

Rajan Verma

MAS WES

Code is : <https://github.com/rverma999/wes237C/tree/main/project2>

project2/cordic/HLS/cordic_baseline/solution1_baseline/syn/report

Cordic Baseline

Csim Report (itr=16) :

```
1INFO: [SIM 2] ***** CSIM start *****
2INFO: [SIM 4] CSIM will launch GCC as the compiler.
3make: 'csim.exe' is up to date.
4---Testing results-----
5Test: x=0.8147, y=0.1269, golden theta=0.1545, golden r=0.8245, your theta=0.1545, your r=0.8245
6Test: x=0.6323, y=-0.2785, golden theta=-0.4149, golden r=0.6909, your theta=-0.4149, your r=0.6909
7Test: x=-0.5469, y=-0.9575, golden theta=-2.0898, golden r=1.1027, your theta=-2.0898, your r=1.1027
8Test: x=-0.4854, y=0.7003, golden theta=2.1769, golden r=0.8521, your theta=2.1769, your r=0.8521
9---RMS error-----
10-----
11      RMSE(R)          RMSE(Theta)
120.000000034798518 0.000016661200789
13-----
14INFO: [SIM 1] CSim done with 0 errors.
15INFO: [SIM 3] ***** CSIM finish *****
16
```

Synthesis Report :

No filter settings

Name	DSP	Pragma	Variable	Op	Impl	Latency
cordiccart2pol	12					
fmul_32ns_32ns_32_4_max_dsp_1_U18	3		mul1	fmul maxdsp		3
fmul_32ns_32ns_32_4_max_dsp_1_U18	3		mul2	fmul maxdsp		3
fadd_32ns_32ns_32_5_full_dsp_1_U16	2		add1	fadd fulldsp		4
fsqrt_32ns_32ns_32_16_no_dsp_1_U20	-		conv1	fsqrt fabric		15
cordiccart2pol_Pipeline_VITIS_LOOP_74_1	4					

No user config_op information

Bind Storage Report

No filter settings

Name	BRAM	URAM	Pragma	Variable	Storage	Impl	Latency
cordiccart2pol	-	-					
cordiccart2pol_Pipeline_VITIS_LOOP_74_1	-	-					

No user config_storage information

Console Errors Warnings Guidance Properties Man Pages Git Repositories Modules/Loops

Synthesis Cosimulation

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
cordiccart2pol				-	252	2.520E3		253	-	no	0	12	1629	2855	0
cordiccart2pol_Pipeline_VITIS_LOOP_74_1	II Violation			-	228	2.280E3		228	-	no	0	4	913	1301	0
VITIS_LOOP_74_1	II Violation	Memory Dependency 1		-	226	2.260E3		19	13	17	yes	-	-	-	-

Now Optimized Final Code :

```
4---Testing results-----
5Test: x=0.8147, y=0.1269, golden theta=0.1545, golden r=0.8245, your theta=0.1544, your r=0.8245
6Test: x=0.6323, y=-0.2785, golden theta=-0.4149, golden r=0.6909, your theta=-0.4146, your r=0.6909
7Test: x=-0.5469, y=-0.9575, golden theta=-2.0898, golden r=1.1027, your theta=-2.0899, your r=1.1027
8Test: x=-0.4854, y=0.7003, golden theta=2.1769, golden r=0.8521, your theta=2.1771, your r=0.8520
9---RMS error-----
10-----
11      RMSE(R)          RMSE(Theta)
120.000029913528124 0.000192288425751
```

Synthesis Report for Optimized Code

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
cordiccart2pol				-	67	670.000	-	1	-	yes	0	2	2470	6405	0

