WES 237C - Hardware for Embedded Systems, Kastner Project 2, CORDIC and Phase Detector	Ricardo Lizárraga rilizarraga@ucsd.edu PID: A69028483	*Submit to:* *Ryan Charles Kastner* *(TA) Zhenghua Ma* *(TA) Abarajithan Gnaneswaran*

CORDIC
https://pp4fpgas.readthedocs.io/en/latest/project2.html Links to an external site. Answer questions. Complete the demo Phase Detector
https://pp4fpgas.readthedocs.io/en/latest/project2.5.html Links to an external site. Your HLS core should pass C simulation and C synthesis Answer question
You DO NOT need to do the Phase Detector Demo since it involves DMA that we have not discussed yet
+10 points Extra Credit to those that implement the Phase Detector Demo

https://github.com/KastnerRG/Read_the_docs/tree/master/project_files/project2

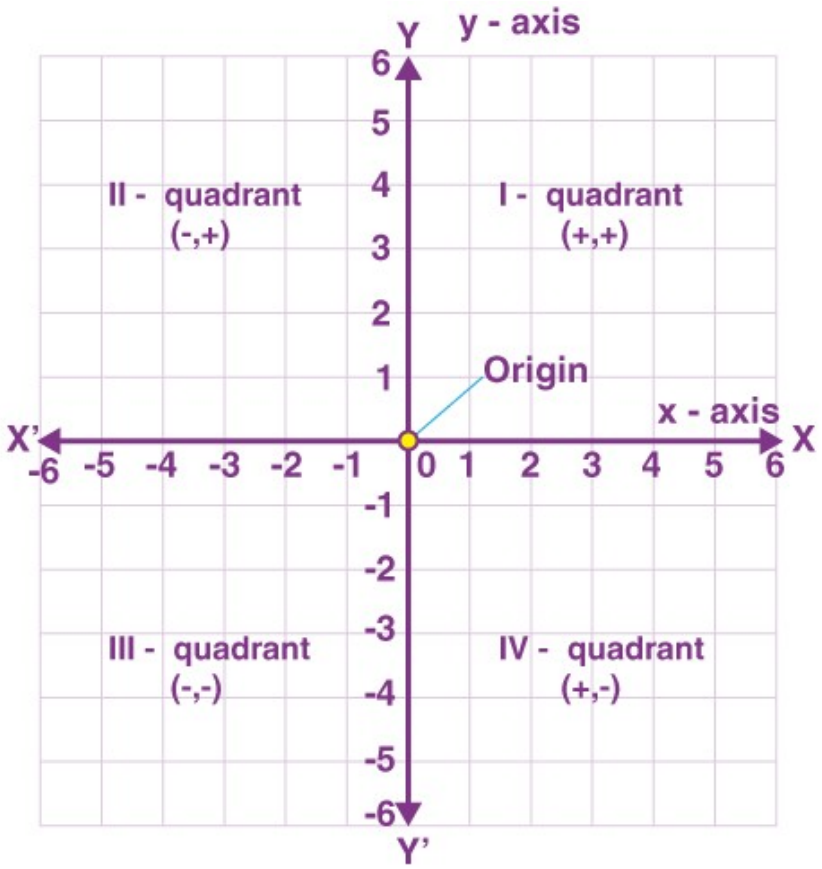
```
wes_2025_r_lizarraga@waiter:~/project2/cordic$ vitis_hls -f script.tcl
wes_2025_r_lizarraga@waiter:~/project2$ cd cordic
wes_2025_r_lizarraga@waiter:~/project2/cordic$ ls
cordiccart2pol.cpp  cordiccart2pol.h  cordiccart2pol_test.cpp  script.tcl
wes_2025_r_lizarraga@waiter:~/project2/wes_2025_r_lizarraga@waiterwes_2025_r_lizarraga@waiter:~/prowes_2025_r_lizarraga@waiter:~/projectwewes_
a@waiter:~/projwes_2025_r_lizarraga@waiter:~/projecwes_2025_r_lizarraga@waiter:wes_2025_r_lizarrawes_2025_r_lizarraga@wwes_2025_r_lizarragawes_
2025_r_lizarraga@waiter:wes_2025_r_lizarwes_2025wes_wewes_2025_r_lizarraga@waiter:~/project2/cordic$ vitis_hls -f script.tcl

****** Vitis HLS - High-Level Synthesis from C, C++ and OpenCL v2022.2 (64-bit)
  **** SW Build 3670227 on Oct 13 2022
  **** IP Build 3669848 on Fri Oct 14 08:30:02 MDT 2022
    ** Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.

source /tools/Xilinx/Vitis_HLS/2022.2/scripts/vitis_hls/hls.tcl -notrace
INFO: [HLS 200-10] Running '/tools/Xilinx/Vitis_HLS/2022.2/bin/unwrapped/lnx64.o/vitis_hls'
INFO: [HLS 200-10] For user 'wes_2025_r_lizarraga' on host 'waiter' (Linux_x86_64 version 5.15.0-122-generic) on Thu Oct 24 10:39:21 PDT 2024
INFO: [HLS 200-10] On os Ubuntu 22.04.5 LTS
INFO: [HLS 200-10] In directory '/home/wes_2025_r_lizarraga/project2/cordic'
Sourcing Tcl script 'script.tcl'
INFO: [HLS 200-1510] Running: open_project cordiccart2pol
INFO: [HLS 200-10] Creating and opening project '/home/wes_2025_r_lizarraga/project2/cordic/cordiccart2pol'.
INFO: [HLS 200-1510] Running: set_top cordiccart2pol
INFO: [HLS 200-1510] Running: add_files cordiccart2pol.cpp
INFO: [HLS 200-10] Adding design file 'cordiccart2pol.cpp' to the project
INFO: [HLS 200-1510] Running: add_files cordiccart2pol.h
INFO: [HLS 200-10] Adding design file 'cordiccart2pol.h' to the project
INFO: [HLS 200-1510] Running: add_files -tb cordiccart2pol_test.cpp
INFO: [HLS 200-10] Adding test bench file 'cordiccart2pol_test.cpp' to the project
INFO: [HLS 200-1510] Running: open_solution solution1
INFO: [HLS 200-10] Creating and opening solution '/home/wes_2025_r_lizarraga/project2/cordic/cordiccart2pol/solution1'.
INFO: [HLS 200-1505] Using default flow_target 'vivado'
Resolution: For help on HLS 200-1505 see www.xilinx.com/cgi-bin/docs/rdoc?v=2022.2;t=hls+guidance;d=200-1505.html
INFO: [HLS 200-435] Setting 'open_solution -flow_target vivado' configuration: config_interface -m_axi_latency=0
INFO: [HLS 200-1510] Running: set_part xc7z020clg400-1
INFO: [HLS 200-1611] Setting target device to 'xc7z020-clg400-1'
INFO: [HLS 200-1510] Running: create_clock -period 10
INFO: [SYN 201-201] Setting up clock 'default' with a period of 10ns.
```

