CORDIC Algorithm

By: Bassant Hany Ammar

Objective:

The main objective of this project is to design and implement a sine and cosine calculation unit using the CORDIC algorithm. The unit efficiently computes trigonometric functions with fixed-point arithmetic, making it suitable for FPGA or ASIC implementations where floating-point operations are resource-intensive.

Algorithm:

The CORDIC algorithm is an iterative method used to calculate trigonometric functions, hyperbolic functions, magnitude, and phase.

It works by performing a sequence of shift and add operations instead of using multipliers, making it very efficient for hardware implementation.

In this project, the algorithm is applied to compute sine and cosine values for a given input angle using fixed-point arithmetic.

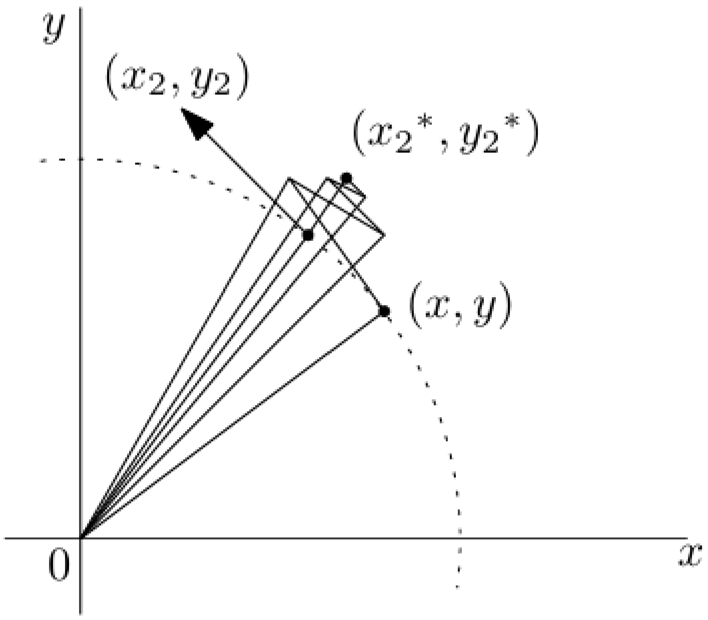
The equations of it is:

$x_i = x_{i-1} - \alpha_i y_{i-1} 2^{i-1}$

$y_i = y_{i-1} + \alpha_i x_{i-1} 2^{i-1}$

$z_i = z_{i-1} - \alpha_i \tan^{-1} 2^{i-1}$

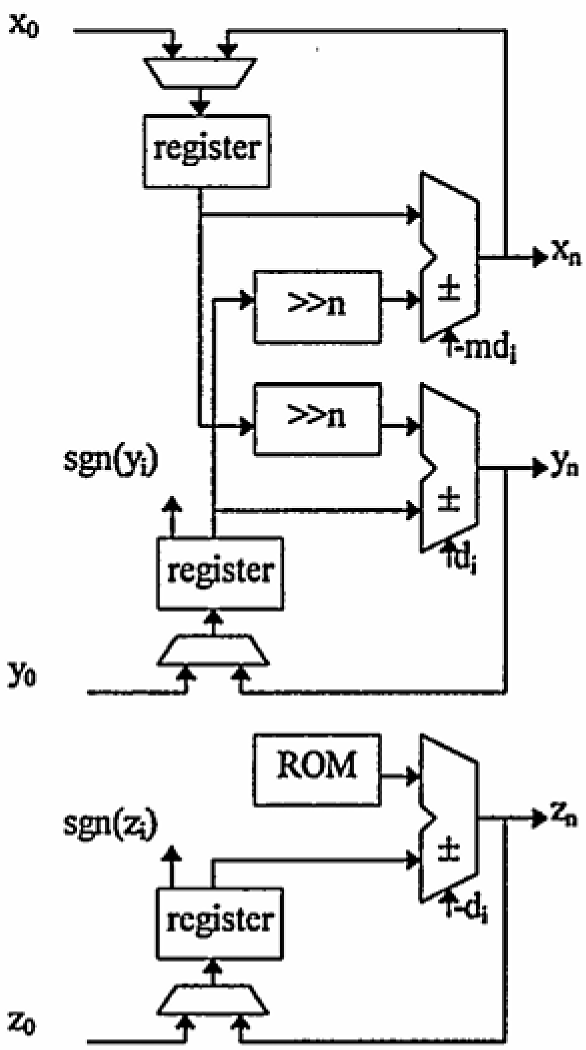
$$\sigma_i = \begin{cases} 1, & \text{if } z_i \geq 0 \\ -1, & \text{otherwise} \end{cases}$$



Project Implementation:

- In this project, the CORDIC algorithm was implemented using Verilog HDL to compute sine and cosine values.
- The design uses fixed-point representation (Q5.27) for efficient hardware realization.
- The inputs (angle values) are represented on 32 bits, and the outputs are generated as fixed-point values.
- A testbench was developed as self-checking to verify the design by applying different input angles and comparing the results with MATLAB golden models.
- Finally, the design was synthesized and prepared for FPGA implementation using Vivado.

