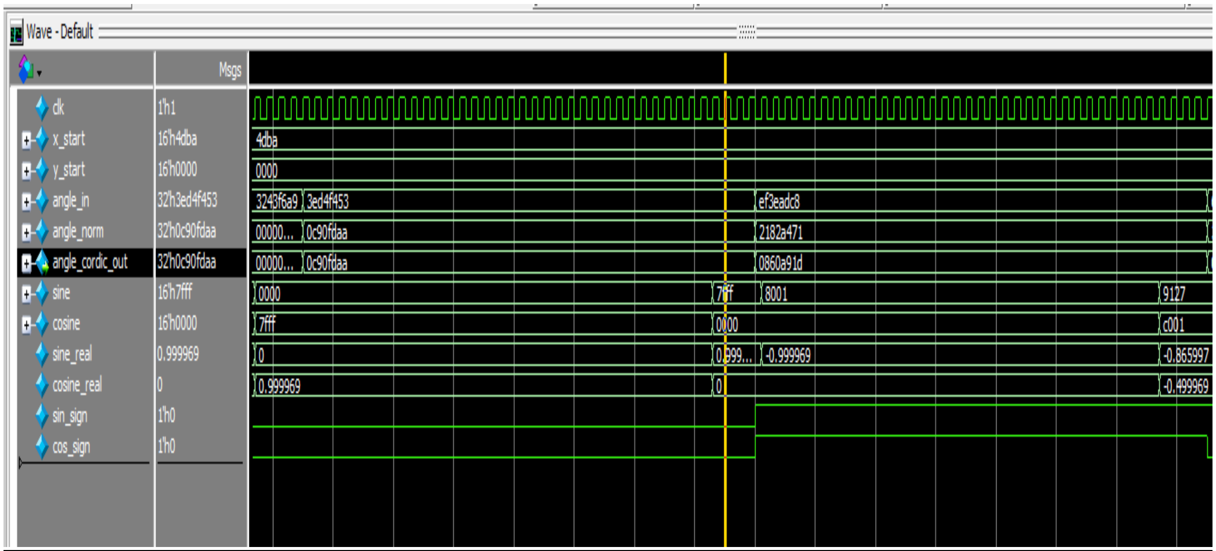
```
>> CORDIC_model
=== CORDIC MATLAB Reference Model ===
File created successfully: MATLAB_outputs.txt

1) CORNER CASES (0,90,180,270,360)
Angle_deg Sine_Hex Cosine_Hex Sine_Real Cosine_Real
      0.0     0000       7FFF  0.000000    0.999969
     90.0     7FFF       0000  0.999969    0.000000
    180.0     0000       8001  0.000000   -0.999969
    270.0     8001       0000 -0.999969    0.000000
    360.0     0000       7FFF  0.000000    0.999969
```

⇨ Transcript and self-checking with MATLAB

```
Reading MATLAB reference results
Read 47 test cases from MATLAB_outputs.txt

================================ CORDIC Algorithm Testbench ==========================================

---------------------------------------- Corner Cases -----------------------------------------

Time Angle(Deg) Angle(Rad)  Angle(Q5.27) Sine(Real)  Cos(Real)   Sine(Q1.15) Cos(Q1.15) Sine_Err(%) Cos_Err(%)
---- ---------- ----------  ------------ ----------  ----------  ----------- ----------- ---------- ----------

 170        0.0   0.000000  000000000000   0.000000    0.999969  00000000000 00000007fff      0.00%      0.00%
 320       90.0   1.570796  00000c90fdaa   0.999969    0.000000  00000007fff 00000000000      0.00%      0.00%
 470      180.0   3.141593  00001921fb54   0.000000   -0.999969  00000000000 00000008001      0.00%      0.00%
 620      270.0   4.712389  000025b2f8ff  -0.999969    0.000000  00000008001 00000000000      0.00%      0.00%
 770      360.0   6.283185  00003243f6a9   0.000000    0.999969  00000000000 00000007fff      0.00%      0.00%
```