3) Tasks

1. Design and verify a functionally complete CORDIC IP core using HLS: **cordic_baseline**. You are provided a testbench that you can use though that the testbench does not cover all cases. You are encouraged to create a more extensive testbench to ensure that your code is correct.

pol/source/cordiccart2pol.cpp

```cpp
#include "cordiccart2pol.h"
#include "math.h"

data_t Kvalues[NO_ITER] = {1,   0.500000000000000,  0.250000000000000,  0.125000000000000
data_t angles[NO_ITER] = {0.785398163397448,    0.463647609000806,  0.244978663126864,

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

    data_t cart_x = x;
    data_t cart_y = y;
    data_t theta_now = 0;   //Initialized in Quadrant I, zero degrees
    // Quadrants:   I   (+,+), II (-,+)
    //              III (+,+), IV (+,-)
    if(cart_x<0){       //If x resides in the left side  quadrants(II & III)?
        if(cart_y>0){   // If (y) resides in quadrant II
            cart_y = -x;
            cart_x = +y;
            theta_now = 1.5707963267945; //Then theta_now is in 90º= + π/2

        }else{          //Else (y) resides in quadrant III ?
            cart_y = +x;
            cart_x = -y;
            theta_now = -1.5707963267945; //Then theta_now is in 270º= - π/2
        }
    }

    data_t x_shifting, y_shifting, sigma;
    coef_t i;
    for(i = 0;i<NO_ITER;i++){
        sigma = (cart_y > 0)? -1:1;
        x_shifting = cart_x * sigma * Kvalues[i];
        y_shifting = cart_y * sigma * Kvalues[i];

        cart_x = cart_x - y_shifting;
        cart_y = cart_y + x_shifting;

        theta_now += -sigma*angles[i];
    }
    data_t gain = 1.64676025786545;
    *r = cart_x/ gain;
    *theta = theta_now;

}
```



Cordic

```
INFO: [SIM 2] *************** CSIM start **************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../cordiccart2pol.cpp in debug mode
   Generating csim.exe
---Testing results----------------------------------
Test: x=0.8147, y=0.1269, golden theta=0.1545, golden r=0.8245, your theta=0.1545, your r=0.8245
Test: x=0.6323, y=-0.2785, golden theta=-0.4149, golden r=0.6909, your theta=-0.4149, your r=0.6909
Test: x=-0.5469, y=-0.9575, golden theta=-2.0898, golden r=1.1027, your theta=-2.0898, your r=1.1027
Test: x=-0.4854, y=0.7003, golden theta=2.1769, golden r=0.8521, your theta=2.1769, your r=0.8521
---RMS error----------------------------------
-----------------------------------------------
   RMSE(R)           RMSE(Theta)
0.000000108794012 0.000016664818759
-----------------------------------------------
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish **************
```

Synthesis Summary Report of 'cordiccart2pol'

General Information

Date: Sat Oct 26 08:26:45 2024
Version: 2022.2 (Build 3670227 on Oct 13 2022)
Project: cordiccart2pol

Solution: solution1 (Vivado IP Flow Target)
Product family: zynq
Target device: xc7z020-clg400-1

Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	7.848 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
cordiccart2pol				-0.55	262	2.620E3	-	263	-	no	0	5	932	1578	0
cordiccart2pol_Pipeline_VITIS_LOOP_31_1	II Violation			-0.55	243	2.430E3	-	243	-	no	0	5	824	1103	0

Performance Pragma

Modules & Loops	Target TI(cycles)	TI(cycles)	TI met
cordiccart2pol	-	-	-
cordiccart2pol_Pipeline_VITIS_LOOP_31_1	-	-	-

HW Interfaces

Other Ports

Interface	Mode	Bitwidth
r	ap_vld	32
theta	ap_vld	32
x	ap_none	32
y	ap_none	32

TOP LEVEL CONTROL

Interface	Type	Ports
ap_clk	clock	ap_clk
ap_rst	reset	ap_rst
ap_ctrl	ap_ctrl_hs	ap_done ap_idle ap_ready ap_start

SW I/O Information

Top Function Arguments

Argument	Direction	Datatype
x	in	float
y	in	float
r	out	float*
theta	out	float*

SW-to-HW Mapping

Argument	HW Interface	HW Type
x	x	port
y	y	port
r	r	port
r	r_ap_vld	port
theta	theta	port
theta	theta_ap_vld	port

Bind Op Report

No filter settings

Name	DSP	Pragma	Variable	Op	Impl	Latency
cordiccart2pol	5					
fdiv_32ns_32ns_32_16_no_dsp_1_U13	-		div	fdiv	fabric	15
cordiccart2pol_Pipeline_VITIS_LOOP_31_1	5					

No user config_op information

Bind Storage Report

No filter settings

Name	BRAM	URAM	Pragma	Variable	Storage	Impl	Latency
cordiccart2pol	-	-					
cordiccart2pol_Pipeline_VITIS_LOOP_31_1	-	-					

2. The ultimate goal is to create an efficient CORDIC that only uses simple operations, i.e., add and shift. You should not be using divide, multiply, etc. in your CORDIC core. First design a baseline using float variables. Once you have a functionally correct CORDIC, then start experimenting with arbitrary precision data types. This should decrease the usage of DSP and change the type of DSP utilized (you should verify that this is indeed happening). Do not change the function header/definition/interface (the input and output variables and their datatypes), only change variables inside the body of your *cordiccart2pol* function. You may use new typedefs if you wish. Try to perform as few operations with float as possible, meaning there will likely be a lot of typecasting involved.
3. A major design tradeoff for the CORDIC revolves around the precision or accuracy of the results. For example, changing the number of rotations effects the accuracy, performance, and resource usage. Another important tradeoff is the data type of the variables. Using large, complex data types (like floating point) is typically most accurate, but not good with respect to performance and resource usage. Fixed-point types provide better performance and resource usage, but may reduce the accuracy of the results. Perform design-space exploration to create a wide range of implementations using various data types for different variables, modifying the number of rotations, and performing other optimizations to find the Pareto optimal designs.

Now use of Fixed-point types to reduce the DSP resource usage:

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../cordiccart2pol.cpp in debug mode
   Generating csim.exe
---Testing results----------------------------------
Test: x=0.8147, y=0.1269, golden theta=0.1545, golden r=0.8245, your theta=0.1514, your r=0.8271
Test: x=0.6323, y=-0.2785, golden theta=-0.4149, golden r=0.6909, your theta=-0.4209, your r=0.6943
Test: x=-0.5469, y=-0.9575, golden theta=-2.0898, golden r=1.1027, your theta=1.9062, your r=1.1055
Test: x=-0.4854, y=0.7003, golden theta=2.1769, golden r=0.8521, your theta=-1.8252, your r=0.8555
---RMS error-----------------------------------
-----------------------------------------------
   RMSE(R)          RMSE(Theta)
0.003076650667936 2.827756166458130
-----------------------------------------------
ERR: [SIM 100] 'csim_design' failed: nonzero return value.
INFO: [SIM 3] *************** CSIM finish ***************
```

```c
#include "cordiccart2pol.h"
#include "ap_int.h"
#include "stdio.h"
#include "math.h"

// Define a fixed-point data type with 12 bits total, 2 fractional bits
typedef ap_fixed<12,2> FIXED_POINT;

FIXED_POINT Kvalues[NO_ITER] = {1,  0.500000000000000,  0.250000000000000,  0.1
FIXED_POINT angles[NO_ITER] = {0.785398163397448,   0.463647609000806,  0.24497


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

    FIXED_POINT cart_x = x;
    FIXED_POINT cart_y = y;
    FIXED_POINT theta_now = 0;  //Initialized in Quadrant I, zero degrees
    // Quadrants:   I   (+,+), II (-,+)
    //              III (+,+), IV (+,-)
    if(cart_x<0){       //If x resides in the left side  quadrants(II & III)?
        if(cart_y>0){   // If (y) resides in quadrant II
            cart_y = -x;
            cart_x = +y;
            theta_now = 1.5707963267945; //Then theta_now is in 90º= + π/2

        }else{          //Else (y) resides in quadrant III ?
            cart_y = +x;
            cart_x = -y;
            theta_now = -1.5707963267945; //Then theta_now is in 270º= - π/2
        }
    }

