QR Decomposition

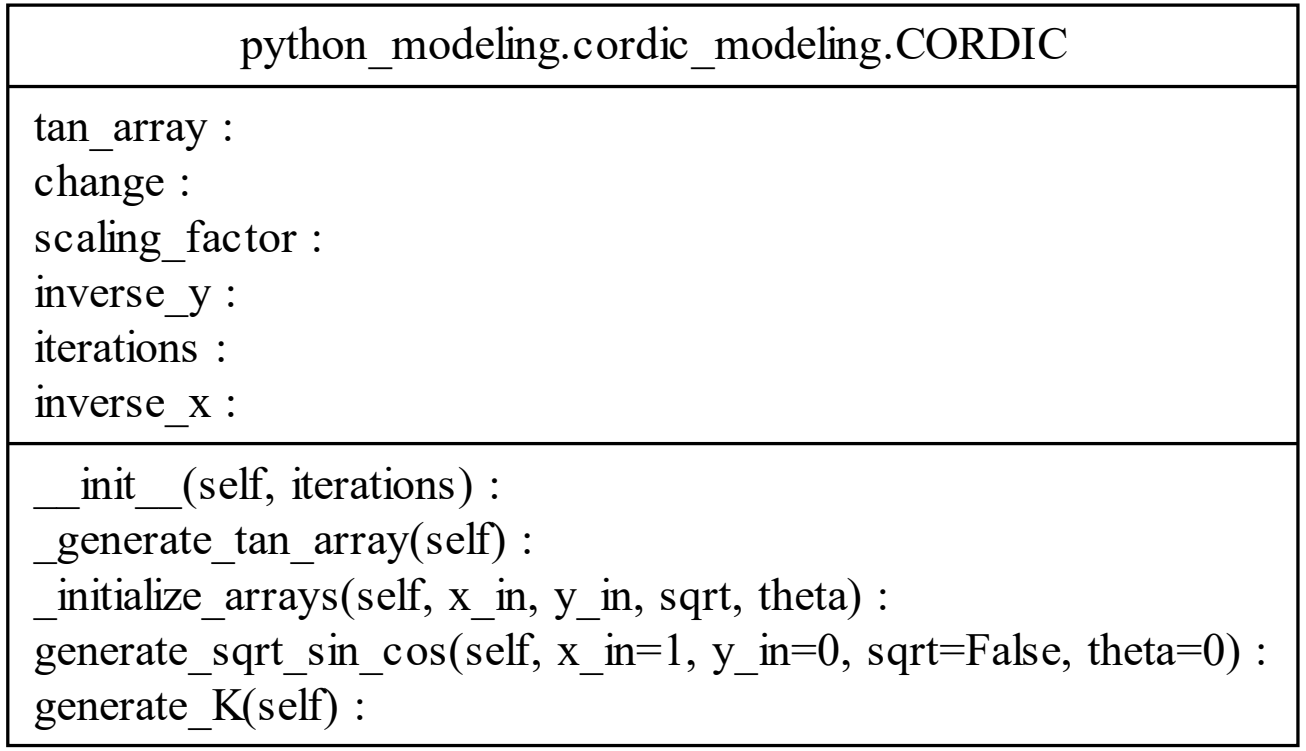
BY:

- Abdulrahman Elsadiq
- Mohab Tarek

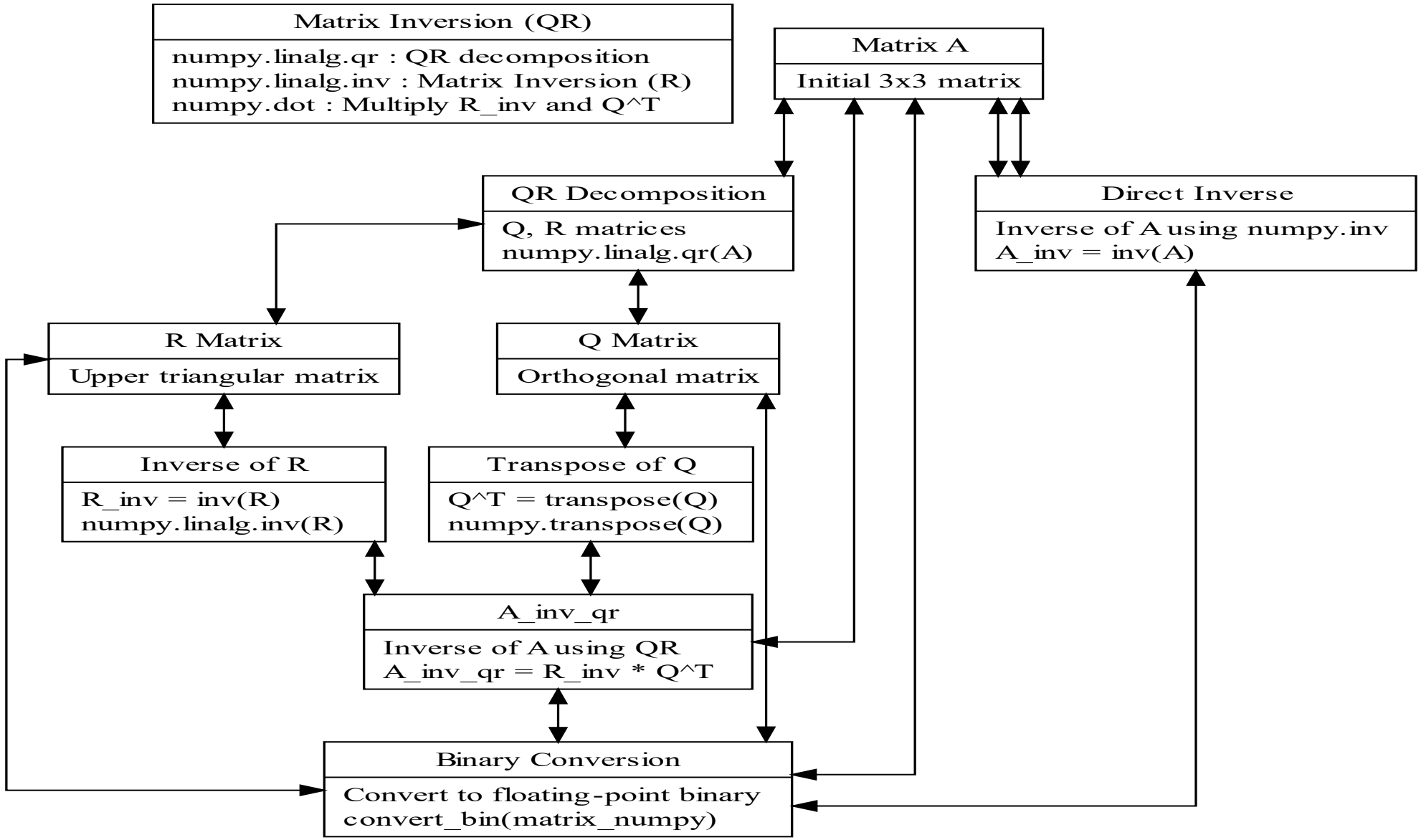
Agenda

- Modeling
- Architecture
- Code
- Simulation
- FPGA

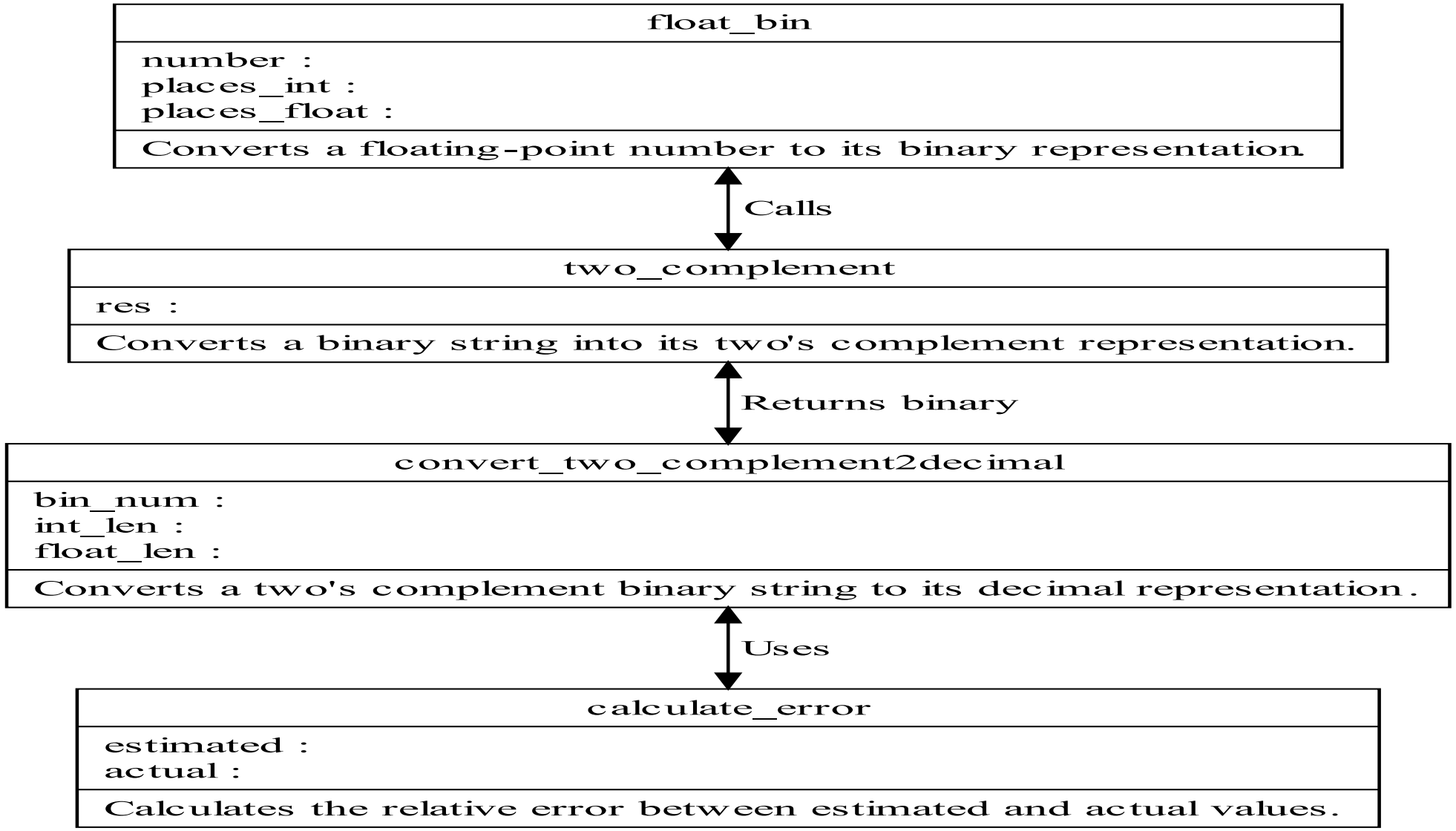
Cordic UML


python_modeling.cordic_modeling.CORDIC
tan_array :
change :
scaling_factor :
inverse_y :
iterations :
inverse_x :
__init__(self, iterations) :
_generate_tan_array(self) :
_initialize_arrays(self, x_in, y_in, sqrt, theta) :
generate_sqrt_sin_cos(self, x_in=1, y_in=0, sqrt=False, theta=0) :
generate_K(self) :