Block Diagram:



MATLAB Script:

```matlab
% Test Cases for CORDIC Sine/Cosine in Q5.27
corner_cases   = [0, pi/2, pi, 3*pi/2, 2*pi];
overflow_cases = [3*pi, -pi/6];
random_cases   = [pi/6, pi/4, 3*pi/4, pi/3, pi/8];
boundary_cases = [4*pi, -2*pi];

% Run & Save Results
run_and_save(corner_cases,   'corner_cases.txt');
run_and_save(overflow_cases, 'over_cases.txt');
run_and_save(random_cases,   'random_cases.txt');
run_and_save(boundary_cases, 'boundary_cases.txt');

disp('✅ All results saved to text files.');

 %% functions
function run_and_save(test_angles, filename)
    % Create file
    fid = fopen(filename,'w');
    fprintf(fid, '   cos(Q5.27)   cos(real)   sin(Q5.27)   sin(real)\n');

    % Loop over angles
    for k = 1:length(test_angles)
        theta = test_angles(k);

        % Call golden model
        [c_val, s_val] = golden_model(theta);

        % Q5.27 integer
        cos_q = round(c_val * 2^27);
        sin_q = round(s_val * 2^27);

        % Convert Q5.27 back to real (like Verilog q5_27_to_real)
        cos_real = double(cos_q) / 2^27;
        sin_real = double(sin_q) / 2^27;

        % Write to file
        fprintf(fid, '%d   %.9f   %d   %.9f\n', ...
        cos_q, cos_real, sin_q, sin_real);
    end

     fclose(fid);
     fprintf('Results written to %s\n', filename);
end
```

```matlab
function [cos_val, sin_val] = golden_model(theta)
    % =========================
    % Fixed-point parameters
    % =========================
    N = 32;                    % iterations
    data_width = 32;           % bits for x,y
    data_frac  = 27;           % frac bits for x,y (Q5.27)
    angle_bits = 32;           % total bits for angle
    angle_frac = 27;           % frac bits for angle

    F = fimath('RoundingMethod','Nearest', ...
               'OverflowAction','Saturate', ...
               'ProductMode','FullPrecision', ...
               'SumMode','FullPrecision');

    Tdata  = numerictype(1, data_width, data_frac);
    Tangle = numerictype(1, angle_bits, angle_frac);

    % =========================
    % Q5.27 constants
    % =========================
    PI         = int32(hex2dec('1921FB54')); % pi
    TWO_PI     = int32(hex2dec('3243F6A8')); % 2*pi
    PI_2       = int32(hex2dec('0C90FDAA')); % pi/2
    THREE_PI_2 = int32(hex2dec('25B2F8FE')); % 3*pi/2

    % =========================
    % Angle Mapping (like Verilog)
    % =========================
    theta_q = int32(round(theta * 2^27)); % rad → Q5.27

    if theta_q < 0
        angle_norm = theta_q + TWO_PI;
    elseif theta_q >= TWO_PI
        angle_norm = theta_q - TWO_PI;
    else
        angle_norm = theta_q;
    end

    if angle_norm <= PI_2
        angle_cordic = angle_norm;
        sine_sign = 0; cosine_sign = 0;
    elseif angle_norm <= PI
        angle_cordic = PI - angle_norm;
        sine_sign = 0; cosine_sign = 1;
    elseif angle_norm <= THREE_PI_2
        angle_cordic = angle_norm - PI;
        sine_sign = 1; cosine_sign = 1;
    else
        angle_cordic = TWO_PI - angle_norm;
        sine_sign = 1; cosine_sign = 0;
    end
```

```matlab
    % =========================
    % Prepare for CORDIC
    % =========================
    theta_wrapped = double(angle_cordic) / 2^27; % back to real
    z = fi(theta_wrapped, Tangle, F);

    hex_vals_atan = {'06487ED5','03B58CE1','01F5B760','00FEADD5', ...
        '007FD56F','003FFAAB','001FFF55','000FFFEB','0007FFFD', ...
        '00040000','00020000','00010000','00008000','00004000',...
        '00002000','00001000','00000800','00000400','00000200', ...
        '00000100','00000080','00000040','00000020','00000010', ...
        '00000008','00000004','00000002','00000001','00000000', ...
        '00000000','00000000','00000000'};
    atan_table = hex2dec(hex_vals_atan) / 2^27;

    % CORDIC gain
    K = prod(1 ./ sqrt(1 + 2.^(-2*(0:N-1))));
    x = fi(K, Tdata, F);
    y = fi(0, Tdata, F);

    % =========================
    % Iterations
    % =========================
    for i = 0:(N-1)
        ai = atan_table(i+1);
        if z >= 0
            x_new = x - bitsra(y, i);
            y_new = y + bitsra(x, i);
            z_new = z - ai;
        else
            x_new = x + bitsra(y, i);
            y_new = y - bitsra(x, i);
            z_new = z + ai;
        end
        x = fi(x_new, Tdata, F);
        y = fi(y_new, Tdata, F);
        z = fi(z_new, Tangle, F);
    end

    % =========================
    % Apply signs (from quadrant mapping)
    % =========================
    cos_val = double(x);
    sin_val = double(y);
    if cosine_sign, cos_val = -cos_val; end
    if sine_sign,   sin_val = -sin_val; end
end
```

RTL Code:

1. PreProcessing Module

```
module preprocessing #(parameter WIDTH = 32)(
    input wire signed [WIDTH-1:0] angle,
    output reg signed [WIDTH-1:0] angle_cordic, // angle input to CORDIC in [0, pi/2]
    output reg sine_sign , // 0 is positive && 1 is negative
    output reg cosine_sign  // 0 is positive && 1 is negative
) ;

// Q5.27 Constants
localparam signed PI = 32'h1921FB54 ;
localparam signed TWO_PI = 32'h3243F6A8;
localparam signed PI_2 = 32'h0C90FDAA;
localparam THREE_PI_2 = 32'h25B2F8FE ;

reg signed [WIDTH-1:0] angle_normalized ; //after normatization to [0, 2pi]
```

```
always @(*) begin
    if (angle < 0 ) begin
        angle_normalized = angle + TWO_PI ;
    end else if (angle >= TWO_PI) begin
        angle_normalized = angle - TWO_PI ;
    end else angle_normalized = angle ;
        // starting normalizing to gent angle_cordic
        if (angle_normalized <= PI_2) begin
            angle_cordic = angle_normalized ;
            sine_sign = 0 ;
            cosine_sign = 0 ;
        end else if (angle_normalized <= PI) begin
            angle_cordic = PI - angle_normalized ;
            sine_sign = 0 ;
            cosine_sign = 1 ;
        end else if (angle_normalized <= THREE_PI_2) begin
            angle_cordic = angle_normalized - PI ;
            sine_sign = 1 ;
            cosine_sign = 1 ;
        end else begin
            angle_cordic = TWO_PI - angle_normalized  ;
            sine_sign = 1 ;
            cosine_sign = 0 ;
        end
end
endmodule
```

2. Cordic_algorithm module

```
module cordic_algorithm #(parameter WIDTH = 32)(
    input wire clk,
    input wire signed [WIDTH-1:0] angle_cordic, // angle input to CORDIC in [0, pi/2]
    input wire signed [WIDTH-1:0] x_start , // initial x value (K factor)
    input wire signed [WIDTH-1:0] y_start , // initial y value (0)
    input wire sine_sign , // 0 is positive && 1 is negative
    input wire cosine_sign , // 0 is positive && 1 is negative
    output reg signed [WIDTH-1:0] sine , // output sine value
    output reg signed [WIDTH-1:0] cosine // output cosine value
) ;
integer i ;
    // atan lookup table (same format Q5.27)
    reg signed [WIDTH-1:0] atan_table [0:WIDTH-1];
    initial begin
        atan_table[00] = 32'h06487ED5;
        atan_table[01] = 32'h03B58CE1;
        atan_table[02] = 32'h01F5B760;
        atan_table[03] = 32'h00FEADD5;
        atan_table[04] = 32'h007FD56F;
        atan_table[05] = 32'h003FFAAB;
        atan_table[06] = 32'h001FFF55;
        atan_table[07] = 32'h000FFFEB;
        atan_table[08] = 32'h0007FFFD;
        atan_table[09] = 32'h00040000;
        atan_table[10] = 32'h00020000;
        atan_table[11] = 32'h00010000;
        atan_table[12] = 32'h00008000;
        atan_table[13] = 32'h00004000;
        atan_table[14] = 32'h00002000;
        atan_table[15] = 32'h00001000;
        atan_table[16] = 32'h00000800;
        atan_table[17] = 32'h00000400;
        atan_table[18] = 32'h00000200;
        atan_table[19] = 32'h00000100;
        atan_table[20] = 32'h00000080;
        atan_table[21] = 32'h00000040;
        atan_table[22] = 32'h00000020;
        atan_table[23] = 32'h00000010;
        atan_table[24] = 32'h00000008;
        atan_table[25] = 32'h00000004;
        atan_table[26] = 32'h00000002;
        atan_table[27] = 32'h00000001;
        atan_table[28] = 32'h00000000;
        atan_table[29] = 32'h00000000;
        atan_table[30] = 32'h00000000;
        atan_table[31] = 32'h00000000;
    end

    reg signed [WIDTH-1:0] x_reg, y_reg, z_reg;
    reg signed [WIDTH-1:0] x_next, y_next, z_next;
    reg signed [WIDTH-1:0] x_prev, y_prev;
```

```
    always @(*) begin
        x_next = x_start ;
        y_next = y_start ;
        z_next = angle_cordic ;

        for (i=0 ; i<WIDTH ; i=i+1) begin
            x_prev = x_next;
            y_prev = y_next;
            if (z_next >= 0) begin
                // compute using previous values (important)
                x_next = x_prev - (y_prev >>> i);
                y_next = y_prev + (x_prev >>> i);
                z_next = z_next - atan_table[i];
            end else begin
                x_next = x_prev + (y_prev >>> i);
                y_next = y_prev - (x_prev >>> i);
                z_next = z_next + atan_table[i];
            end
        end
    end
    always@(posedge clk) begin
        x_reg <= x_next ;
        y_reg <= y_next ;
        z_reg <= z_next ;

    if (sine_sign)begin
        sine <= -y_reg ;
    end else sine <= y_reg ;

        if (cosine_sign) begin
            cosine <= -x_reg ;
        end else cosine <= x_reg ;
    end
endmodule
```

3. Top Module:

```
module cordic_top #(parameter WIDTH=32) (
    input wire clk,
    input wire signed [WIDTH-1:0] x_start,
    input wire signed [WIDTH-1:0] y_start,
    input wire signed [WIDTH-1:0] angle,
    output wire signed [WIDTH-1:0] cosine,
    output wire signed [WIDTH-1:0] sine
);
wire signed [WIDTH-1:0] angle_cordic ;
wire sine_sign, cosine_sign ;

preprocessing #(.WIDTH (WIDTH)) U0 (
    .angle(angle),
    .angle_cordic(angle_cordic), // angle input to CORDIC in [0, pi/2]
    .sine_sign(sine_sign), // 0 is positive && 1 is negative
    .cosine_sign(cosine_sign)
);

cordic_algorithm #(.WIDTH(WIDTH)) U1 (
    .clk(clk),
    .angle_cordic(angle_cordic), // angle input to CORDIC in [0, pi/2]
    .x_start(x_start) , // initial x value (K factor)
    .y_start(y_start) , // initial y value (0)
    .sine_sign(sine_sign) , // 0 is positive && 1 is negative
    .cosine_sign(cosine_sign) , // 0 is positive && 1 is negative
    .sine(sine) , // output sine value
    .cosine(cosine)
);
endmodule
```