    FIXED_POINT x_shifting, y_shifting, sigma;
    coef_t i;
    for(i = 0;i<NO_ITER;i++){
        sigma = (cart_y > 0)? -1:1;
        x_shifting = cart_x * sigma * Kvalues[i];
        y_shifting = cart_y * sigma * Kvalues[i];

        cart_x = cart_x - y_shifting;
        cart_y = cart_y + x_shifting;

        theta_now += -sigma*angles[i];
    }
    FIXED_POINT gain = 1.64676025786545;
    *r = cart_x/ gain;
    *theta = theta_now;

}
```

Baseline (typedef float data_t):

Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT
262	2.620E3	-	263	-	no	0	5	932	1578
243	2.430E3	-	243	-	no	0	5	824	1103

Optimized test 1 (typedef int data_t): changing to INTEGER, doesn't synthesize, raises error

Optimized test 2 (typedef ap_fixed<8,0> data_t): {cordic_optimized1}

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../cordiccart2pol_test.cpp in debug mode
   Compiling ../../../../cordiccart2pol.cpp in debug mode
   Generating csim.exe
---Testing results----------------------------------
Test: x=-0.1875, y=0.1250, golden theta=0.1545, golden r=0.8245, your theta=-0.4297, your r=-0.3516
Test: x=-0.3711, y=-0.2812, golden theta=-0.4149, golden r=0.6909, your theta=0.4258, your r=0.2109
Test: x=0.4492, y=0.0391, golden theta=-2.0898, golden r=1.1027, your theta=0.0000, your r=-0.2617
Test: x=-0.4883, y=-0.3008, golden theta=2.1769, golden r=0.8521, your theta=0.4258, your r=0.1562
---RMS error-----------------------------------
-----------------------------------------------
   RMSE(R)           RMSE(Theta)
0.994902014732361 1.456150293350220
-----------------------------------------------
ERR: [SIM 100] 'csim_design' failed: nonzero return value.
INFO: [SIM 3] *************** CSIM finish ***************
```

Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT
4	40.000	-	5	-	no	0	1	22	161

Optimized test 3 (typedef ap_fixed<16,8> data_t): {DSP reduced from 5 TO 4), here I got error :
ERR: [SIM 100] 'csim_design' failed: nonzero return value.{ I consider this configuration not valid}

Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT
69	690.000	-	70	-	no	0	4	232	729
66	660.000	-	66	-	no	0	2	119	403

Optimized test 4(typedef ap_fixed<12,4> data_t): {DSP reduced from 5 TO 3)

Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT
69	690.000	-	70	-	no	0	3	192	601
66	660.000	-	66	-	no	0	2	99	360

Optimized test 5(typedef ap_fixed<32,8> data_t;

```
---Testing results-----------------------------------
Test: x=0.8147, y=0.1269, golden theta=0.1545, golden r=0.8245, your theta=0.1545, your r=0.8245
Test: x=0.6323, y=-0.2785, golden theta=-0.4149, golden r=0.6909, your theta=-0.4149, your r=0.6909
Test: x=-0.5469, y=-0.9575, golden theta=-2.0898, golden r=1.1027, your theta=-2.0898, your r=1.1027
Test: x=-0.4854, y=0.7003, golden theta=2.1769, golden r=0.8521, your theta=2.1769, your r=0.8521
---RMS error-----------------------------------
-----------------------------------------------
   RMSE(R)           RMSE(Theta)
0.000000335274706 0.000016652495106
-----------------------------------------------
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
```

Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
122	1.220E3	-	123	-	no	0	14	1736	1773	0
114	1.140E3	-	114	-	no	0	6	1042	1042	0

Optimized test 6 (typedef ap_fixed<64,8> data_t;