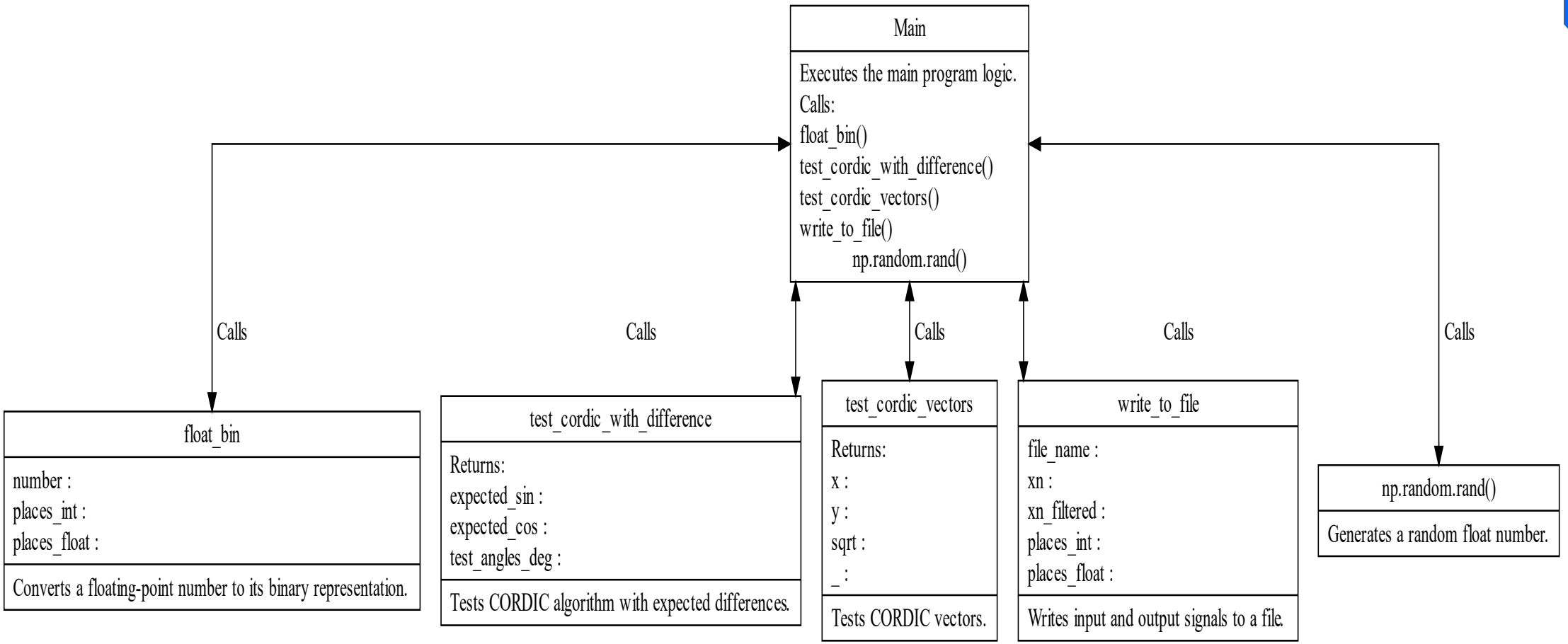
Matix inverse using QR



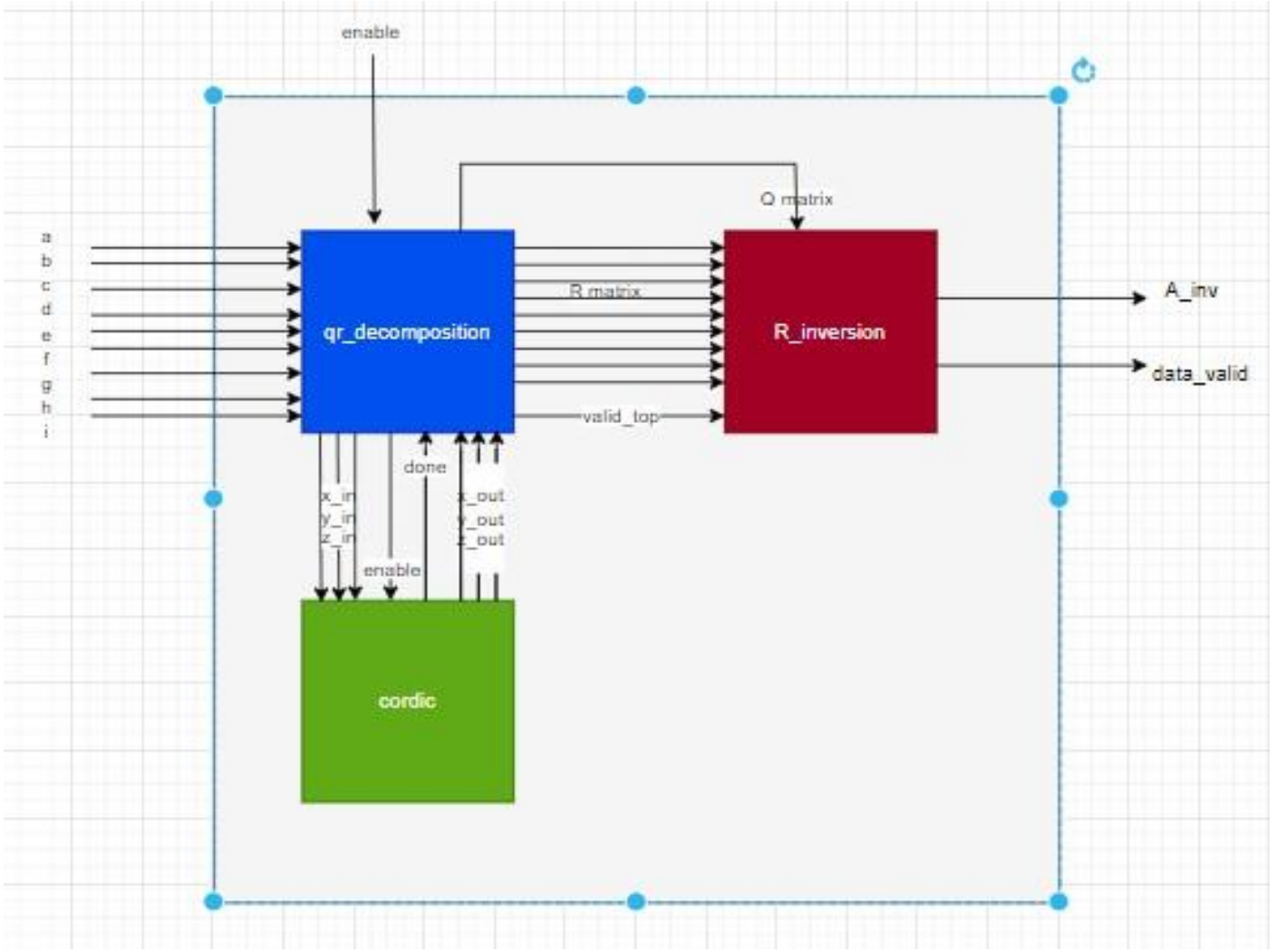
Binary conversion

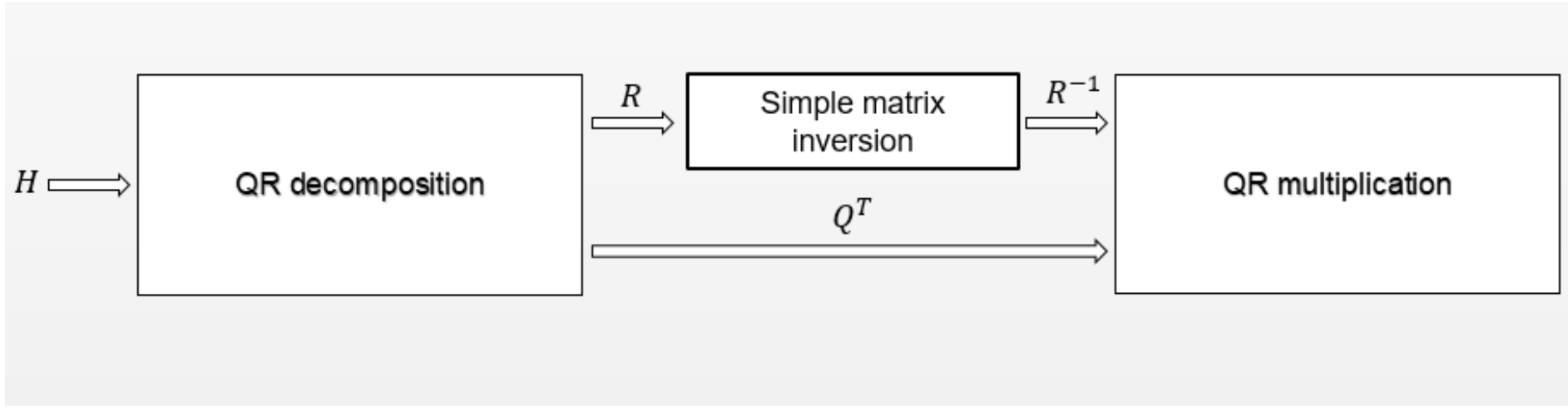


Main test

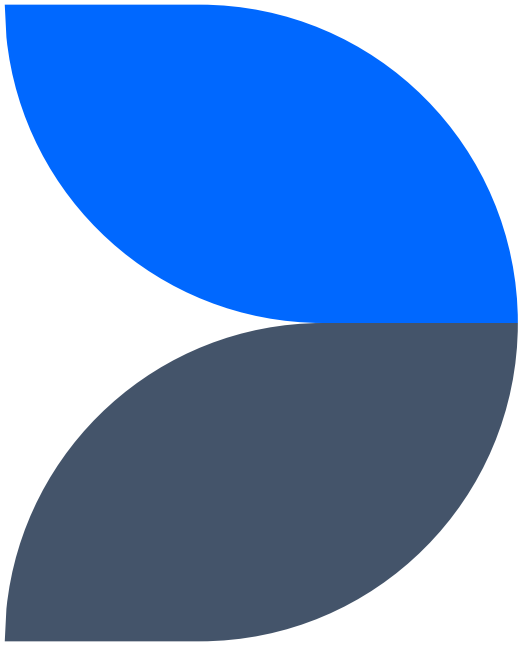

Main
Executes the main program logic.
Calls:
float_bin()
test_cordic_with_difference()
test_cordic_vectors()
write_to_file()
np.random.rand()
Calls
Calls
Calls
Calls
Calls
float_bin
number :
places_int :
places_float :
Converts a floating-point number to its binary representation.
test_cordic_with_difference
Returns:
expected_sin :
expected_cos :
test_angles_deg :
Tests CORDIC algorithm with expected differences.
test_cordic_vectors
Returns:
x :
y :
sqrt :
_ :
Tests CORDIC vectors.
write_to_file
file_name :
xn :
xn_filtered :
places_int :
places_float :
Writes input and output signals to a file.
np.random.rand()
Generates a random float number.

Architecture




H
QR decomposition
R
Simple matrix inversion
R^{-1}
QR multiplication
Q^T

Code



```
module cordic (
    input logic clk,
    input logic rst_n,
    input logic select,
    input logic enable,
    input logic signed [31:0] x_in,
    input logic signed [31:0] y_in,
    input logic signed [31:0] z_in,
    output logic signed [31:0] x_out,
    output logic signed [31:0] y_out,
    output logic signed [31:0] z_out,
    output logic done
);
    // Parameters for CORDIC
    parameter ITER = 15;

    // Internal registers and wires
    logic signed [31:0] x [0:ITER-1];
    logic signed [31:0] y [0:ITER-1];
    logic signed [31:0] z [0:ITER-1];
    logic done_1;

    // Arctangent lookup table
    // q8.24 fixed-point format
    logic signed [31:0] arctan_table [0:14] = '{
        32'sb00000000110010010000111111011010,
        32'sb00000000011101101011000110011100,
        32'sb00000000001111101011011011101011,
        32'sb00000000000111111010101101110010,
        32'sb00000000000011111111010101101101,
        32'sb00000000000001111111111101010101,
        32'sb00000000000000111111111111101010,
        32'sb00000000000000011111111111111101,
        32'sb00000000000000001111111111111111,
        32'sb00000000000000000111111111111111,
        32'sb00000000000000000011111111111111,
        32'sb00000000000000000001111111111111,
        32'sb00000000000000000000111111111111,
        32'sb00000000000000000000011111111111,
        32'sb00000000000000000000001111111111
    };
    logic signed [31:0] scaling_factor = 32'b00000000100110110111010011101101;  // Adjust as needed
    logic signed [63:0] x_out_64,y_out_64;
```

Cordic block

```
    always_comb begin
        if (enable) begin
            // Initialize inputs
            x[0] = x_in;
            y[0] = y_in;
            z[0] = z_in;

            for (int i = 0; i < ITER-1; i++) begin
                if (!select) begin
                    // Rotational Mode
                    if (z[i] >= 0) begin
                        x[i+1] = x[i] - (y[i] >>> i);
                        y[i+1] = y[i] + (x[i] >>> i);
                        z[i+1] = z[i] - arctan_table[i];
                    end else begin
                        x[i+1] = x[i] + (y[i] >>> i);
                        y[i+1] = y[i] - (x[i] >>> i);
                        z[i+1] = z[i] + arctan_table[i];
                    end
                end else begin
                    // Vectoring Mode
                    if (y[i] >= 0) begin
                        x[i+1] = x[i] + (y[i] >>> i);
                        y[i+1] = y[i] - (x[i] >>> i);
                        z[i+1] = z[i] + arctan_table[i];
                    end else begin
                        x[i+1] = x[i] - (y[i] >>> i);
                        y[i+1] = y[i] + (x[i] >>> i);
                        z[i+1] = z[i] - arctan_table[i];
                    end
                end
            end

            // Assert done after last iteration
            done_1 = 1;
            x_out_64=x[ITER-1] * scaling_factor;
            y_out_64=y[ITER-1] * scaling_factor;
        end else begin
            done_1 = 0;
        end
    end
```

```
    // Output the final iteration values
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            // Reset state
            done <= 0;
            x_out <= 0;
            y_out <= 0;
            z_out <= 0;
        end else begin
            // Scaling factor to convert the final result back to proper format
            // Multiply by scaling factor, considering fixed-point format (q8.24)
            x_out <= x_out_64[55:24];  // Scaling down to correct fractional bits
            y_out <= y_out_64[55:24];
            z_out <= z[ITER-1];
            done <= done_1;
        end
    end
endmodule
```

```
module qr_decomposition (
    input logic clk,
    input logic rst_n,
    input logic enable,
    input logic done1, // the cordic out is ready
    input logic signed  [31:0] a, b, c,
    input logic signed  [31:0] d, e, f,
    input logic signed  [31:0] g, h, i,
    input logic signed  [31:0] cor_x_out,
    input logic signed  [31:0] cor_y_out,
    input logic signed  [31:0] cor_z_out,
    output logic               enable_out,
    output logic signed [31:0] cor_x_in,
    output logic signed [31:0] cor_y_in,
    output logic signed [31:0] cor_z_in,
    output logic               select_out, // for kind of cordic
    output logic signed [31:0] a_out, b_out, c_out,
    output logic signed [31:0] d_out, e_out, f_out,
    output logic signed [31:0] g_out, h_out, i_out,
    output logic signed [31:0] Q [2:0][2:0],
    output logic               data_valid
);

logic [5:0] current_state, next_state;
parameter IDLE        = 'b00000,
          phase1_1_in = 'b00001,
          phase1_1_out = 'b00010,
          phase1_2_in = 'b00011,
          phase1_2_out = 'b00100,
          phase1_3_in = 'b00101,
          phase1_3_out = 'b00110,
          phase2_1_in = 'b00111,
          phase2_1_out = 'b01000,
          phase2_2_in = 'b01001,
          phase2_2_out = 'b01010,
          phase2_3_in = 'b01011,
          phase2_3_out = 'b01100,
          phase3_1_in = 'b01101,
          phase3_1_out = 'b01110,
          phase3_2_in = 'b01111,
          phase3_2_out = 'b10000,
          multiply    = 'b10001;

logic signed [31:0] R [2:0][2:0];
logic signed [31:0] a_prime, b_prime, c_prime, d_prime, e_prime, f_prime;
logic signed [31:0] g_prime, h_prime, i_prime, theta1;
logic signed [31:0] phay1 [2:0][2:0], phay2 [2:0][2:0], phay3 [2:0][2:0];
logic signed [31:0] temp [2:0][2:0]; // Intermediate matrix after phay1 * phay2
```

```
// State transition logic
always_ff @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        current_state <= IDLE;
    end else begin
        current_state <= next_state;
    end
end

// Next state logic
always_comb begin
    case(current_state)
        IDLE: begin
            if (enable)
                next_state = phase1_1_in;
            else
                next_state = IDLE;
        end
        phase1_1_in: next_state = phase1_1_out;
        phase1_1_out: begin
            if (done1)
                next_state = phase1_2_in;
            else
                next_state = phase1_1_out;
        end
        phase1_2_in: next_state = phase1_2_out;
        phase1_2_out: begin
            if (done1)
                next_state = phase1_3_in;
            else
                next_state = phase1_2_out;
        end
        phase1_3_in: next_state = phase1_3_out;
        phase1_3_out: begin
            if (done1)
                next_state = phase2_1_in;
            else
                next_state = phase1_3_out;
        end
```