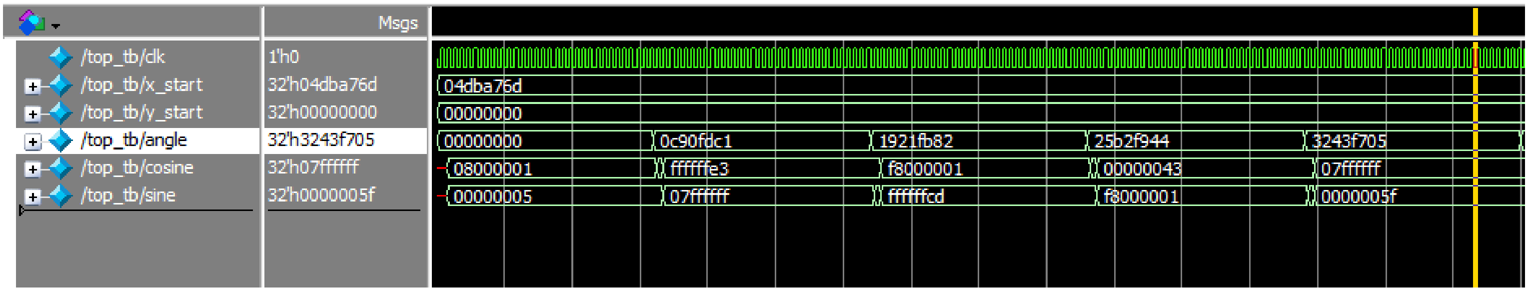
Testbench code:

```
module cordic_tb;
    localparam WIDTH = 32;
    localparam N_CORNER  = 5;
    localparam N_RANDOM  = 5;
    localparam N_OVER    = 2;
    localparam N_BOUND   = 2;

    reg clk;
    reg signed [WIDTH-1:0] x_start;
    reg signed [WIDTH-1:0] y_start;
    reg signed [WIDTH-1:0] angle;
    wire signed [WIDTH-1:0] cosine;
    wire signed [WIDTH-1:0] sine;

    // DUT
    cordic #(.WIDTH(WIDTH)) dut (
        .clk(clk),
        .x_start(x_start),
        .y_start(y_start),
        .angle(angle),
        .cosine(cosine),
        .sine(sine)
    );

    // Clock
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    // Convert functions
    function real q5_27_to_real;
        input [31:0] val;
        begin
            q5_27_to_real = $itor($signed(val)) / (2.0**27);
        end
    endfunction

    function [31:0] real_to_q5_27;
        input real val;
        begin
            real_to_q5_27 = $rtoi(val * (2.0**27));
        end
    endfunction

    // arrays of angles
    real corner_angles [0:N_CORNER-1];
    real random_angles [0:N_RANDOM-1];
    real over_angles   [0:N_OVER-1];
    real bound_angles  [0:N_BOUND-1];

    integer idx;

    // Task for each group
    task run_corner;
        input [1023:0] fname;
        integer fd, r;
        reg [1023:0] line;
        integer cos_q, sin_q;
        real cos_real, sin_real;
        real cos_dut, sin_dut, err_cos, err_sin;
        real err_cos_pos, err_sin_pos; // for positive error

        begin
            fd = $fopen(fname,"r");
            if (fd == 0) begin
                $display("ERROR: Cannot open %s", fname);
                $finish;
            end
```

```verilog
            r = $fgets(line, fd); // skip header
            for (idx=0; idx<N_CORNER; idx=idx+1) begin
                r = $fscanf(fd,"%d %f %d %f\n",cos_q,cos_real,sin_q,sin_real);
                angle = real_to_q5_27(corner_angles[idx]);
                #320;
                cos_dut = q5_27_to_real(cosine);
                sin_dut = q5_27_to_real(sine);

                err_cos = (cos_dut - cos_real) ;
                err_cos_pos = (err_cos<0)? -err_cos: err_cos ;

                err_sin_pos = (sin_dut - sin_real);
                err_sin_pos = (err_sin<0)? -err_sin: err_sin ;
                $display("Corner[%0d] theta=%.6f | DUT cos=%.9f sin=%.9f | MATLAB cos=%.9f sin=%.9f | err=%10.3f %10.3f",
                         idx, corner_angles[idx], cos_dut, sin_dut, cos_real, sin_real, err_cos_pos, err_sin_pos);
            end
            $fclose(fd);
        end
    endtask

    task run_random;
        input [1023:0] fname;
        integer fd, r;
        reg [1023:0] line;
        integer cos_q, sin_q;
        real cos_real, sin_real;
        real cos_dut, sin_dut, err_cos, err_sin;
        real err_cos_pos, err_sin_pos;
        begin
            fd = $fopen(fname,"r");
            if (fd == 0) begin
                $display("ERROR: Cannot open %s", fname);
                $finish;
            end
            r = $fgets(line, fd);
            for (idx=0; idx<N_RANDOM; idx=idx+1) begin
                r = $fscanf(fd,"%d %f %d %f\n",cos_q,cos_real,sin_q,sin_real);
                angle = real_to_q5_27(random_angles[idx]);
                #320;
                cos_dut = q5_27_to_real(cosine);
                sin_dut = q5_27_to_real(sine);

                err_cos = (cos_dut - cos_real) ;
                err_cos_pos = (err_cos<0)? -err_cos: err_cos ;

                err_sin_pos = (sin_dut - sin_real);
                err_sin_pos = (err_sin<0)? -err_sin: err_sin ;
                $display("Random[%0d] theta=%.6f | DUT cos=%.9f sin=%.9f | MATLAB cos=%.9f sin=%.9f | err=%.3f %.3f",
                         idx, random_angles[idx], cos_dut, sin_dut, cos_real, sin_real, err_cos_pos, err_sin_pos);