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../cordiccart2pol_test.cpp in debug mode
   Compiling ../../../../cordiccart2pol.cpp in debug mode
   Generating csim.exe
---Testing results-----------------------------------
Test: x=0.8147, y=0.1269, golden theta=0.1545, golden r=0.8245, your theta=0.1545, your r=0.8245
Test: x=0.6323, y=-0.2785, golden theta=-0.4149, golden r=0.6909, your theta=-0.4149, your r=0.6909
Test: x=-0.5469, y=-0.9575, golden theta=-2.0898, golden r=1.1027, your theta=-2.0898, your r=1.1027
Test: x=-0.4854, y=0.7003, golden theta=2.1769, golden r=0.8521, your theta=2.1769, your r=0.8521
---RMS error-----------------------------------
-----------------------------------------------
   RMSE(R)           RMSE(Theta)
0.000000033684572 0.000016587686332
-----------------------------------------------
```

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

Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT
203	2.030E3	-	204	-	no	0	81	5018	3283
194	1.940E3	-	194	-	no	0	56	3435	2090

Optimized test 7 (typedef ap_fixed<26,8> data_t;

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
make: 'csim.exe' is up to date.
---Testing results----------------------------------
Test: x=0.8147, y=0.1269, golden theta=0.1545, golden r=0.8245, your theta=0.1545, your r=0.8245
Test: x=0.6323, y=-0.2785, golden theta=-0.4149, golden r=0.6909, your theta=-0.4149, your r=0.6909
Test: x=-0.5469, y=-0.9575, golden theta=-2.0898, golden r=1.1027, your theta=-2.0898, your r=1.1027
Test: x=-0.4854, y=0.7003, golden theta=2.1769, golden r=0.8521, your theta=2.1769, your r=0.8521
---RMS error----------------------------------
-----------------------------------------------
   RMSE(R)           RMSE(Theta)
0.000018855182134 0.000039036549424
-----------------------------------------------
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT
122	1.220E3	-	123	-	no	0	8	1559	1433
114	1.140E3	-	114	-	no	0	4	1019	891

Optimized test 8 (typedef ap_fixed<26,6> data_t;

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
make: 'csim.exe' is up to date.
---Testing results----------------------------------
Test: x=0.8147, y=0.1269, golden theta=0.1545, golden r=0.8245, your theta=0.1545, your r=0.8245
Test: x=0.6323, y=-0.2785, golden theta=-0.4149, golden r=0.6909, your theta=-0.4149, your r=0.6909
Test: x=-0.5469, y=-0.9575, golden theta=-2.0898, golden r=1.1027, your theta=-2.0898, your r=1.1027
Test: x=-0.4854, y=0.7003, golden theta=2.1769, golden r=0.8521, your theta=2.1769, your r=0.8521
---RMS error----------------------------------
-----------------------------------------------
   RMSE(R)           RMSE(Theta)
0.000004682628060 0.000017950256733
-----------------------------------------------
```

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

Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT
107	1.070E3	-	108	-	no	0	8	1603	1356
99	990.000	-	99	-	no	0	4	1045	802

Optimized test 9 (typedef ap_fixed<24,6> data_t;

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../cordiccart2pol_test.cpp in debug mode
   Compiling ../../../../cordiccart2pol.cpp in debug mode
   Generating csim.exe
---Testing results----------------------------------
Test: x=0.8147, y=0.1269, golden theta=0.1545, golden r=0.8245, your theta=0.1545, your r=0.8245
Test: x=0.6323, y=-0.2785, golden theta=-0.4149, golden r=0.6909, your theta=-0.4149, your r=0.6909
Test: x=-0.5469, y=-0.9575, golden theta=-2.0898, golden r=1.1027, your theta=-2.0898, your r=1.1027
Test: x=-0.4854, y=0.7003, golden theta=2.1769, golden r=0.8521, your theta=2.1769, your r=0.8521
---RMS error----------------------------------
-----------------------------------------------
   RMSE(R)           RMSE(Theta)
0.000018855182134 0.000039036549424
-----------------------------------------------
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT
89	890.000	-	90	-	no	0	8	1330	1460
82	820.000	-	82	-	no	0	4	779	817

Optimized test 10 (typedef ap_fixed<24,5> data_t;

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../cordiccart2pol_test.cpp in debug mode
   Compiling ../../../../cordiccart2pol.cpp in debug mode
   Generating csim.exe
---Testing results----------------------------------
Test: x=0.8147, y=0.1269, golden theta=0.1545, golden r=0.8245, your theta=0.1545, your r=0.8245
Test: x=0.6323, y=-0.2785, golden theta=-0.4149, golden r=0.6909, your theta=-0.4149, your r=0.6909
Test: x=-0.5469, y=-0.9575, golden theta=-2.0898, golden r=1.1027, your theta=-2.0898, your r=1.1027
```

```
Test: x=-0.4854, y=0.7003, golden theta=2.1769, golden r=0.8521, your theta=2.1769, your r=0.8521
---RMS error-----------------------------------
-----------------------------------------------
   RMSE(R)           RMSE(Theta)
0.000010583441508 0.000018502911189
-----------------------------------------------
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT
121	1.210E3	-	122	-	no	0	8	1446	1474
114	1.140E3	-	114	-	no	0	4	946	958

Optimized test 11 (typedef ap_fixed<18,5> data_t;

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../cordiccart2pol_test.cpp in debug mode
   Compiling ../../../../cordiccart2pol.cpp in debug mode
   Generating csim.exe
---Testing results----------------------------------
Test: x=0.8147, y=0.1268, golden theta=0.1545, golden r=0.8245, your theta=0.1536, your r=0.8252
Test: x=0.6322, y=-0.2786, golden theta=-0.4149, golden r=0.6909, your theta=-0.4158, your r=0.6914
Test: x=-0.5470, y=-0.9575, golden theta=-2.0898, golden r=1.1027, your theta=-2.0896, your r=1.1034
Test: x=-0.4855, y=0.7002, golden theta=2.1769, golden r=0.8521, your theta=2.1766, your r=0.8527
---RMS error-----------------------------------
-----------------------------------------------
   RMSE(R)           RMSE(Theta)
0.000620615843218 0.000668342516292
-----------------------------------------------
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT
55	550.000	-	56	-	no	0	7	862	969
51	510.000	-	51	-	no	0	4	554	586

Optimized test 12 (typedef ap_fixed<16,3> data_t; {cordic_optimized2}

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../cordiccart2pol_test.cpp in debug mode
```