QR Decomposition block

```
        phase2_1_in: next_state = phase2_1_out;
        phase2_1_out: begin
            if (done1)
                next_state = phase2_2_in;
            else
                next_state = phase2_1_out;
        end
        phase2_2_in: next_state = phase2_2_out;
        phase2_2_out: begin
            if (done1)
                next_state = phase2_3_in;
            else
                next_state = phase2_2_out;
        end
        phase2_3_in: next_state = phase2_3_out;
        phase2_3_out: begin
            if (done1)
                next_state = phase3_1_in;
            else
                next_state = phase2_3_out;
        end
        phase3_1_in: next_state = phase3_1_out;
        phase3_1_out: begin
            if (done1)
                next_state = phase3_2_in;
            else
                next_state = phase3_1_out;
        end
        phase3_2_in: next_state = phase3_2_out;
        phase3_2_out: begin
            if (done1)
                next_state = multiply;
            else
                next_state = phase3_2_out;
        end
        multiply: begin
            if (data_valid)
                next_state = IDLE;
            else
                next_state = multiply;
        end
        default: next_state = IDLE;
    endcase
end
```

```
// Output logic
always_comb begin

    case(current_state)
        IDLE: begin
            cor_x_in = 0;
            cor_y_in = 0;
            cor_z_in = 0;
        end
        phase1_1_in: begin
            enable_out = 1;
            cor_x_in = a;
            cor_y_in = d;
            cor_z_in = 0;
            select_out = 1;
        end
        phase1_1_out: begin
            enable_out = 0;
            a_prime = cor_x_out;
            theta1 = cor_y_out;
        end
        phase1_2_in: begin
           enable_out = 1;
            cor_x_in = b;
            cor_y_in = e;
            cor_z_in = theta1;
            select_out = 0;
        end
        phase1_2_out: begin
           enable_out = 0;
            b_prime = cor_x_out;
            e_prime = cor_y_out;
        end
        phase1_3_in: begin
           enable_out = 1;
            cor_x_in = c;
            cor_y_in = f;
            cor_z_in = theta1;
            select_out = 0;
        end
```

```
        phase1_3_out: begin
            enable_out = 0;
             c_prime = cor_x_out;
             f_prime = cor_y_out;
             R[0][0] = a_prime;
             R[0][1] = b_prime;
             R[0][2] = c_prime;
             R[1][0] = 0;
             R[1][1] = e_prime;
             R[1][2] = f_prime;
             R[2][0] = g;
             R[2][1] = h;
             R[2][2] = i;
             phay1[0][0] = c_prime;
             phay1[0][1] = f_prime;
             phay1[0][2] = 0;
             phay1[1][0] = -f_prime;
             phay1[1][1] = c_prime;
             phay1[1][2] = 0;
             phay1[2][0] = 0;
             phay1[2][1] = 0;
             phay1[2][2] = 1;
        end
        phase2_1_in: begin
            enable_out = 1;
             cor_x_in = R[0][0];
             cor_y_in = R[2][0];
             cor_z_in = 0;
             select_out = 1;
        end
        phase2_1_out: begin
            enable_out = 0;
             a_prime = cor_x_out;
             theta1 = cor_y_out;
        end
        phase2_2_in: begin
            enable_out = 1;
             cor_x_in = R[0][1];
             cor_y_in = R[2][1];
             cor_z_in = theta1;
             select_out = 0;
        end
        phase2_2_out: begin
            enable_out = 0;
             b_prime = cor_x_out;
             h_prime = cor_y_out;
        end
        phase2_3_in: begin
            enable_out = 1;
             cor_x_in = R[0][2];
             cor_y_in = R[2][2];
             cor_z_in = theta1;
             select_out = 0;
        end
```

```
        phase2_3_out: begin
            enable_out = 0;
             c_prime = cor_x_out;
             i_prime = cor_y_out;
             R[0][0] = a_prime;
             R[0][1] = b_prime;
             R[0][2] = c_prime;
             R[2][0] = 0;
             R[2][1] = h_prime;
             R[2][2] = i_prime;
             phay2[0][0] = c_prime;
             phay2[0][1] = i_prime;;
             phay2[0][2] = 0;
             phay2[1][0] = 0;
             phay2[1][1] = 1;
             phay2[1][2] = 0;
             phay2[2][0] = - i_prime;
             phay2[2][1] = c_prime;
             phay2[2][2] = 1;
        end
        phase3_1_in: begin
            enable_out = 1;
             cor_x_in = R[1][1];
             cor_y_in = R[2][1];
             cor_z_in = 0;
             select_out = 1;
        end
        phase3_1_out: begin
            enable_out = 0;
             e_prime = cor_x_out;
             theta1 = cor_y_out;
        end
        phase3_2_in: begin
            enable_out = 1;
             cor_x_in = R[1][2];
             cor_y_in = R[2][2];
             cor_z_in = theta1;
             select_out = 0;
        end
```

```
        phase3_2_out: begin
            enable_out = 0;
            f_prime = cor_x_out;
            i_prime = cor_y_out;
            R[1][1] = e_prime;
            R[1][2] = f_prime;
            R[2][1] = 0;
            R[2][2] = i_prime;
            phay3[0][0] = 1;
            phay3[0][1] = 0;
            phay3[0][2] = 0;
            phay3[1][0] = 0;
            phay3[1][1] = f_prime;
            phay3[1][2] = i_prime;
            phay3[2][0] = 0;
            phay3[2][1] = - i_prime;
            phay3[2][2] = f_prime;
        end
        multiply: begin
            for (int i = 0; i < 3; i++) begin
              for (int j = 0; j < 3; j++) begin
                temp[i][j] = 0;
                  for (int k = 0; k < 3; k++) begin
                    temp[i][j] = temp[i][j] + (phay1[i][k] * phay2[k][j]);
                  end
              end
            end
              for (int i = 0; i < 3; i++) begin
                for (int j = 0; j < 3; j++) begin
                  Q[i][j] = 0;
                  for (int k = 0; k < 3; k++) begin
                    Q[i][j] = Q[i][j] + (temp[i][k] * phay3[k][j]);
                  end
              end
            end

            data_valid = 1;
        end
    endcase
end

always_comb begin
    a_out= R[0][0];
    b_out= R[0][1];
    c_out= R[0][2];
    d_out= R[1][0];
    e_out= R[1][1];
    f_out= R[1][2];
    g_out= R[2][0];
    h_out= R[2][1];
    i_out= R[2][2];

end
endmodule
```

```
module R_inversion (

    input logic clk,
    input logic rst_n,
    input logic signed [31:0] a_in, b_in, c_in,
    input logic signed [31:0] d_in, e_in, f_in,
    input logic signed [31:0] g_in, h_in, i_in,
    input logic signed [31:0] Q [2:0][2:0],
    input logic               start,
    output logic signed [31:0] A_inv [2:0][2:0],
    output logic data_valid
);

    logic [1:0] current_state, next_state;
    logic signed [31:0] r11_inv, r22_inv, r33_inv; // Diagonal inverses
    logic signed [31:0] R [2:0][2:0];
    logic signed [31:0] R_inv [2:0][2:0]; // Inverse of R matrix (3x3)
    logic signed [31:0] Q_T [2:0][2:0];
    logic done;
    typedef enum logic [1:0] {
        IDLE,
        INVERT_DIAGONAL,
        INVERT_OFF_DIAGONAL,
        DONE
    } state_t;

always_comb begin
    R[0][0] = a_in;
    R[0][1] = b_in;
    R[0][2] = c_in;
    R[1][0] = d_in;
    R[1][1] = e_in;
    R[1][2] = f_in;
    R[2][0] = g_in;
    R[2][1] = h_in;
    R[2][2] = i_in;
    Q_T[0][0] = Q[0][0];
    Q_T[0][1] = Q[1][0];
    Q_T[0][2] = Q[2][0];
    Q_T[1][0] = Q[0][1];
    Q_T[1][1] = Q[1][1];
    Q_T[1][2] = Q[2][1];
    Q_T[2][0] = Q[0][2];
    Q_T[2][1] = Q[1][2];
    Q_T[2][2] = Q[2][2];

end
```

```
    // State Transition
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            current_state <= IDLE;
        else
            current_state <= next_state;
    end

    // Next State Logic
    always_comb begin
        case (current_state)
            IDLE:
                if (start)
                    next_state = INVERT_DIAGONAL;
                else
                    next_state = IDLE;
            INVERT_DIAGONAL:
                next_state = INVERT_OFF_DIAGONAL;
            INVERT_OFF_DIAGONAL:
                next_state = DONE;
            DONE:
                next_state = IDLE;
            default:
                next_state = IDLE;
        endcase
    end
```