2) Overflow and Underflow:

⇨ Waveform:



⇨ MATLAB Results:

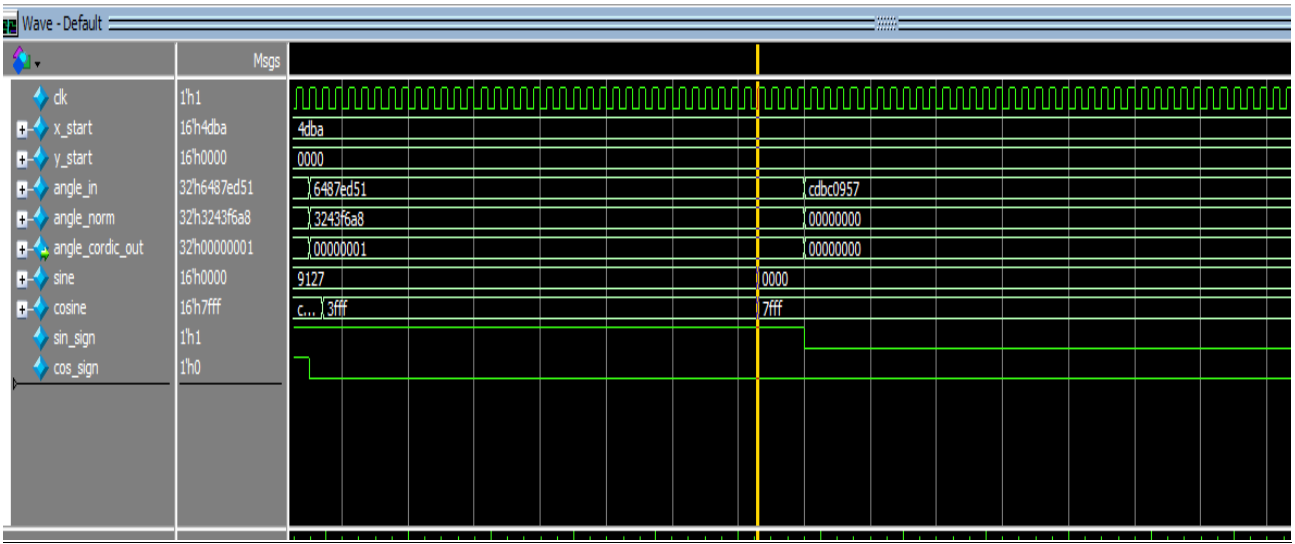
```
2) OVERFLOW AND UNDERFLOW (450,-120)
Angle_deg Sine_Hex Cosine_Hex Sine_Real Cosine_Real
    450.0     7FFF       0000  0.999969   0.000000
   -120.0     9127       C001 -0.865997  -0.499969
```

⇨ Transcript and self-checking with MATLAB

```
# ----------------------------------------- Overflow and Underflow -----------------------------------------
#
# Time Angle(Deg) Angle(Rad)  Angle(Q5.27) Sine(Real)  Cos(Real)   Sine(Q1.15) Cos(Q1.15)  Sine_Err(%) Cos_Err(%)
# ---- ---------- ----------  ------------ ----------  ----------  ----------- ----------- ----------- ----------
#
#  920      450.0   7.853982  00003ed4f453   0.999969    0.000000  00000007fff 00000000000       0.00%      0.00%
# 1070     -120.0  -2.094395  0000ef3eadc8  -0.865997   -0.499969  00000009127 0000000c001       0.00%      0.00%
```

3) Boundary Conditions:

⇨ Waveform:



⇨ MATLAB Results:

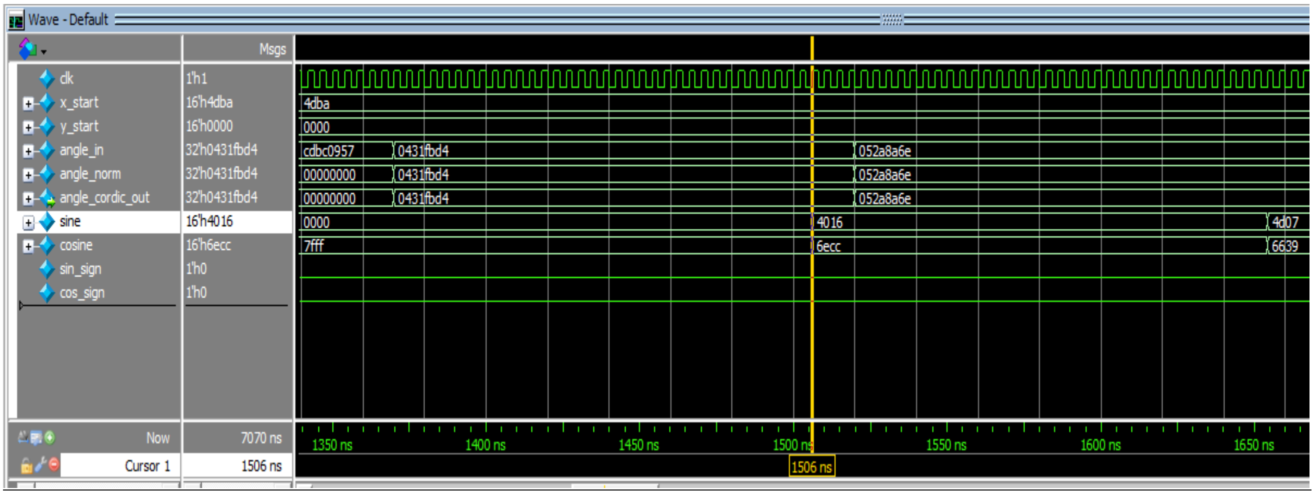
```
3) BOUNDARY CONDITIONS (720,-360)
Angle_deg Sine_Hex Cosine_Hex Sine_Real Cosine_Real
     720.0       0000        7FFF   0.000000    0.999969
    -360.0       0000        7FFF   0.000000    0.999969
```

⇨ Transcript and self-checking with MATLAB

```
---------------------------------------- Boundary conditions -----------------------------------------

Time Angle(Deg) Angle(Rad)  Angle(Q5.27) Sine(Real)  Cos(Real)   Sine(Q1.15) Cos(Q1.15)  Sine_Err(%) Cos_Err(%)
---- ---------- ----------  ------------ ----------  ----------  ----------- ----------- ----------- ----------

1220      720.0  12.566371  00006487ed51   0.000000    0.999969  00000000000 00000007fff       0.00%       0.00%
1370     -360.0  -6.283185  0000cdbc0957   0.000000    0.999969  00000000000 00000007fff       0.00%       0.00%
```

4) Random Tests (Positive):

⇨ Waveform:



⇨ MATLAB Results:

4) RANDOM TESTS POSITIVE

Angle_deg	Sine_Hex	Cosine_Hex	Sine_Real	Cosine_Real
30.0	3FFF	6ED9	0.499969	0.865997
37.0	4D07	6639	0.601776	0.798615
45.0	5A81	5A81	0.707062	0.707062
60.0	6ED9	3FFF	0.865997	0.499969
102.0	7D33	E564	0.978119	-0.207886
120.0	6ED9	C001	0.865997	-0.499969
135.0	5A81	A57F	0.707062	-0.707062
150.0	3FFF	9127	0.499969	-0.865997
190.0	E9C7	81F3	-0.173615	-0.984772
210.0	C001	9127	-0.499969	-0.865997
225.0	A57F	A57F	-0.707062	-0.707062
250.0	87B9	D439	-0.939667	-0.342010
300.0	9127	3FFF	-0.865997	0.499969
315.0	A57F	5A81	-0.707062	0.707062
340.0	D439	7847	-0.342010	0.939667
40.6	534C	612F	0.650757	0.759247
100.3	7DEF	E91E	0.983856	-0.178772
220.4	AD0B	9E87	-0.648102	-0.761505
350.7	EB51	7E50	-0.161591	0.986816