            end
            $fclose(fd);
        end
    endtask
    task run_over;
        input [1023:0] fname;
        integer fd, r;
        reg [1023:0] line;
        integer cos_q, sin_q;
        real cos_real, sin_real;
        real cos_dut, sin_dut, err_cos, err_sin;
        real err_cos_pos, err_sin_pos;
        begin
            fd = $fopen(fname,"r");
            if (fd == 0) begin
                $display("ERROR: Cannot open %s", fname);
                $finish;
            end
            r = $fgets(line, fd);
            for (idx=0; idx<N_OVER; idx=idx+1) begin
                r = $fscanf(fd,"%d %f %d %f\n",cos_q,cos_real,sin_q,sin_real);
                angle = real_to_q5_27(over_angles[idx]);
                #320;
                cos_dut = q5_27_to_real(cosine);
                sin_dut = q5_27_to_real(sine);

                err_cos = (cos_dut - cos_real) ;
                err_cos_pos = (err_cos<0)? -err_cos: err_cos ;

                err_sin_pos = (sin_dut - sin_real);
                err_sin_pos = (err_sin<0)? -err_sin: err_sin ;

                $display("Over[%0d] theta=%.6f | DUT cos=%.9f sin=%.9f | MATLAB cos=%.9f sin=%.9f | err=%10.2f %10.2f",
                         idx, over_angles[idx], cos_dut, sin_dut, cos_real, sin_real, err_cos_pos, err_sin_pos);
            end
            $fclose(fd);
        end
    endtask
```

```verilog
task run_bound;
    input [1023:0] fname;
    integer fd, r;
    reg [1023:0] line;
    integer cos_q, sin_q;
    real cos_real, sin_real;
    real cos_dut, sin_dut, err_cos, err_sin;
    real err_cos_pos, err_sin_pos;

    begin
        fd = $fopen(fname,"r");
        if (fd == 0) begin
            $display("ERROR: Cannot open %s", fname);
            $finish;
        end
        r = $fgets(line, fd);
        for (idx=0; idx<N_BOUND; idx=idx+1) begin
            r = $fscanf(fd,"%d %f %d %f\n",cos_q,cos_real,sin_q,sin_real);
            angle = real_to_q5_27(bound_angles[idx]);
            #320;
            cos_dut = q5_27_to_real(cosine);
            sin_dut = q5_27_to_real(sine);

            err_cos = (cos_dut - cos_real) ;
            err_cos_pos = (err_cos<0)? -err_cos: err_cos ;

            err_sin_pos = (sin_dut - sin_real);
            err_sin_pos = (err_sin<0)? -err_sin: err_sin ;

            $display("Bound[%0d] theta=%.6f | DUT cos=%.9f sin=%.9f | MATLAB cos=%.9f sin=%.9f | err=%.3f  %.3f",
                     idx, bound_angles[idx], cos_dut, sin_dut, cos_real, sin_real, err_cos_pos, err_sin_pos);
        end
        $fclose(fd);
    end
endtask

    initial begin
        // CORDIC gain init
        x_start = real_to_q5_27(0.607252935);
        y_start = 0;
        angle   = 0;

        // Define angles manually
        corner_angles[0]=0.0;
        corner_angles[1]=3.141593/2.0;
        corner_angles[2]=3.141593;
        corner_angles[3]=3.0*3.141593/2.0;
        corner_angles[4]=2.0*3.141593;

        random_angles[0]=3.141593/6.0;
        random_angles[1]=3.141593/4.0;
        random_angles[2]=3.0*3.141593/4.0;
        random_angles[3]=3.141593/3.0;
        random_angles[4]=3.141593/8.0;

        over_angles[0]=3.0*3.141593;
        over_angles[1]=-3.141593/6.0;

        bound_angles[0]=4.0*3.141593;
        bound_angles[1]=-2.0*3.141593;

        $display(" Starting self-checking CORDIC testbench...");

        $display("=========== Corner Cases ==============");
        run_corner("corner_cases.txt");

        $display("=========== Random Cases ==============");
        run_random("random_cases.txt");

        $display("=========== Overflow Cases ============");
        run_over("over_cases.txt");

        $display("=========== Boundary Cases ============");
        run_bound("boundary_cases.txt");

        $display(" All tests done.");
        $finish;
    end
endmodule
```