R_inversion block

```
    // Diagonal Inversion
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            r11_inv <= 0;
            r22_inv <= 0;
            r33_inv <= 0;
            R_inv[0][0] <= 0;
            R_inv[1][1] <= 0;
            R_inv[2][2] <= 0;
        end
        else if (current_state == INVERT_DIAGONAL) begin
            // Invert diagonal elements (if non-zero)
            r11_inv <= (R[0][0] != 0) ? (1 << 16) / R[0][0] : 0;  // Fixed-point inversion (1/R)
            r22_inv <= (R[1][1] != 0) ? (1 << 16) / R[1][1] : 0;
            r33_inv <= (R[2][2] != 0) ? (1 << 16) / R[2][2] : 0;

            // Store the inverted diagonal elements
            R_inv[0][0] <= r11_inv;
            R_inv[1][1] <= r22_inv;
            R_inv[2][2] <= r33_inv;
        end
    end

    // Off-diagonal Inversion using Back Substitution
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            R_inv[0][1] <= 0;
            R_inv[0][2] <= 0;
            R_inv[1][2] <= 0;
        end
        else if (current_state == INVERT_OFF_DIAGONAL) begin
            // Invert the off-diagonal elements using back substitution

            // R_inv[1,2] = R[1,2] / (R[1,1] * R[2,2])
            R_inv[1][2] <= ((R[1][2] * r22_inv * r33_inv) >> 16);  // Fixed-point multiplication

            // R_inv[0,1] = -R[0,1] / (R[0,0] * R[1,1])
            R_inv[0][1] <= -((R[0][1] * r11_inv * r22_inv) >> 16);  // Fixed-point multiplication

            // R_inv[0,2] = -R[0,2] / R[0,0] - (R[0,1] * R[1,2]) / (R[0,0] * R[1,1] * R[2,2])
            R_inv[0][2] <= -(((R[0][2] * r11_inv * r33_inv) >> 16) +
                             ((R[0][1] * R[1][2] * r11_inv * r22_inv * r33_inv) >> 32));  // Second-order fixed-point multiplication
        end
    end
```

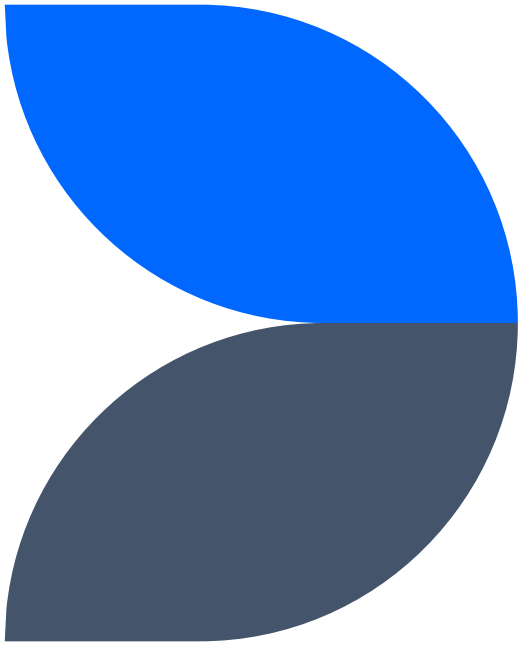
```
    assign done = (current_state == DONE);
always_comb begin
   if(done)begin
      for (int i = 0; i < 3; i++) begin
                for (int j = 0; j < 3; j++) begin
                  A_inv[i][j] = 0;
                    for (int k = 0; k < 3; k++) begin
                      A_inv[i][j] = A_inv[i][j] + (R_inv[i][k] * Q_T[k][j]);
                    end
                end
            end
      data_valid = 1;
   end else begin
      data_valid = 0;
      end
end
endmodule
```

```
module top (
    input logic clk,
    input logic rst_n,
    input logic enable,
    input logic signed  [31:0] a, b, c,
    input logic signed  [31:0] d, e, f,
    input logic signed  [31:0] g, h, i,
    output logic signed [31:0] A_inv [2:0][2:0],
    output logic               data_valid

);



    logic done1_top;
    logic enable_top;
    logic signed [31:0] cor_x_out_top;
    logic signed [31:0] cor_y_out_top;
    logic signed [31:0] cor_z_out_top;
    logic signed [31:0] cor_x_in_top;
    logic signed [31:0] cor_y_in_top;
    logic signed [31:0] cor_z_in_top;
    logic               select_out_top;
    logic               valid_top;
    logic signed [31:0] Q_top [2:0][2:0];
    logic signed [31:0] a_out, b_out, c_out;
    logic signed [31:0] d_out, e_out, f_out;
    logic signed [31:0] g_out, h_out, i_out;

   cordic u0_cordic (
.clk(clk),
.rst_n(rst_n),
.select(select_out_top),
.enable(enable_top),
.x_in(cor_x_in_top),
.y_in(cor_y_in_top),
.z_in(cor_z_in_top),
.x_out(cor_x_out_top),
.y_out(cor_y_out_top),
.z_out(cor_z_out_top),
.done(done1_top)
);
```

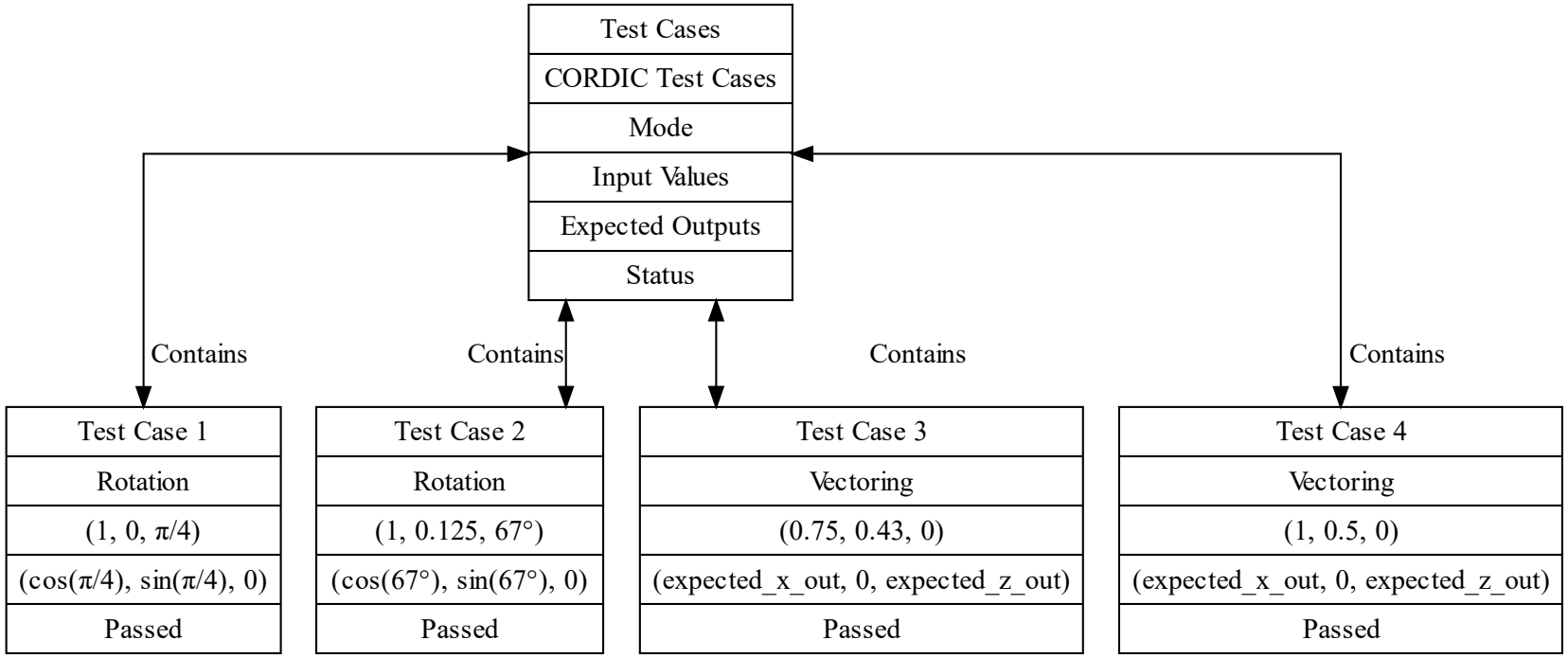
```
 qr_decomposition u0_qr_decomposition (
.clk(clk),
.rst_n(rst_n),
.enable(enable),
.done1(done1_top),
.a(a),
.b(b),
.c(c),
.d(d),
.e(e),
.f(f),
.g(g),
.h(h),
.i(i),
.enable_out(enable_top),
.cor_x_out(cor_x_out_top),
.cor_y_out(cor_y_out_top),
.cor_z_out(cor_z_out_top),
.cor_x_in(cor_x_in_top),
.cor_y_in(cor_y_in_top),
.cor_z_in(cor_z_in_top),
.select_out(select_out_top),
.a_out(a_out),
.b_out(b_out),
.c_out(c_out),
.d_out(d_out),
.e_out(e_out),
.f_out(f_out),
.g_out(g_out),
.h_out(h_out),
.i_out(i_out),
.Q(Q_top),
.data_valid(valid_top)
);

R_inversion u0_R_inversion(

.clk(clk),
.rst_n(rst_n),
.a_in(a_out),
.b_in(b_out),
.c_in(c_out),
.d_in(d_out),
.e_in(e_out),
.f_in(f_out),
.g_in(g_out),
.h_in(h_out),
.i_in(i_out),
.Q(Q_top),
.start(valid_top),
.A_inv(A_inv),
.data_valid(data_valid)
 );
endmodule
```