In Header file :

```
typedef ap_fixed<20,3> data_t;
```

```
#include "cordiccart2pol.h"
```

```
data_t Kvalues[NO_ITER] = {1, 0.500000000000000, 0.250000000000000, 0.125000000000000, 0.062500000000000,
0.031250000000000, 0.015625000000000, 0.007812500000000, 0.003906250000000,
0.001953125000000, 0.000976562500000, 0.000488281250000, 0.000244140625000,
0.000122070312500, 6.1035156250000e-05, 3.0517578125000e-05};
```

```
data_t angles[NO_ITER] = {0.785398163397448, 0.463647609000806, 0.244978663126864, 0.124354994546761,
0.0624188099959574, 0.0312398334302683, 0.0156237286204768, 0.00781234106010111,
0.00390623013196697, 0.00195312251647882, 0.000976562189559320, 0.000488281211194898,
0.000244140620149362, 0.000122070311893670, 6.10351561742088e-05, 3.05175781155261e-05};
```

```
void cordiccart2pol(data_t x, data_t y, data_t *r, data_t *theta)
```

```
//void cordiccart2pol(ap_fixed<16,2> x, ap_fixed<16,2> y, ap_fixed<16,2> *r, ap_fixed<20,3> *theta)
```

```
{
```

```
    // Write your code here
```

```
    #pragma HLS INTERFACE ap_ctrl_none port=return
```

```
    #pragma HLS INTERFACE ap_none port=x
```

```
    #pragma HLS INTERFACE ap_none port=y
```

```
    #pragma HLS INTERFACE ap_none port=r
```

```
    #pragma HLS INTERFACE ap_none port=theta
```

```
    // Pipeline the function
```

```
    #pragma HLS PIPELINE II=1
```

```
//ap_int<3> direction;
```

```
int direction;
```

```
//const data_t K = 0.607252935;
```

```
const data_t K = 0.6072788442269037;
```

```
const data_t PI = 3.14159265358979323846;
```

```
//data_t calc_K=1;
```

```

// Store current values

//data_t angle=0;

//data_t x_temp;// = x;

//data_t y_temp;// = y;

//data_t org_x = x;

//data_t org_y = y;

//data_t quadrant_angle = 0.0;

ap_fixed<16,2> x_temp;// = x;

ap_fixed<16,2> y_temp;// = y;

ap_fixed<16,2> org_x = x;

ap_fixed<16,2> org_y = y;

ap_fixed<16,2> quadrant_angle = 0.0;

ap_fixed<20,3> angle=0;

```

```

// Determine quadrant and pre-rotate if needed

```

```

if (x < 0) {

    if (y < 0) {

        // Quadrant III

        quadrant_angle = -PI;

        x = -x;

        y = -y;

    } else {

        // Quadrant II

        quadrant_angle = PI;

        x = -x;

    }

} else if (y < 0 && x >= 0) {

    // Quadrant IV

    quadrant_angle = 0;

    x=x;

```

```

        y=-y;
    }

//for(int i =0;i<=NO_ITER;i++){
//for(ap_int<5> i =0;i<=17;i++){
for(ap_uint<5> i = 0; i < NO_ITER; i++) {

    #pragma HLS UNROLL factor=4

    #pragma HLS LOOP_TRIPCOUNT min=16 max=16

    direction = (y >= 0) ? -1 : 1;

    x_temp = x;
    y_temp = y;

    // Perform rotation using K-values
    x = x_temp - direction*(y_temp*Kvalues[i]);
    y = y_temp + direction*(x_temp*Kvalues[i]);

    // Accumulate angle (radians)
    angle -= direction * angles[i];

    // Calculate final angle based on quadrant
    if (org_x < 0) {
        if (org_y < 0) {
            *theta = -PI + angle; // Quadrant III
        } else {
            *theta = PI - angle; // Quadrant II
        }
    } else if (org_y < 0) {
        *theta = -angle; // Quadrant IV
    } else {
        *theta = angle; // Quadrant I
    }

    //calc_K*= Kvalues[i];

    //printf("Step %2d: x=%0.6f, y=%0.6f, angle=%0.6f calc_k=%0.8f \n",i+1, x, y, angle,calc_K);
}

```

```

// Apply CORDIC scaling factor

//doesn't work with ap_fixed *r = sqrt(org_x*org_x + org_y*org_y);

// Convert to double before sqrt

double temp = (double)(org_x*org_x + org_y*org_y);

*r = (data_t)sqrt(temp);
}

```

- **Question 1:** One important design parameter is the number of rotations. Change that number to numbers between 10 and 20 and describe the trends. What happens to performance? Resource usage? Accuracy of the results? Why does the accuracy stop improving after some number of iterations? Can you precisely state when that occurs?