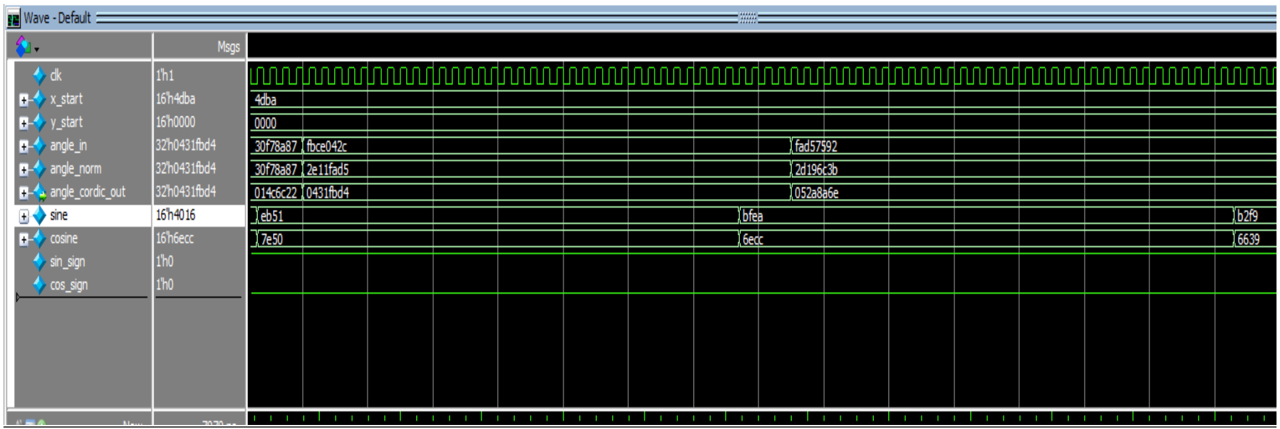
⇨ Transcript and self-checking with MATLAB:

------------------------------------ Random Positive Angles --

Time	Angle(Deg)	Angle(Rad)	Angle(Q5.27)	Sine(Real)	Cos(Real)	Sine(Q1.15)	Cos(Q1.15)	Sine_Err(%)	Cos_Err(%)
1520	30.0	0.524406	00000431fbd4	0.500671	0.865601	00000004016	00000006ecc	0.14%	0.05%
1670	37.0	0.645772	0000052a8a6e	0.601776	0.798615	00000004d07	00000006639	0.00%	0.00%
1820	45.0	0.785398	000006487ed5	0.707062	0.707062	00000005a81	00000005a81	0.00%	0.00%
1970	60.0	1.047198	00000860a91d	0.865997	0.499969	00000006ed9	00000003fff	0.00%	0.00%
2120	102.0	1.780236	00000e3dec4a	0.978119	-0.207886	00000007d33	0000000e564	0.00%	0.00%
2270	120.0	2.094395	000010c15239	0.865997	-0.499969	00000006ed9	0000000c001	0.00%	0.00%
2420	135.0	2.356194	000012d97c80	0.707062	-0.707062	00000005a81	0000000a57f	0.00%	0.00%
2570	150.0	2.617994	000014f1a6c7	0.499969	-0.865997	00000003fff	00000009127	0.00%	0.00%
2720	190.0	3.316126	00001a876cd9	-0.173615	-0.984772	0000000e9c7	000000081f3	0.00%	0.00%
2870	210.0	3.665191	00001d524fe3	-0.499969	-0.865997	0000000c001	00000009127	0.00%	0.00%
3020	225.0	3.926991	00001f6a7a2a	-0.707062	-0.707062	0000000a57f	0000000a57f	0.00%	0.00%
3170	250.0	4.363323	000022e815f5	-0.939667	-0.342010	000000087b9	0000000d439	0.00%	0.00%
3320	300.0	5.235988	000029e34d8d	-0.865997	0.499969	00000009127	00000003fff	0.00%	0.00%
3470	315.0	5.497787	00002bfb77d4	-0.707062	0.707062	0000000a57f	00000005a81	0.00%	0.00%
3620	340.0	5.934119	00002f7913a0	-0.342010	0.939667	0000000d439	00000007847	0.00%	0.00%
3770	40.6	0.708604	000005ab3868	0.650757	0.759247	0000000534c	0000000612f	0.00%	0.00%
3920	100.3	1.750565	00000e01285a	0.983856	-0.178772	00000007def	0000000e91e	0.00%	0.00%
4070	220.4	3.846706	00001ec60da0	-0.648102	-0.761505	0000000ad0b	00000009e87	0.00%	0.00%
4220	350.7	6.120870	000030f78a87	-0.161591	0.986816	0000000eb51	00000007e50	0.00%	0.00%