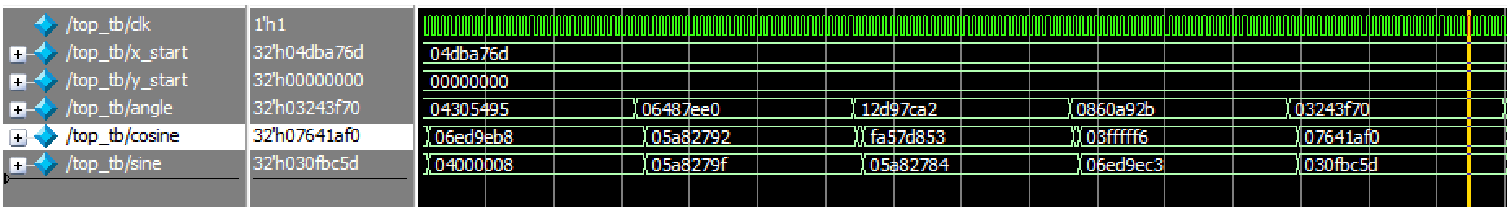
Waveform of corner cases[0, pi/2, pi, 3pi/2, 2pi]:



Transcript of corner cases:

```
# =========== Corner Cases ==============
# Corner[0] theta=0.000000 | DUT cos=1.000000007 sin=0.000000037 | MATLAB cos=1.000000007 sin=0.000000037 | err=    0.000      0.000
# Corner[1] theta=1.570796 | DUT cos=-0.000000216 sin=0.999999993 | MATLAB cos=0.000000022 sin=1.000000007 | err=    0.000      0.000
# Corner[2] theta=3.141593 | DUT cos=-0.999999993 sin=-0.000000380 | MATLAB cos=-1.000000007 sin=0.000000037 | err=    0.000      0.000
# Corner[3] theta=4.712389 | DUT cos=0.000000499 sin=-0.999999993 | MATLAB cos=-0.000000022 sin=-1.000000007 | err=    0.000      0.000
# Corner[4] theta=6.283186 | DUT cos=0.999999993 sin=0.000000708 | MATLAB cos=1.000000007 sin=0.000000037 | err=    0.000      0.000
```

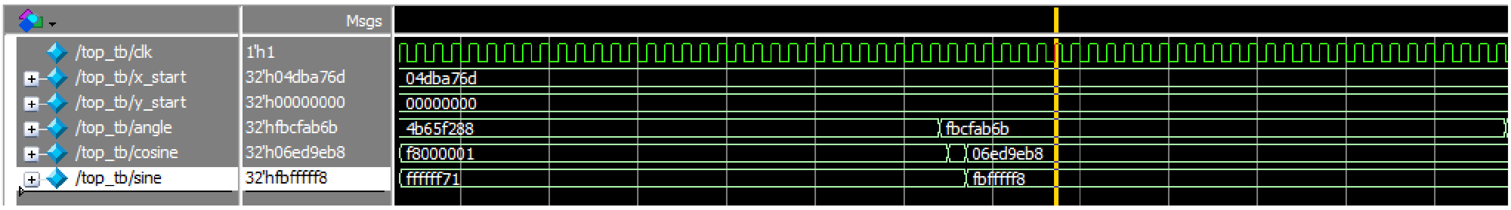
Waveform of Random Cases [pi/6, pi/4, 3pi/4, 3pi, 8pi]



Transcript of Random Cases:

```
# =========== Random Cases ==============
# Random[0] theta=0.523599 | DUT cos=0.866025388 sin=0.500000060 | MATLAB cos=0.866025418 sin=0.500000015 | err=0.000 0.000
# Random[1] theta=0.785398 | DUT cos=0.707106724 sin=0.707106821 | MATLAB cos=0.707106799 sin=0.707106747 | err=0.000 0.000
# Random[2] theta=2.356195 | DUT cos=-0.707106926 sin=0.707106620 | MATLAB cos=-0.707106799 sin=0.707106747 | err=0.000 0.000
# Random[3] theta=1.047198 | DUT cos=0.499999925 sin=0.866025470 | MATLAB cos=0.500000000 sin=0.866025433 | err=0.000 0.000
# Random[4] theta=0.392699 | DUT cos=0.923879504 sin=0.382683493 | MATLAB cos=0.923879519 sin=0.382683448 | err=0.000 0.000
```

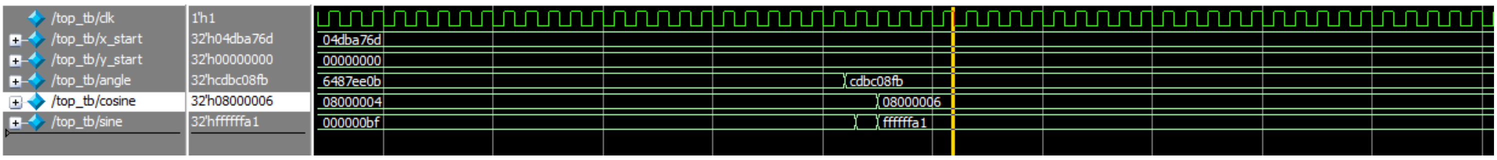
Waveform of overflow and underflow cases[3pi, -pi/6]:



Transcript of overflow and underflow cases:

```
# =========== Overflow Cases ============
# Over[0] theta=9.424779 | DUT cos=-0.999999993 sin=-0.000001065 | MATLAB cos=-1.000000007 sin=-0.000000037 | err=      0.00        0.00
# Over[1] theta=-0.523599 | DUT cos=0.866025388 sin=-0.500000060 | MATLAB cos=0.866025418 sin=-0.500000015 | err=      0.00        0.00
```

Waveform of boundary cases [-pi/2, 4pi]:



Transcript of boundary cases:

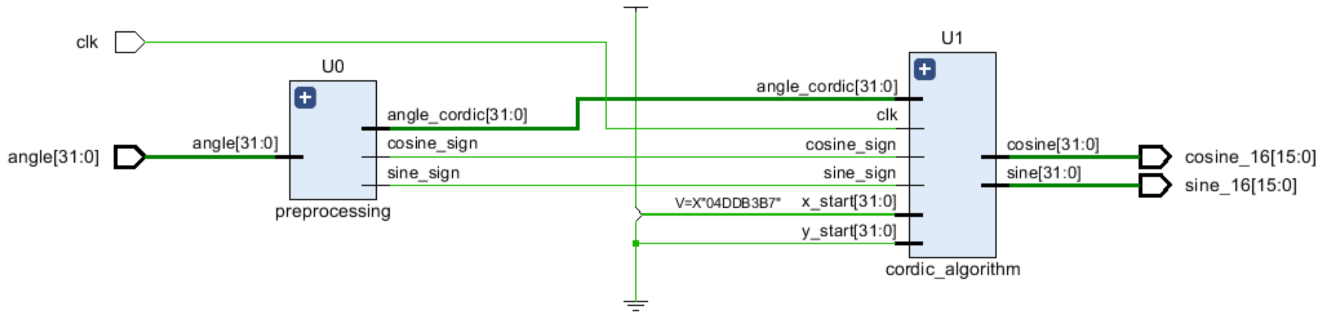
```
# =========== Boundary Cases ============
# Bound[0] theta=12.566372 | DUT cos=1.000000030 sin=0.000001423 | MATLAB cos=1.000000067 sin=0.000000022 | err=0.000  0.000
# Bound[1] theta=-6.283186 | DUT cos=1.000000045 sin=-0.000000708 | MATLAB cos=1.000000067 sin=-0.000000022 | err=0.000  0.000
#  All tests done.
```

FPGA Implementation:

The implementation of the project was carried out on the Zybo Z7-20 FPGA development board, which is based on the Xilinx Zynq-7000 SoC. This board provides both an ARM Cortex-A9 processing system and programmable logic, making it suitable for hardware prototyping and testing. The design was developed and synthesized using the Xilinx Vivado Design Suite, with the on-board 100 MHz clock used as the main clock source. After synthesis, placement, and routing, a bitstream was generated and programmed onto the Zybo Z7-20 board.

NOTE: To match the board’s I/O limitations, only the lower 16 bits of the sine and cosine outputs were used and x_start & y_start are entered as localparam, as the total number of output ports exceeded the available pins on the board.

Elaborated Design:



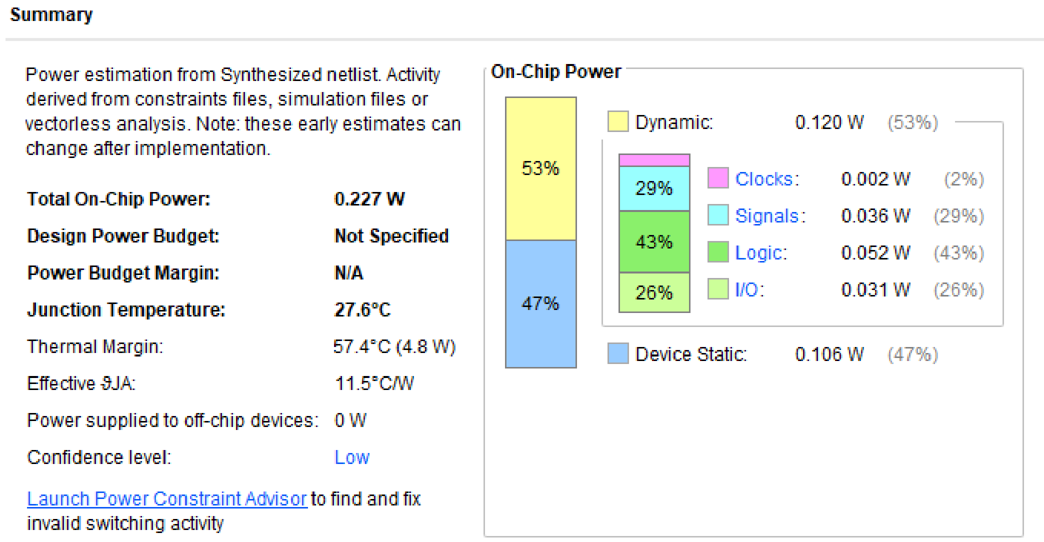
Timing summary after synthesis :