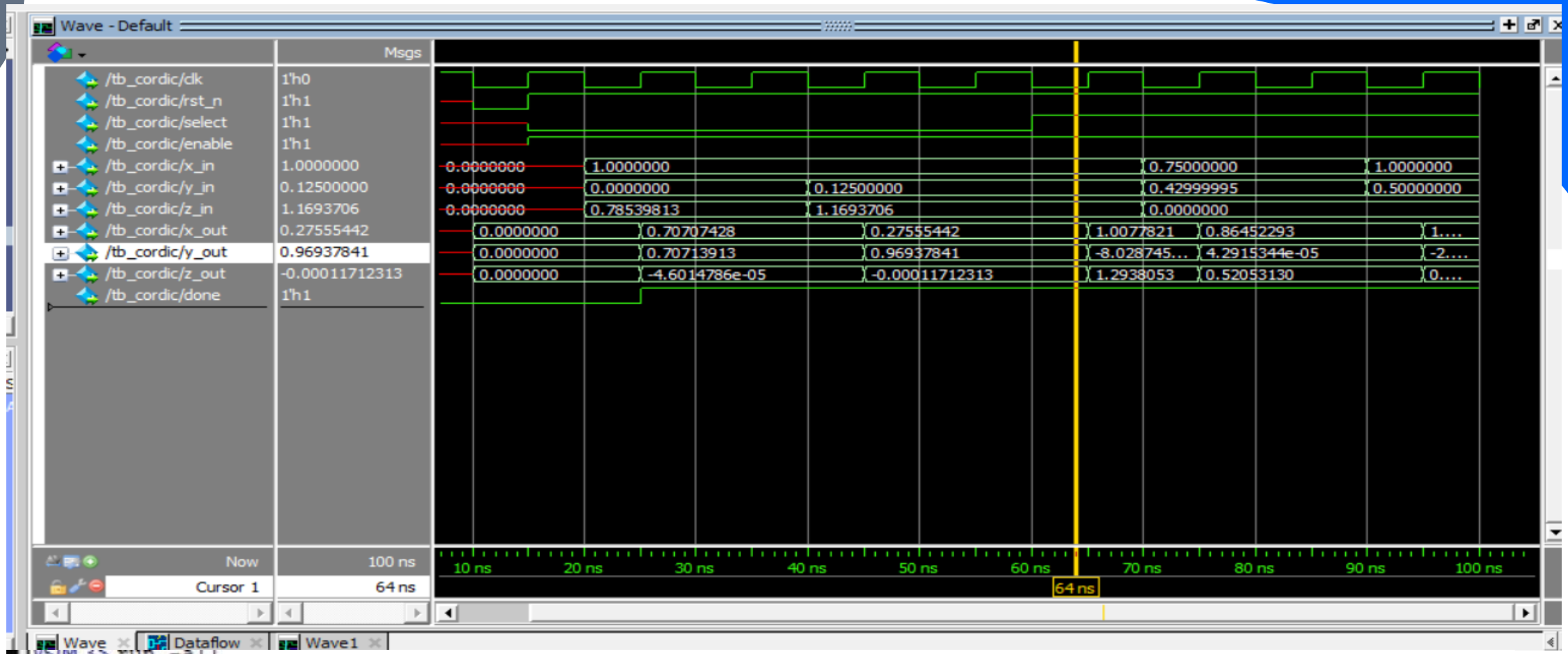
TOP module

Simulation



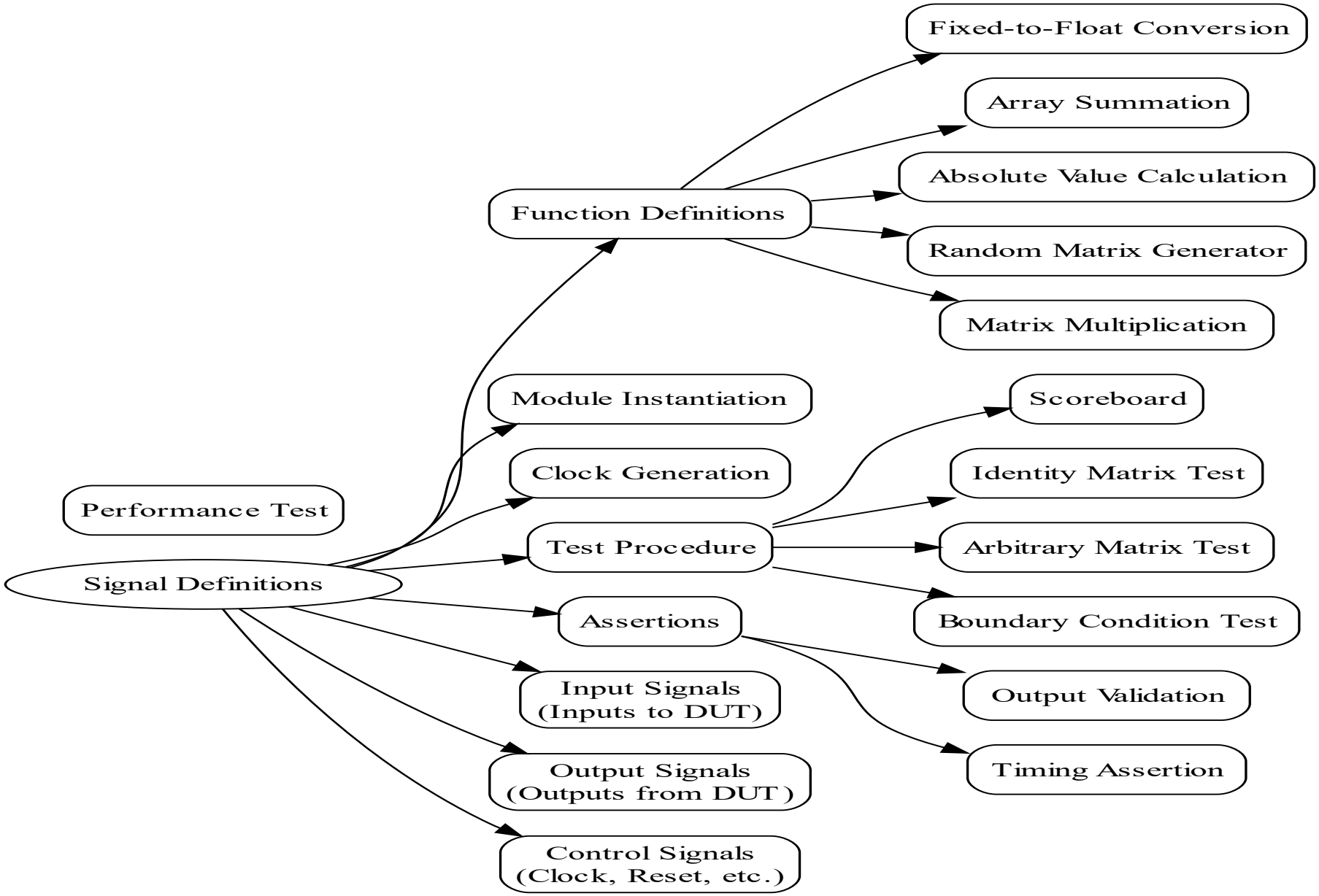
Cordic tests






```
VSIM 3> run -all
# Rotation Mode
# vectoring Mode
# Number of passed cases:5
# Number of failed cases:0
```

System test


Fixed-to-Float Conversion
Array Summation
Absolute Value Calculation
Function Definitions
Random Matrix Generator
Matrix Multiplication
Module Instantiation
Scoreboard
Clock Generation
Identity Matrix Test
Performance Test
Test Procedure
Arbitrary Matrix Test
Signal Definitions
Boundary Condition Test
Assertions
Input Signals
(Inputs to DUT)
Output Validation
Output Signals
(Outputs from DUT)
Timing Assertion
Control Signals
(Clock, Reset, etc.)