ITR=10

```

1INFO: [SIM 2] ***** CSIM start *****
2INFO: [SIM 4] CSIM will launch GCC as the compiler.
3  Compiling ../../../../cordic/cordiccart2pol.cpp in debug mode
4  Generating csim.exe
5---Testing results-----
6Test: x=0.8147, y=0.1269, golden theta=0.1545, golden r=0.8245, your theta=0.1553, your r=0.8245
7Test: x=0.6323, y=-0.2785, golden theta=-0.4149, golden r=0.6909, your theta=-0.4141, your r=0.6909
8Test: x=-0.5469, y=-0.9575, golden theta=-2.0898, golden r=1.1027, your theta=-2.0902, your r=1.1027
9Test: x=-0.4854, y=0.7003, golden theta=2.1769, golden r=0.8521, your theta=2.1770, your r=0.8521
10---RMS error-----
11-----
12  RMSE(R)          RMSE(Theta)
130.000000034798518  0.000598536396865
14-----
15INFO: [SIM 1] CSim done with 0 errors.
16INFO: [SIM 3] ***** CSIM finish *****
17

```

ltr=13

```

1INFO: [SIM 2] ***** CSIM start *****
2INFO: [SIM 4] CSIM will launch GCC as the compiler.
3  Compiling ../../../../cordic/cordiccart2pol.cpp in debug mode
4  Generating csim.exe
5---Testing results-----
6Test: x=0.8147, y=0.1269, golden theta=0.1545, golden r=0.8245, your theta=0.1544, your r=0.8245
7Test: x=0.6323, y=-0.2785, golden theta=-0.4149, golden r=0.6909, your theta=-0.4150, your r=0.6909
8Test: x=-0.5469, y=-0.9575, golden theta=-2.0898, golden r=1.1027, your theta=-2.0899, your r=1.1027
9Test: x=-0.4854, y=0.7003, golden theta=2.1769, golden r=0.8521, your theta=2.1769, your r=0.8521
10---RMS error-----
11-----
12  RMSE(R)          RMSE(Theta)
130.000000034798518  0.00081046528067
14-----
15INFO: [SIM 1] CSim done with 0 errors.
16INFO: [SIM 3] ***** CSIM finish *****
17

```

itr=16:

```

1 INFO: [SIM 2] ***** CSIM start *****
2 INFO: [SIM 4] CSIM will launch GCC as the compiler.
3 make: 'csim.exe' is up to date.
4 ---Testing results-----
5 Test: x=0.8147, y=0.1269, golden theta=0.1545, golden r=0.8245, your theta=0.1545, your r=0.8245
6 Test: x=0.6323, y=-0.2785, golden theta=-0.4149, golden r=0.6909, your theta=-0.4149, your r=0.6909
7 Test: x=-0.5469, y=-0.9575, golden theta=-2.0898, golden r=1.1027, your theta=-2.0898, your r=1.1027
8 Test: x=-0.4854, y=0.7003, golden theta=2.1769, golden r=0.8521, your theta=2.1769, your r=0.8521
9 ---RMS error-----
10 -----
11      RMSE(R)          RMSE(Theta)
12 0.000000034798518 0.000016661200789
13 -----
14 INFO: [SIM 1] CSim done with 0 errors.
15 INFO: [SIM 3] ***** CSIM finish *****
16