5) Random Tests (Negative):

➩ Waveform:



➩ MATLAB Results:

```
5) RANDOM TESTS NEGATIVE
Angle_deg Sine_Hex Cosine_Hex Sine_Real  Cosine_Real
    -30.0     C001       6ED9 -0.499969    0.865997
    -37.0     B2F9       6639 -0.601776    0.798615
    -45.0     A57F       5A81 -0.707062    0.707062
    -60.0     9127       3FFF -0.865997    0.499969
   -102.0     82CD       E564 -0.978119   -0.207886
   -120.0     9127       C001 -0.865997   -0.499969
   -135.0     A57F       A57F -0.707062   -0.707062
   -150.0     C001       9127 -0.499969   -0.865997
   -190.0     1639       81F3  0.173615   -0.984772
   -210.0     3FFF       9127  0.499969   -0.865997
   -225.0     5A81       A57F  0.707062   -0.707062
   -250.0     7847       D439  0.939667   -0.342010
   -300.0     6ED9       3FFF  0.865997    0.499969
   -315.0     5A81       5A81  0.707062    0.707062
   -340.0     2BC7       7847  0.342010    0.939667
    -40.6     ACB4       612F -0.650757    0.759247
   -100.3     8211       E91E -0.983856   -0.178772
   -220.4     52F5       9E87  0.648102   -0.761505
   -350.7     14AF       7E50  0.161591    0.986816
```

➩ Transcript and self-checking with MATLAB:

```
----------------------------------- Random Negative Angles -------------------------------------------------

Time Angle(Deg) Angle(Rad)  Angle(Q5.27) Sine(Real)  Cos(Real)   Sine(Q1.15) Cos(Q1.15) Sine_Err(%) Cos_Err(%)
---- ---------- ----------  ------------ ----------  ----------  ----------- ----------- ---------- ----------

4370      -30.0  -0.524406  0000fbce042c  -0.500671    0.865601  0000000bfea 00000006ecc       0.14%       0.05%
4520      -37.0  -0.645772  0000fad57592  -0.601776    0.798615  0000000b2f9 00000006639       0.00%       0.00%
4670      -45.0  -0.785398  0000f9b7812b  -0.707062    0.707062  0000000a57f 00000005a81       0.00%       0.00%
4820      -60.0  -1.047198  0000f79f56e3  -0.865997    0.499969  00000009127 00000003fff       0.00%       0.00%
4970     -102.0  -1.780236  0000f1c213b6  -0.978119   -0.207886  000000082cd 0000000e564       0.00%       0.00%
5120     -120.0  -2.094395  0000ef3eadc7  -0.865997   -0.499969  00000009127 0000000c001       0.00%       0.00%
5270     -135.0  -2.356194  0000ed268380  -0.707062   -0.707062  0000000a57f 0000000a57f       0.00%       0.00%
5420     -150.0  -2.617994  0000eb0e5939  -0.499969   -0.865997  0000000c001 00000009127       0.00%       0.00%
5570     -190.0  -3.316126  0000e5789327   0.173615   -0.984772  00000001639 000000081f3       0.00%       0.00%
5720     -210.0  -3.665191  0000e2adb01d   0.499969   -0.865997  00000003fff 00000009127       0.00%       0.00%
5870     -225.0  -3.926991  0000e09585d6   0.707062   -0.707062  00000005a81 0000000a57f       0.00%       0.00%
6020     -250.0  -4.363323  0000dd17ea0b   0.939667   -0.342010  00000007847 0000000d439       0.00%       0.00%
6170     -300.0  -5.235988  0000d61cb273   0.865997    0.499969  00000006ed9 00000003fff       0.00%       0.00%
6320     -315.0  -5.497787  0000d404882c   0.707062    0.707062  00000005a81 00000005a81       0.00%       0.00%
6470     -340.0  -5.934119  0000d086ec60   0.342010    0.939667  00000002bc7 00000007847       0.00%       0.00%
6620      -40.6  -0.708604  0000fa54c798  -0.650757    0.759247  0000000acb4 0000000612f       0.00%       0.00%
6770     -100.3  -1.750565  0000f1fed7a6  -0.983856   -0.178772  00000008211 0000000e91e       0.00%       0.00%
6920     -220.4  -3.846706  0000e139f260   0.648102   -0.761505  000000052f5 00000009e87       0.00%       0.00%
7070     -350.7  -6.120870  0000cf087579   0.161591    0.986816  000000014af 00000007e50       0.00%       0.00%
```