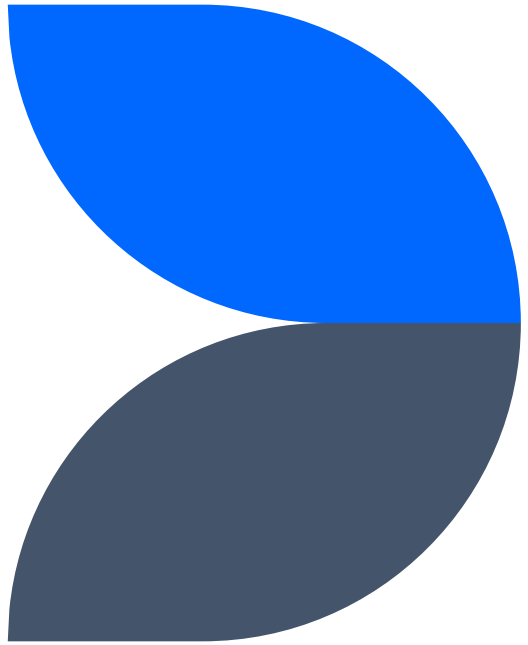
Results

```
#
# Test Case 1: Identity matrix
#
# input Matrix
#
# a=1.000000, b=0.000000, c=0.000000
#
# d=0.000000, e_=1.000000, f=0.000000
#
# g=0.000000, h=0.000000, i=1.000000
#
```

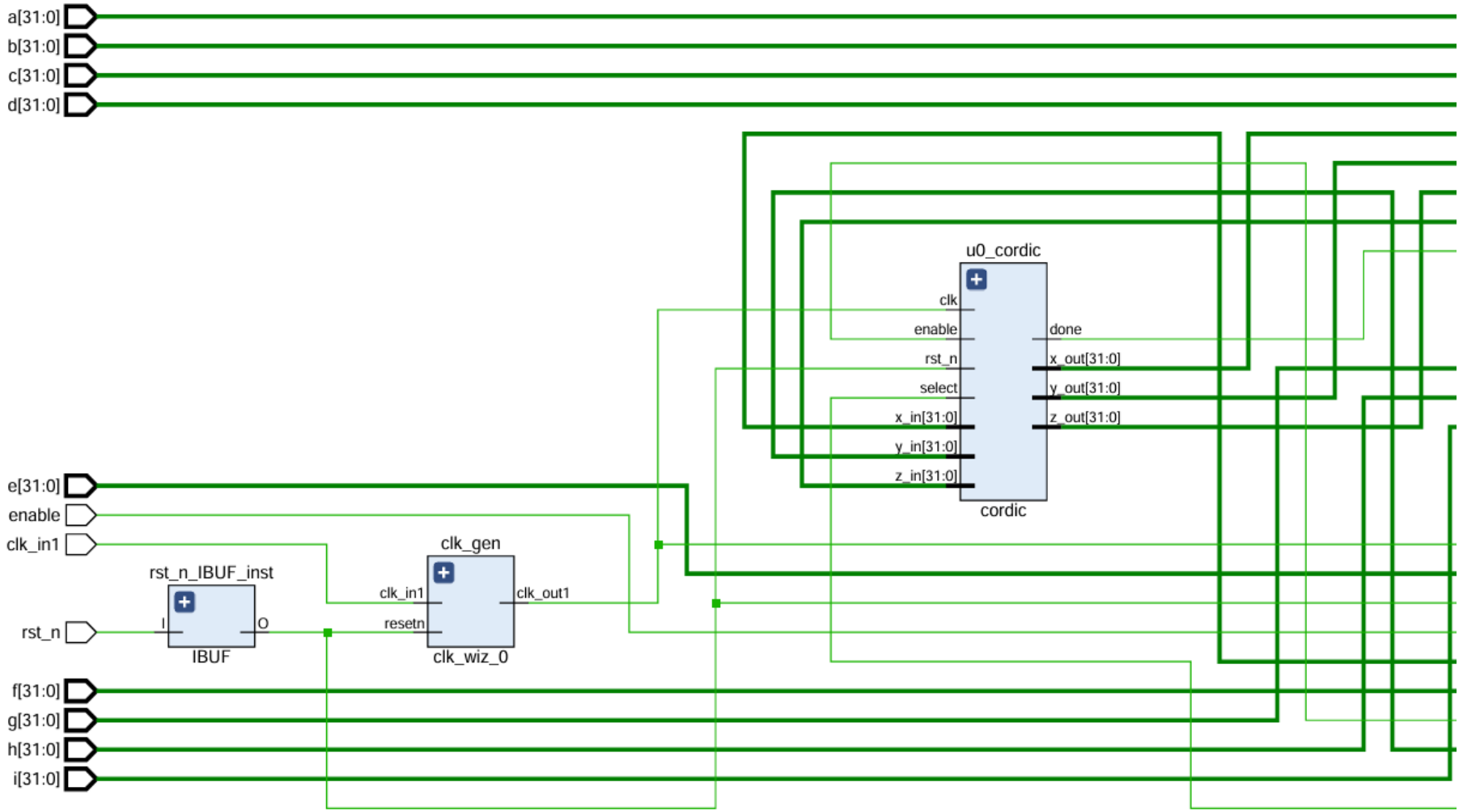
```
# Result (R):
#
# a_out_r=1.000000, b_out_r=0.000074, c_out_r=0.000074
#
# d_out_r=0.000000, e_out_r=1.000000, f_out_r=-0.000074
#
# g_out_r=0.000000, h_out_r=0.000000, i_out_r=1.000000
#
# Result (R expected):
#
# a_out_r=1.000000, b_out_r=0.000000, c_out_r=0.000000
#
# d_out_r=0.000000, e_out_r=1.000000, f_out_r=0.000000
#
# g_out_r=0.000000, h_out_r=0.000000, i_out_r=1.000000
#
```

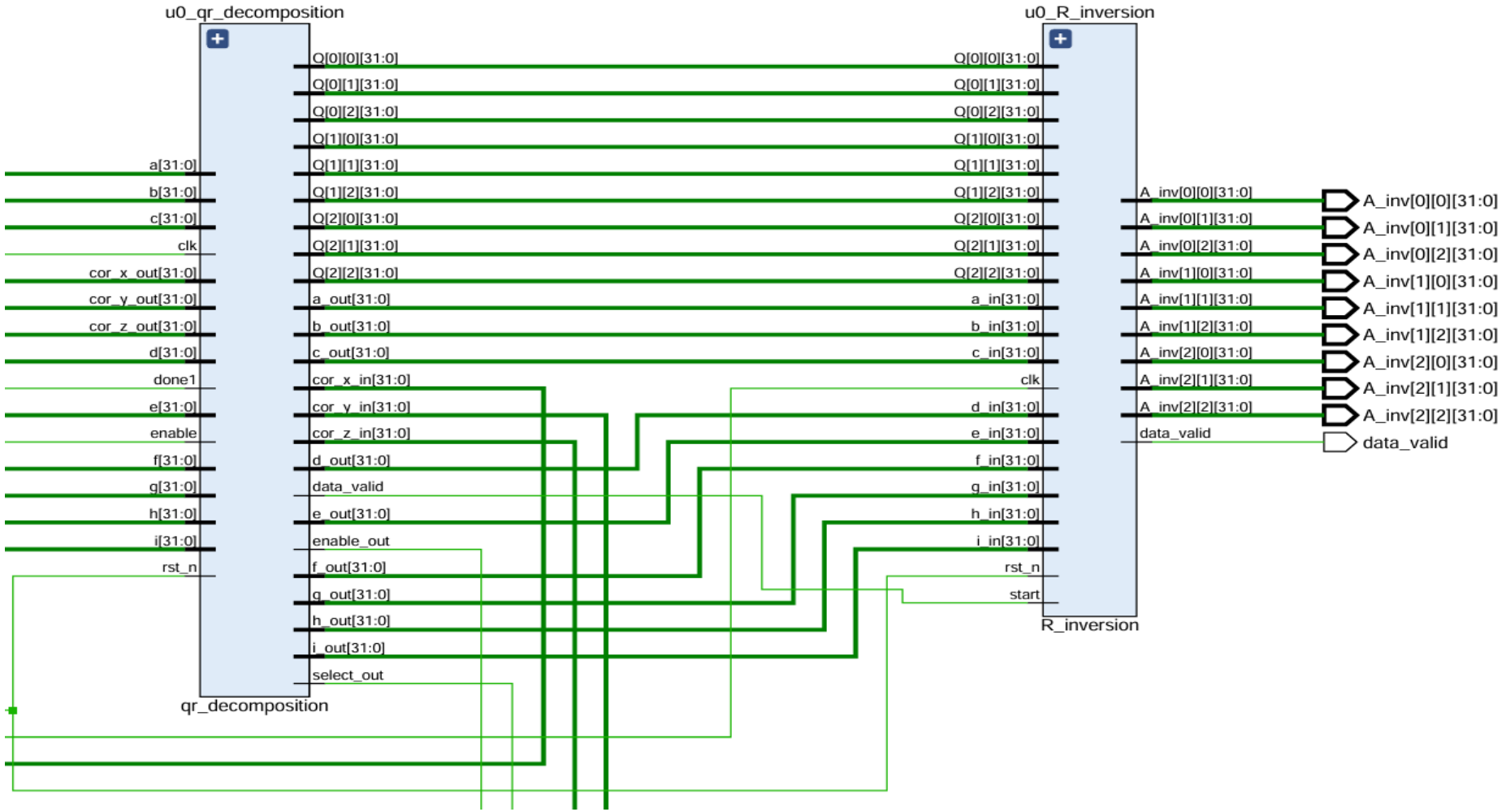
```
input Matrix

a=1.000000, b=2.000000, c=3.000000

d=2.000000, e_=1.000000, f=1.000000

g=6.000000, h=0.000000, i=1.000000

Result (Q ):

a_out_q=0.000000, b_out_q=0.000000, c_out_q=0.000000

d_out_q=0.000000, e_out_q=0.000000, f_out_q=0.000000

g_out_q=0.000000, h_out_q=0.000000, i_out_q=0.000000
```