Design Timing Summary

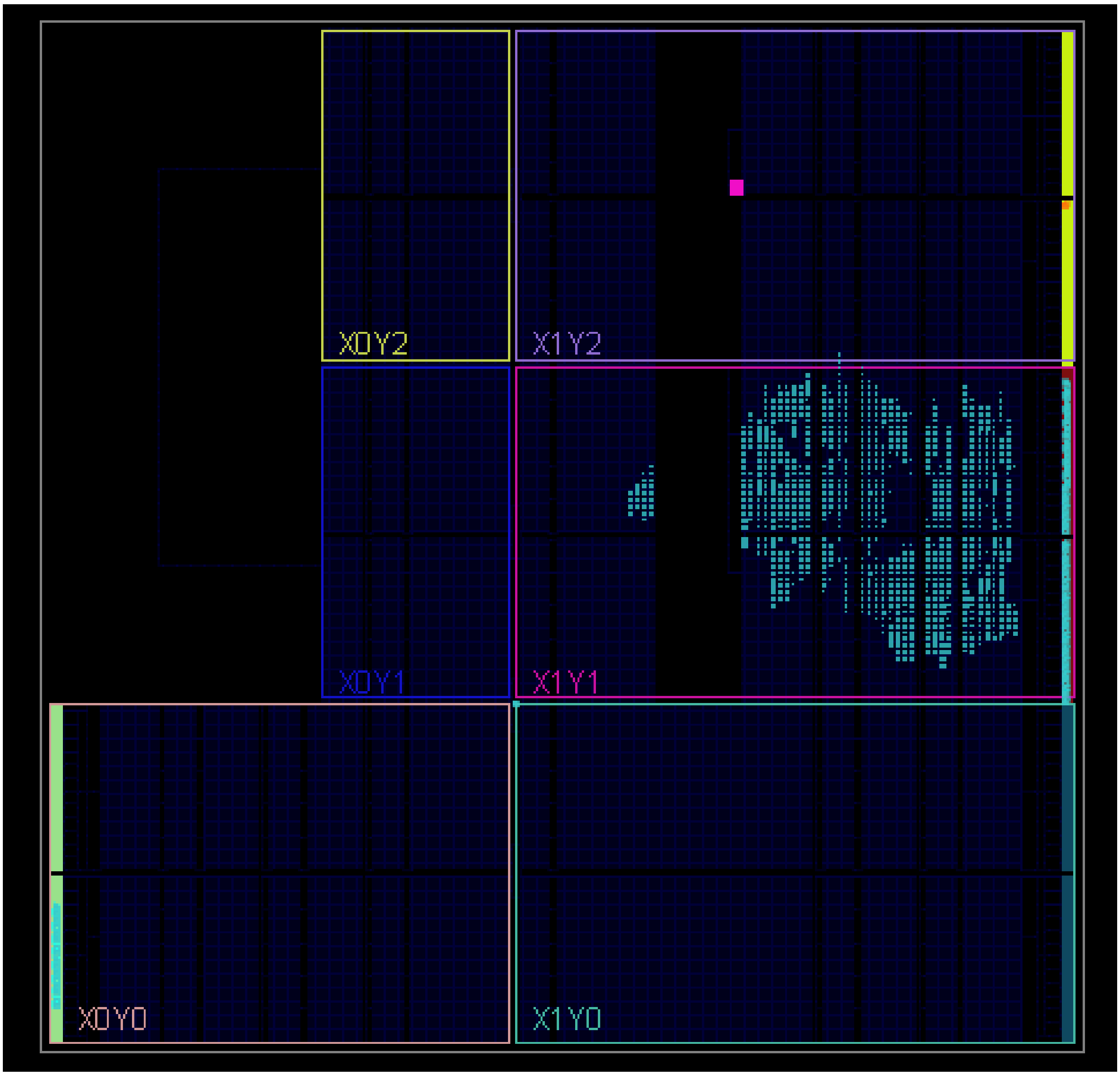
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.668 ns	Worst Hold Slack (WHS):	0.156 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	64	Total Number of Endpoints:	64	Total Number of Endpoints:	129

All user specified timing constraints are met.

Summary of power after synthesis:



Implementation



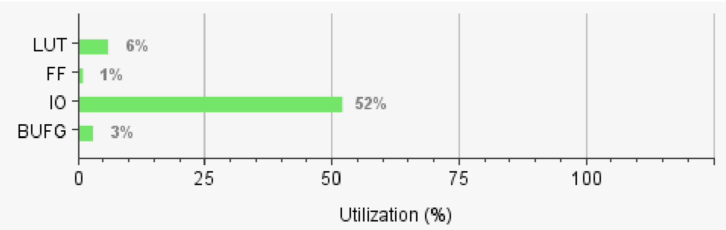
Timing Summary after implementation:

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.373 ns	Worst Hold Slack (WHS):	0.351 ns	Worst Pulse Width Slack (WPWS):	4.500 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:	32	Total Number of Endpoints:	32	Total Number of Endpoints:	95

All user specified timing constraints are met.

Utilization Report:



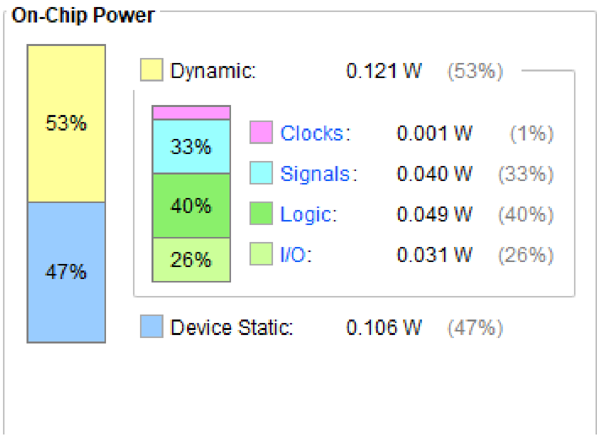
Resource	Utilization	Available	Utilization %
LUT	3451	53200	6.49
FF	94	106400	0.09
IO	65	125	52.00
BUFG	1	32	3.13

Summary of power after implementation:

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.227 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**27.6°C**
Thermal Margin:	57.4°C (4.8 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low



Clock of 100MHz :

Clocks Constraint Calculated

Utilization	Name	Frequency (MHz)	Buffer	Clock Buffer Enable (%)	Enable Signal	Bel Fanout	Sites	Fanout/Site
0.002 W (1% of total)	cordic_top							
0.002 W (1% of total)	clk	100.000	N/A	N/A	N/A	96	55	1.745