```

ITR=18

```

4 ---Testing results-----
5 Test: x=0.8147, y=0.1269, golden theta=0.1545, golden r=0.8245, your theta=0.1545, your r=0.8245
6 Test: x=0.6323, y=-0.2785, golden theta=-0.4149, golden r=0.6909, your theta=-0.4149, your r=0.6909
7 Test: x=-0.5469, y=-0.9575, golden theta=-2.0898, golden r=1.1027, your theta=-2.0898, your r=1.1027
8 Test: x=-0.4854, y=0.7003, golden theta=2.1769, golden r=0.8521, your theta=2.1769, your r=0.8521
9 ---RMS error-----
10 -----
11      RMSE(R)          RMSE(Theta)
12 0.000000034798518 0.000016661200789

```

I observe that the precision increases till 16 or 17 iterations and then more or less stabilizes to me to observe any significant change.

in Theory the resource usage should increase as we increase iterations/accuracy.

Contrary to expectation : Resource usage was almost same for 10, 16 and 18 iterations.

10 iterations had 1 just less flipflop!

Performance wise the latency increases. I see interval column of the synthesis report increasing.

As we increase the iteration the accuracy increase by smaller and smaller amount. so beyond a point if improvement is less than the precision point then that's an optimum point. here I think 17 iteration is is that point.

- **Question 2:** Another important design parameter is the data type of the variables. Is one data type sufficient for every variable or is it better for each variable to have a different type? Does the best data type depend on the input data? What is the best technique for the designer to determine the data type(s)?

different datatype suits for variable to optimize and so that we don't waste hardware.

we should use ap_fixed,

I wanted to customize a lot of variable but B also had to be changes for that so I refrained from doing to.

Designed can use HLS tools to let the tool decide

I would use following if I could in tb and the files.

```
ap_fixed<16,2> x_temp;// = x,y and other variable;
```

```
ap_fixed<16,2> quadrant_angle = 0.0;
```

```
ap_fixed<20,3> angle=0;
```

itr=16, data type is ap_fixed<20,3>

```

4---Testing results-----
5Test: x=0.8147, y=0.1269, golden theta=0.1545, golden r=0.8245, your theta=0.1544, your r=0.8245
6Test: x=0.6323, y=-0.2785, golden theta=-0.4149, golden r=0.6909, your theta=-0.4146, your r=0.6909
7Test: x=-0.5469, y=-0.9575, golden theta=-2.0898, golden r=1.1027, your theta=-2.0899, your r=1.1027
8Test: x=-0.4854, y=0.7003, golden theta=2.1769, golden r=0.8521, your theta=2.1771, your r=0.8520
9---RMS error-----
10-----
11 RMSE(R)          RMSE(Theta)
12 0.000029913528124 0.000192288425751

```

Synthesis Report

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
cordiccart2pol			-		67	670.000	-	1	-	yes	0	2	2470	6405	0

- **Question 3:** What is the effect of using simple operations (add and shift) in the CORDIC as opposed to multiply and divide? How does the resource usage change? Performance? Accuracy?

If we use multiplies and shifters then its pretty simpler design and not a lot of DSP. while with multiple and divide we will use more DSPs.

In terms of accuracy, in this approach accuracy depends on number of iterations while in the traditional approach it will depend upon the accuracy of libraries.

Performance:

This approach is simpler. so fixed number of simpler operations.

in Traditional Complex operations, variable latency.

- **Question 4:** These questions all refer to the lookup table (LUT) implementation of the Cartesian to Polar transformation.

- How does the input data type affect the size of the LUT? How does the output data type affect the size of the LUT? Precisely describe the relationship between input/output data types and the number of bits required for the LUT.

LUT Size = $2^{(2*W)}$, W is bit width

for W=8 , LUT size = 65,536 entries.

Every single bit addition leads to additional 4 times the LUT size.

Every LUT entry stores r and theta.