FPGA

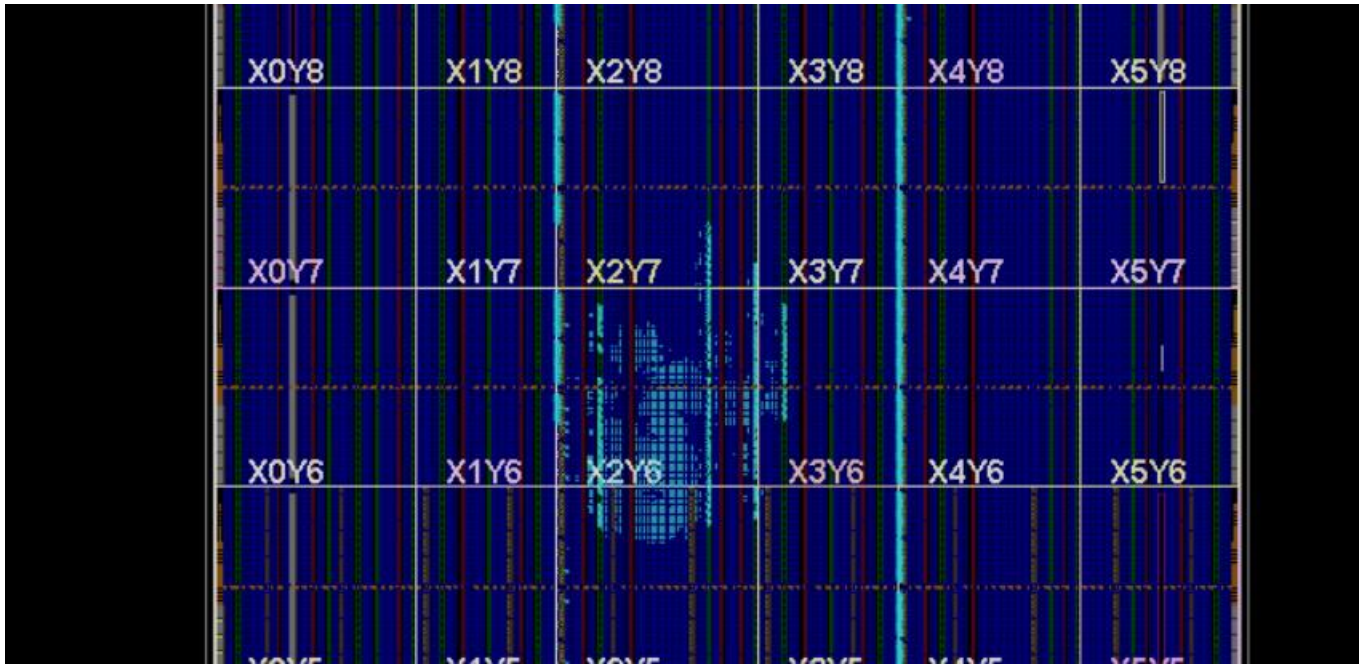


RTL diagram


a[31:0]
b[31:0]
c[31:0]
d[31:0]
u0_cordic
clk
enable
rst_n
select
x_in[31:0]
y_in[31:0]
z_in[31:0]
done
x_out[31:0]
y_out[31:0]
z_out[31:0]
cordic
e[31:0]
enable
clk_in1
rst_n
rst_n_IBUF_inst
I
O
IBUF
clk_gen
clk_in1
resetn
clk_out1
clk_wiz_0
f[31:0]
g[31:0]
h[31:0]
i[31:0]


u0_qr_decomposition
a[31:0]
b[31:0]
c[31:0]
clk
cor_x_out[31:0]
cor_y_out[31:0]
cor_z_out[31:0]
d[31:0]
done1
e[31:0]
enable
f[31:0]
g[31:0]
h[31:0]
i[31:0]
rst_n
Q[0][0][31:0]
Q[0][1][31:0]
Q[0][2][31:0]
Q[1][0][31:0]
Q[1][1][31:0]
Q[1][2][31:0]
Q[2][0][31:0]
Q[2][1][31:0]
Q[2][2][31:0]
a_out[31:0]
b_out[31:0]
c_out[31:0]
cor_x_in[31:0]
cor_y_in[31:0]
cor_z_in[31:0]
d_out[31:0]
data_valid
e_out[31:0]
enable_out
f_out[31:0]
g_out[31:0]
h_out[31:0]
i_out[31:0]
select_out
qr_decomposition
u0_R_inversion
Q[0][0][31:0]
Q[0][1][31:0]
Q[0][2][31:0]
Q[1][0][31:0]
Q[1][1][31:0]
Q[1][2][31:0]
Q[2][0][31:0]
Q[2][1][31:0]
Q[2][2][31:0]
a_in[31:0]
b_in[31:0]
c_in[31:0]
clk
d_in[31:0]
e_in[31:0]
f_in[31:0]
g_in[31:0]
h_in[31:0]
i_in[31:0]
rst_n
start
R_inversion
A_inv[0][0][31:0]
A_inv[0][1][31:0]
A_inv[0][2][31:0]
A_inv[1][0][31:0]
A_inv[1][1][31:0]
A_inv[1][2][31:0]
A_inv[2][0][31:0]
A_inv[2][1][31:0]
A_inv[2][2][31:0]
data_valid
A_inv[0][0][31:0]
A_inv[0][1][31:0]
A_inv[0][2][31:0]
A_inv[1][0][31:0]
A_inv[1][1][31:0]
A_inv[1][2][31:0]
A_inv[2][0][31:0]
A_inv[2][1][31:0]
A_inv[2][2][31:0]
data_valid

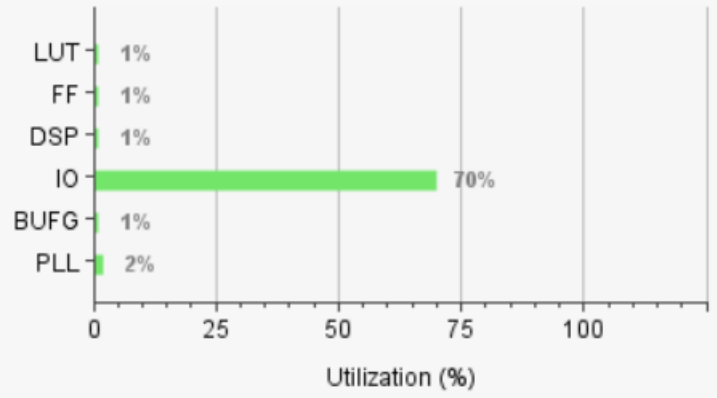
implementation snippet



Reports

Utilization Post-Synthesis | **Post-Implementation**

Graph | Table



Timing **Setup** | Hold | Pulse Width

Worst Negative Slack (WNS):	0.328 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	274

Implemented Timing Report

Power **Summary** | On-Chip

Total On-Chip Power:	**2.971 W**
Junction Temperature:	**26.6 °C**
Thermal Margin:	73.4 °C (126.1 W)
Effective ϑJA:	0.5 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy
	7496	**235**	**0.00**	**0**	**100**	**10/2/24 9:59 PM**	**00:02:46**	**Vivado Synthesis Defaults (Vivado Synthesis 2018)**
								Vivado Implementation Defaults (Vivado Implementation 2018)
0	**7495**	**235**	**0.00**	**0**	**100**	**10/2/24 10:04 PM**	**00:25:00**	**Vivado Implementation Defaults (Vivado Implementation 2018)**
						10/2/24 9:58 PM	00:01:13	Vivado Synthesis Defaults (Vivado Synthesis 2018)

Thank you