• *Overall Verification Results:*

- The verification strategy successfully validated the CORDIC algorithm implementation against all specified requirements. The comprehensive test coverage ensured functional correctness, numerical accuracy, and timing compliance. The MATLAB comparison provided quantitative accuracy metrics, confirming the implementation meets precision requirements for trigonometric function generation.

- **Verification Completeness**: 100% of specified requirements covered and validated through automated testing and reference model comparison.

- **We have worked on fixed point arithmetic format (Qmn) of** :
  1) Q1.15 (16-bit fixed width) for input and output (sin and cos) signals.
  2) Q5.27 (32-bit for angles) to represent angles [-2π to 4π] range.

- **Coverage Metrics**:

    1) **Angle Range Coverage**: 100% of specified [-2π to 4π] range.
    2) **Corner cases**: (0,90,180,270,360)
    3) **Quadrant Coverage**: All four trigonometric quadrants tested.
    4) **Sign Coverage**: Positive and negative angles verified. (0° to 720° and -360 to 0)
    5) **Boundary coverage**: Minimum and Maximum angles (-360 and 720)
    6) **Special Value Coverage**: All major trigonometric angles including floating-point angles (40.6°, 100.3°, 220.4°, 350.7°).

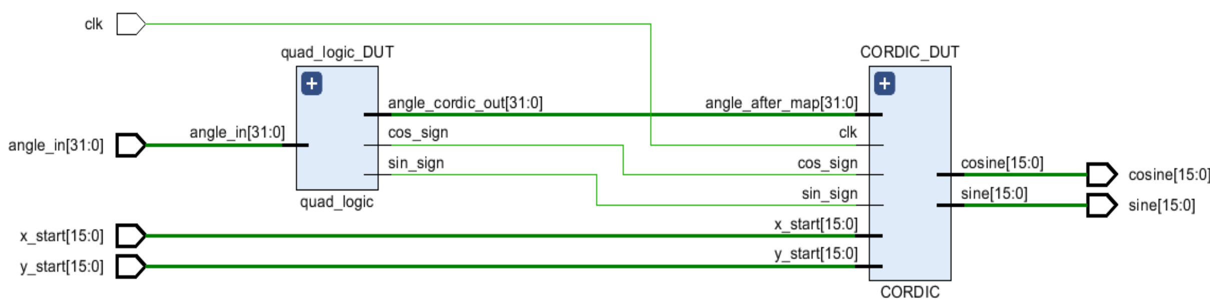
- **Matching between Design and MATLAB reference model**: most of angles have zero percentage error between Design and MATLAB results and maximum error achieved is about 0.1% which is a good tolerance.

```
================================ ERROR STATISTICS ========================================

Maximum Sine Error:     0.14%
Maximum Cosine Error:   0.05%
===================================== End of Testing ===============================================
```