- The testbench assumes that the inputs x, y are normalized between [-1,1]. What is the minimum number of integer bits required for x and y? What is the minimal number of integer bits for the output data type R and Theta?

for inputs x,y : 2 (1 for value, 1 for sign)

For outputs : R needs 2 inputs

- Theta needs 3 inputs. {-PI to PI}

- Modify the number of fractional bits for the input and output data types. How does the precision of the input and output data types affect the accuracy (RMSE) results?

- What is the performance (throughput, latency) of the LUT implementation. How does this change as the input and output data types change?

- W =8

```
6 Testbench min_theta=-3.1316, max_theta=3.1416
7 Testbench min_r=0.0000, max_r=1.4142
8 RMSE(R) RMSE(Theta)
9 0.023094084113836 0.051045715808868
10 INFO: [SIM 1] CSim done with 0 errors.
```

W=12

```
6 Testbench min_theta=-3.1316, max_theta=3.1416
7 Testbench min_r=0.0000, max_r=1.4142
8 RMSE(R) RMSE(Theta)
9 0.023094084113836 0.051045715808868
10 INFO: [SIM 1] CSim done with 0 errors.
```

I had expected that precision will improve and eRMSE values will go down but interestingly they remain same even after changing.

- What advantages/disadvantages of the CORDIC implementation compared to the LUT-based implementation?

CORDIC vs. LUT Comparison:

CORDIC Advantages:

- Smaller hardware footprint
- No memory requirements
- More suitable for high-precision applications

LUT Advantages:

- Single-cycle latency
- Better for low-precision applications
- Simpler implementation

CORDIC Disadvantages:

- Higher latency (iterations)
- Variable timing
- More complex control logic

LUT Disadvantages:

- Exponential memory growth with precision
- Limited precision due to memory constraints
- Higher power consumption
- Not practical for high-precision

Csim Report :

```
1 INFO: [SIM 2] ***** CSIM start *****
2 INFO: [SIM 4] CSIM will launch GCC as the compiler.
3   Compiling ../../../../cordiccart2pol_test.cpp in debug mode
4   Compiling ../../../../cordiccart2pol.cpp in debug mode
5   Generating csim.exe
6 Testbench min_theta=-3.1316, max_theta=3.1416
7 Testbench min_r=0.0000, max_r=1.4142
8   RMSE(R)          RMSE(Theta)
9 0.023094084113836 0.051045715808868
10 INFO: [SIM 1] CSim done with 0 errors.
11 INFO: [SIM 3] ***** CSIM finish *****
12
```

Phase Detector

I got stuck at segmentation fault :

Code is : https://github.com/rverma999/wes237C/tree/main/project2/project2/phase_detector/HLS/phasedetector

Code :

```
void fir ( data_t I, data_t Q, data_t *X, data_t *Y ) {
```

```
    // Write your code here
```

```
    //Calculate X
```

```
    //Calculate Y
```

```
#pragma HLS PIPELINE II=1
```

```
//int i1_out, i2_out, q1_out, q2_out;
```

```
data_t i1_out, i2_out, q1_out, q2_out;
```

```
// Process through individual FIR filters
```

```

    firI1(&i1_out, I);
    firI2(&i2_out, I);
    firQ1(&q1_out, Q);
    firQ2(&q2_out, Q);

    // Complex matched filter combination
    //  $X = I1 - Q2$ 
    //  $Y = I2 + Q1$ 
    *X = (data_t)(i1_out - q2_out);
    *Y = (data_t)(i2_out + q1_out);
}

void phasedetector (
    data_t *I,
    data_t *Q,

    data_t *R,
    data_t *Theta,

    int length
){

    // Buffers for matched filter outputs
    data_t X_out, Y_out;

    // Write your code here

    // Buffers for FIR filter outputs
    //data_t I_matched_real, I_matched_imag;
    //data_t Q_matched_real, Q_matched_imag;

    // Add null pointer checks
    if (!I || !Q || !R || !Theta || length <= 0) {
        return;
    }