```
   Compiling ../../../../cordiccart2pol.cpp in debug mode
   Generating csim.exe
---Testing results-----------------------------------
Test: x=0.8147, y=0.1268, golden theta=0.1545, golden r=0.8245, your theta=0.1536, your r=0.8252
Test: x=0.6322, y=-0.2786, golden theta=-0.4149, golden r=0.6909, your theta=-0.4158, your r=0.6914
Test: x=-0.5470, y=-0.9575, golden theta=-2.0898, golden r=1.1027, your theta=-2.0896, your r=1.1034
Test: x=-0.4855, y=0.7002, golden theta=2.1769, golden r=0.8521, your theta=2.1766, your r=0.8527
---RMS error-----------------------------------
-----------------------------------------------
   RMSE(R)           RMSE(Theta)
0.000620615843218 0.000668342516292
-----------------------------------------------
INFO: [SIM 1] CSim done with 0 errors.
```

INFO: [SIM 3] ********** CSIM finish **********

Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT
55	550.000	-	56	-	no	0	7	711	924
51	510.000	-	51	-	no	0	4	441	567

Optimization summary

CORDIC code version	DSP	FF	LUT	OBSERVATIONS
typedef float data_t	5	932	1578	Original version
typedef ap_fixed<8,0> data_t	1	22	161	Optimized 1: Actually not workable solution, since I get error "`ERR: [SIM 100] 'csim_design' failed: nonzero return value`"
typedef ap_fixed<16,3>	7	711	924	Optimized 2 Floating point solution has less DSPs, but this one has less LUT, really depends on what resources we want to optimize. This solution passes all test.

4. Explore the architectural tradeoffs of a lookup table (LUT) architecture. We have provided a fully functional base implementation of this. You should read and understand this code. You should analyze the design-space of this lookup table IP core by changing the parameters of the look-up tables, e.g., by varying the data type of the input data and the output data types. This should allow you to find architectures with different resource usage, performance, and accuracy results.

Cordic_LUT

Test/ Optimization #	Changes on: • **Data type of a value in the LUT** • **Total size of fixed-point representation** • **Number of integer bits of fixed-point representation**	Errors impact: • RMSE(**R**) • RMSE(**Theta**)	**Resources**	**Observations**

1	`typedef ap_fixed<8,3> data_t;`
	`#define W 8`
	`#define I 2`
		0.023094084113836
		0.051045715808868
			BRAM DSP FF LUT
			8 0 3 102
				This is the baseline solution

2	`typedef ap_fixed<6,3> data_t;`
	`#define W 7`
	`#define I 2`
		0.086298309266567
		0.170881420373917
			BRAM DSP FF LUT
			2 0 3 78
				Here the both RMSE errors still very small and we have reduced BRAM from 8 to 2, Reduced LUT from 102 TO 78

3	`typedef ap_fixed<6,2> data_t;`
	`#define W 6`
	`#define I 2`
		0.045045133680105
		2.514456033706665
			BRAM DSP FF LUT
			2 0 3 48