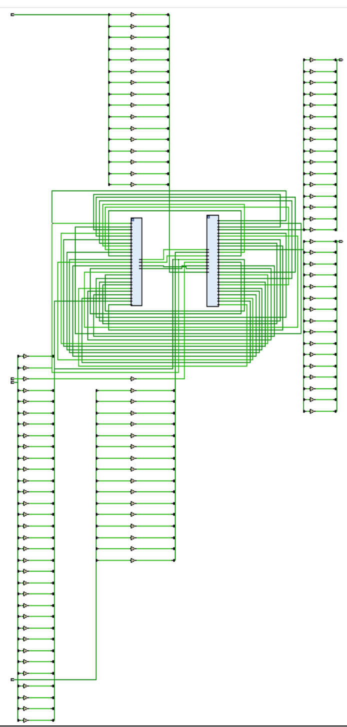
• *FPGA Implementation of CORDIC Algorithm:*

Our goal is to implement the CORDIC module for sine and cosine calculation on a **Zybo Z7 20/A.0** board.

1) Elaborated Design:



2) Synthesis:



⇨ Timing Summary after Synthesis on 100 MHZ:

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.136 ns	Worst Hold Slack (WHS):	0.079 ns	Worst Pulse Width Slack (WPWS):	4.020 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	2765	Total Number of Endpoints:	2765	Total Number of Endpoints:	2830

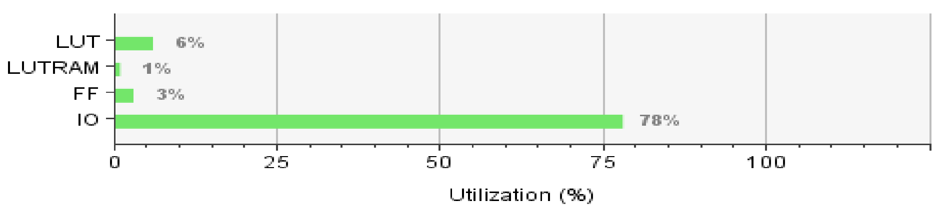
All user specified timing constraints are met.

```
3. Global Clock Source Details
------------------------------

+-----------+-----------+-----------------+------------+------------+------------+-------------+-----------------+---------------------+--------------+-----------------+----------+
| Source Id | Global Id | Driver Type/Pin | Constraint | Site       | Clock Region | Clock Loads | Non-Clock Loads | Source Clock Period | Source Clock | Driver Pin      | Net      |
+-----------+-----------+-----------------+------------+------------+------------+-------------+-----------------+---------------------+--------------+-----------------+----------+
| src0      | g0        | IBUF/O          | IOB_X1Y126 | IOB_X1Y126 | X1Y2         |           1 |               0 |              10.000 | sys_clk_pin  | clk_IBUF_inst/O | clk_IBUF |
+-----------+-----------+-----------------+------------+------------+------------+-------------+-----------------+---------------------+--------------+-----------------+----------+
* Clock Loads column represents the clock pin loads (pin count)
** Non-Clock Loads column represents the non-clock pin loads (pin count)
```