```

```

// Process each sample
for (int i = 0; i < length; i++) {
    #pragma HLS PIPELINE II=1

    // Complex Matched Filter Implementation
    // Process I channel
    //fir(&I_matched_real, I[i]); // Real part of I channel
    //fir(&I_matched_imag, I[i]); // Imaginary part of I channel
    //
    /// Process Q channel
    //fir(&Q_matched_real, Q[i]); // Real part of Q channel
    //fir(&Q_matched_imag, Q[i]); // Imaginary part of Q channel
    //
    /// Combine matched filter outputs
    /// Complex multiplication result
    //data_t real_part = I_matched_real - Q_matched_imag;
    //data_t imag_part = I_matched_imag + Q_matched_real;

    // Apply complex matched filter
    // fir function takes I and Q inputs and produces X and Y outputs
    fir(I[i], Q[i], &X_out, &Y_out);

    // Convert to polar coordinates using CORDIC
    cordiccart2pol(X_out, Y_out, &R[i], &Theta[i]);
}
}

```

```

INFO: [HLS 200-1510] Running: csim design -quiet
Running Dispatch Server on port: 45597
INFO: [SIM 211-2] ***** CSIM start *****
INFO: [SIM 211-4] CSIM will launch GCC as the compiler.
Compiling ../../../../cordiccart2pol.cpp in debug mode
Generating csim.exe
@E Simulation failed: SIGSEGV.
ERROR: [SIM 211-100] CSim failed with errors.
INFO: [SIM 211-3] ***** CSIM finish *****
INFO: [HLS 200-111] Finished Command csim_design CPU user time: 0.55 seconds. CPU system time: 0.17 seconds. Elapsed time: 0.76 seconds
4
while executing
"source /home/wes_2025_r_verma/project2/phase_detector/HLS/phasedetector/PD_hls/solution1/csim.tcl"
invoked from within
"hls::main /home/wes_2025_r_verma/project2/phase_detector/HLS/phasedetector/PD_hls/solution1/csim.tcl"
("uplevel" body line 1)
invoked from within
"uplevel 1 hls::main {*} $newargs"
(procedure "hls_proc" line 16)
invoked from within
"hls_proc [info nameofexecutable] $argv"
INFO: [HLS 200-112] Total CPU user time: 1.23 seconds. Total CPU system time: 0.35 seconds. Total elapsed time: 11.52 seconds; peak all
Finished C simulation.

```

I am just presenting what I did in phase detector lab :
 mplementation of a Phase Detector system using High-Level Synthesis (HLS). The system processes complex input signals (I/Q) through matched filters and converts the results to polar coordinates (magnitude and phase).

2. System Architecture

- The implementation consists of three main components:
 - Phase Detector Top Level
 - Complex Matched Filter Bank
 - Cartesian to Polar Converter (CORDIC)
- 3. Implementation Details
 - 3.1 Phase Detector
 - Input: I/Q samples (complex data)
 - Output: R (magnitude) and Theta (phase)
 - Processing: Sequential sample processing with pipelining
 - Throughput: One sample per clock cycle
 - 3.2 Matched Filter Implementation
 - Structure: Four parallel FIR filters
 - Configuration:
 - Two filters for I-channel (fir11, fir12)
 - Two filters for Q-channel (firQ1, firQ2)
 - Filter Length: 32 taps
 - Output Combination:
 - $X = I1 - Q2$
 - $Y = I2 + Q1$

- 3.3 FIR Filter Design
- Architecture: Shift register-based implementation
- Features:
 - Static shift registers for sample history
 - Fixed coefficient arrays
 - Multiply-accumulate (MAC) operations
- Optimizations:
 - Complete array partitioning
 - Pipeline implementation
 - Unrolled MAC operations

Advantages

- High throughput
- Deterministic timing
- Efficient resource utilization
- Parallel processing capability

The implemented Phase Detector system provides an efficient hardware solution for complex signal processing, optimized for FPGA deployment. The design achieves high throughput through pipelining and parallel processing while maintaining resource efficiency through careful optimization.