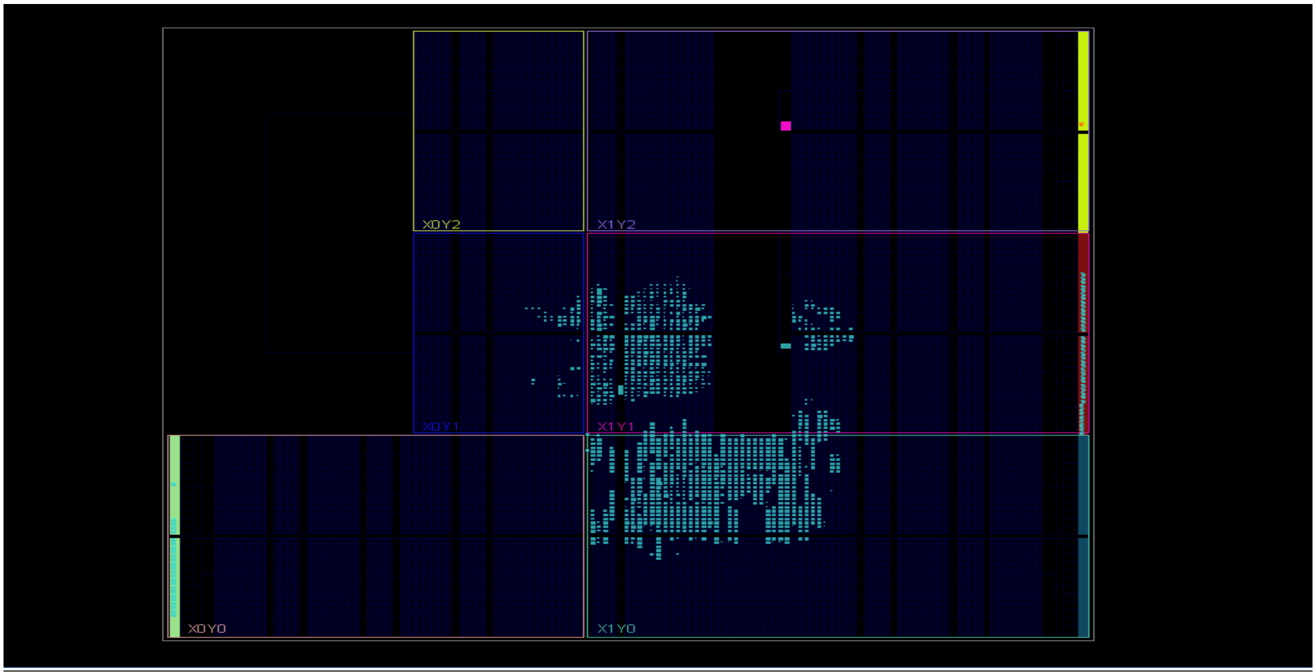
⇨ Utilization Report after Synthesis:

Summary

Resource	Utilization	Available	Utilization %
LUT	3050	53200	5.73
LUTRAM	15	17400	0.09
FF	2814	106400	2.64
IO	97	125	77.60



3) Implementation:



⇨ Timing Summary after Implementation on 100 MHZ:

Design Timing Summary

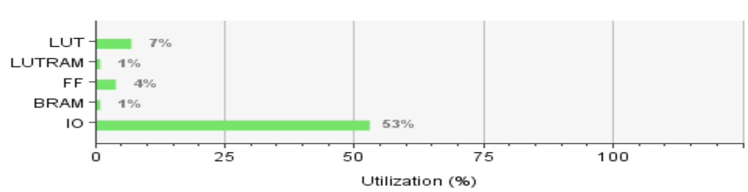
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2.578 ns	Worst Hold Slack (WHS):	0.044 ns	Worst Pulse Width Slack (WPWS):	3.750 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	6165	Total Number of Endpoints:	6149	Total Number of Endpoints:	4659

All user specified timing constraints are met.

⇨ Utilization Report after Implementation:

Summary

Resource	Utilization	Available	Utilization %
LUT	3913	53200	7.36
LUTRAM	100	17400	0.57
FF	4499	106400	4.23
BRAM	0.50	140	0.36
IO	66	125	52.80



- ***Conclusion:***

  The CORDIC algorithm was successfully implemented in Verilog with a quadrant-mapping stage and a fully parameterized iterative core. Through extensive testing against a MATLAB fixed-point reference model, the design was verified to compute sine and cosine accurately across a wide input range, with only minor quantization errors inherent to fixed-point arithmetic. The project demonstrated how CORDIC provides an efficient, hardware-friendly alternative to multipliers for trigonometric functions, while also highlighting the trade-offs between accuracy, iteration count, and hardware cost.