				RMSE (**R)** is very acceptable, however RMSE (**Theta**) jumped very high (2.51), and of course resources are less

4	`typedef ap_fixed<4,1> data_t;`
	`#define W 4`
	`#define I 2`
		1.107706665992737
		1.726269364356995
			BRAM DSP FF LUT
			0 0 5 128
				Both RMSE errors are relatively high, in this case BRAM went down to 0.

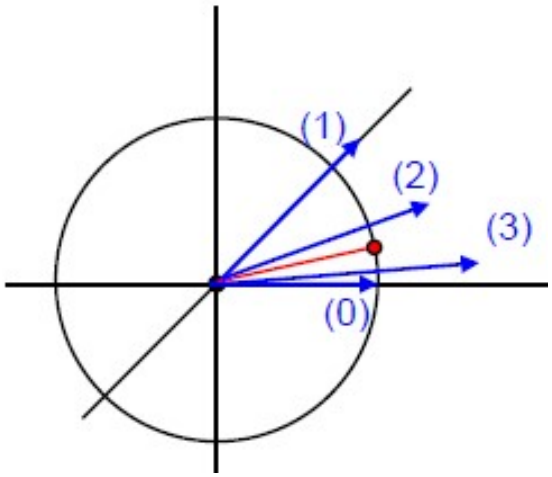
5	`typedef ap_fixed<1,0> data_t;`
	`#define W 1`
	`#define I 1`
		0.820569872856140
		1.804754376411438
			BRAM DSP FF LUT
			0 0 0 0
				Here minimum resource usage, but RMSE errors are significantly high

5	`typedef ap_fixed<4,3> data_t;`
	`#define W 5`
	`#define I 2`
		0.289403885602951
		0.595246791839600
			BRAM DSP FF LUT
			2 0 3 74
				THIS COULD BE THE BEST OPTIMUS I FOUND,
				LUT tables reduced from 102 TO 74
				BRAM from 8 to 2

5) Report

Your report should answer the following questions. Make it very clear where you are answering each of these questions (e.g., make each question a header or separate section or copy/paste the questions in your report and add your answer or simply put a bold or emphasized **Question X** before your answer). Your report will be graded based on your responses.

- **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?

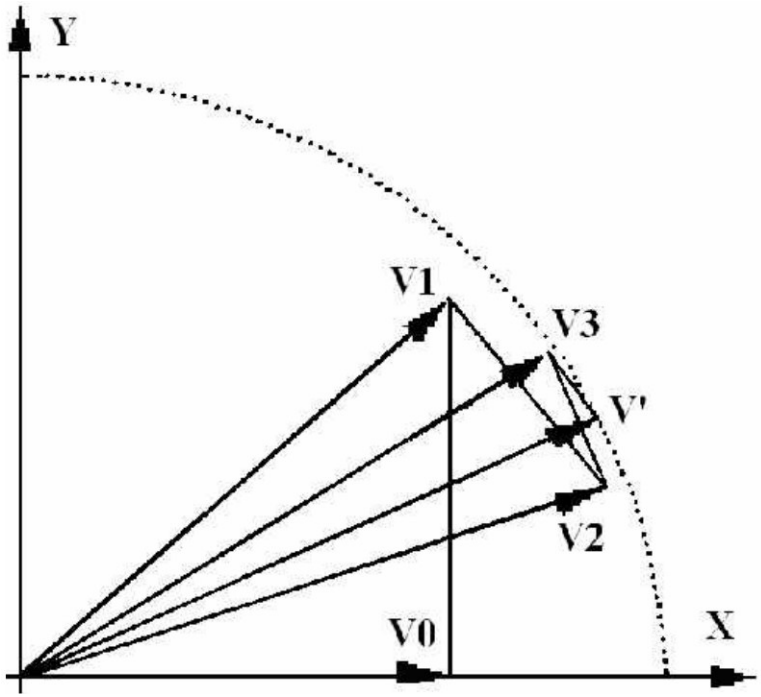


The more iterations (more rotations), we have better accuracy, but this comes with a cost of performance, because the more iteration, it will cost more computing processing, so the tradeoff between accuracy of the application versus computing performance resources.

See figure with "Iterative vector rotation, initialized with V_0"

After a number iterations, the accuracy will stop improving because, every new iteration, comes with a new smaller ROTATION jump (see below table "Rotation angle", see after Iteration 6, Rotation angle is too small 0.015625), so after a few ROTATION more the step will be too small and not necessary to go that far, so it is important to determine a reasonable number of iterations based on the application accuracy required.

i	2^{-i}	Rotating Angle	Scaling Factor	CORDIC Gain
0	1.0	45.000°	1.41421	1.41421
1	0.5	26.565°	1.11803	1.58114
2	0.25	14.036°	1.03078	1.62980
3	0.125	7.125°	1.00778	1.64248
4	0.0625	3.576°	1.00195	1.64569
5	0.03125	1.790°	1.00049	1.64649
6	0.015625	0.895°	1.00012	1.64669



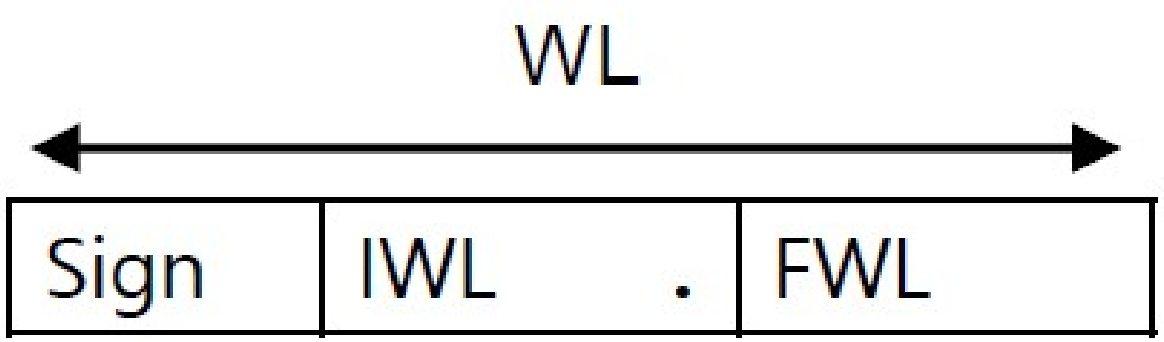
- **Question 2:** Another important design parameter is the data type of the variables. Is one data type sufficient for every variable or is it better for each variable to have a different type? Does the best data type depend on the input data? What is the best technique for the designer to determine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

Only one data type is not sufficient, the data type has to be selected with the right number of bits and format to be good enough for the application's accuracy & performance required, for example:

For an iteration variable that just requires INTEGER and UNSIGNED values, we can use UNSIGNED INTEGER (we don't need to use FLOATING POINT, since this type of data takes more bits and requires additional hardware resources like **DSPs**.

Another example: Some applications required decimal places. But FLOATING POINT can have unnecessary number of decimal places, for those cases we can customize the data type (less INTEGER part bits and less FRACTIONAL WORD LENGTH..

On HLS C++ we have what is called **"Fixed-Point Optimization"** for number of digits BEFORE/ AFTER the **radix point**



S = sign bit

IWL = integer word length

WL = word length

FWL = fractional word length

The least number of bits used for "**Fixed-Point Optimization"** this will reduce the number of hardware resources specially **DSP's.**

This "**Fixed-Point Optimization"** can be configure like this example: "**typedef ap_fixed<8,0> data_t"**
Here 8 bits for integer part and 0 after radix point.
During the optimization that I did, I had to use

CORDIC code version	DSP	FF	LUT	OBSERVATIONS
typedef float data_t	5	932	1578	Original version
typedef ap_fixed<8,0> data_t	1	22	161	Optimized 1: Actually not workable solution, since I get error "ERR: [SIM 100] 'csim_design' failed: nonzero return value"
typedef ap_fixed<16,3>	7	711	924	Optimized 2 Floating point solution has less DSPs, but this one has less LUT, really depends on what resources we want to optimize. This solution passes all test.

The technique to determine the data type configuration, is to understand the range of values of each of the standard data types, but in cases that we want to further optimize to increase performance and/or reduce resources, we can customize the data type by using "**Fixed-Point Optimization"**

- **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?

The **multiplication/division** operation is a very expensive computing processing task; this takes drastically more resources (especially **DSP's**) than Adding/ Subtracting and Bitwise SHIFTING.

The CORDIC algorithm/ technique is based on avoiding **multiplications at all,** so using ADDITIONS & SHIFTING instead.

This technique consist of iterations of (+)/ (-) angle rotations, and as we move on the iteration, the next rotation is smaller and smaller.

For every iteration, the ROTATION gets smaller, but there's a point that the ROTATION becomes INSIGNIFICANT (based on the application's accuracy required), that's when the ITERATIONS should STOP.

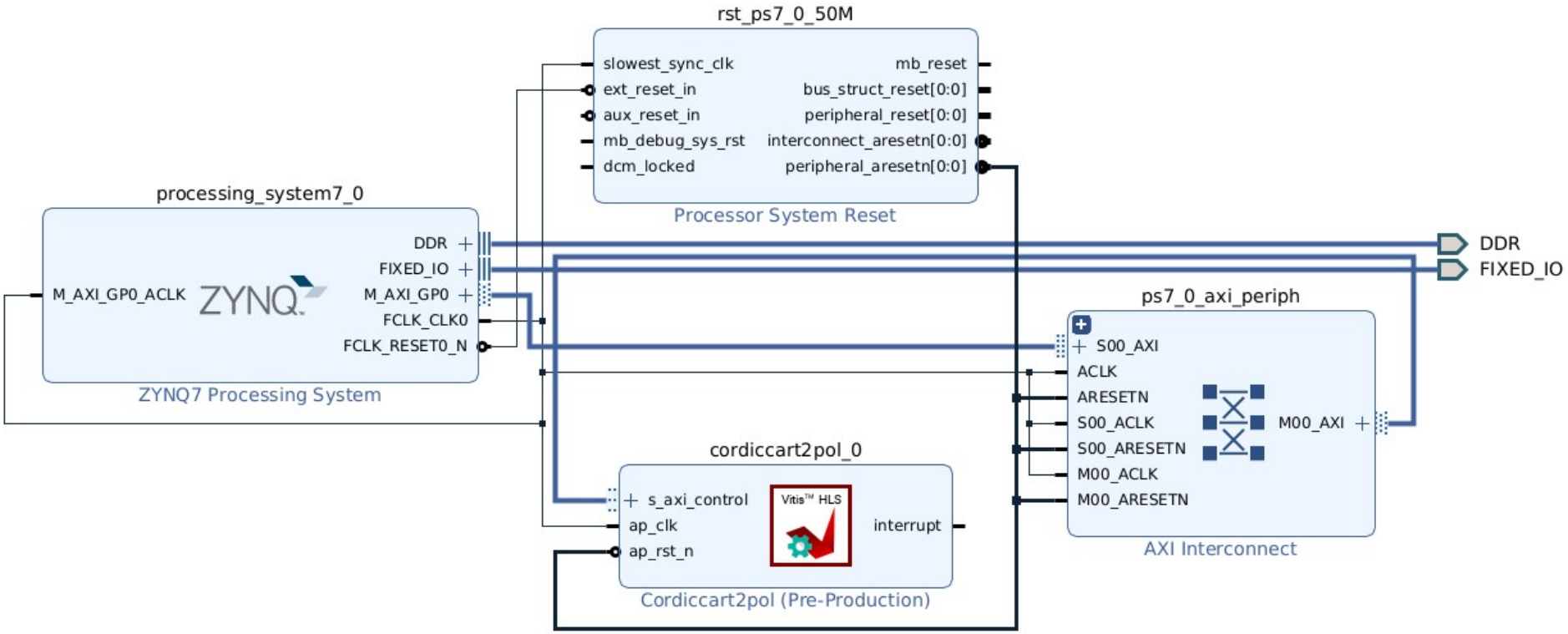
This algorithm is very popular for computing systems due to its simplicity.

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

- The data type affected the size of the LUT, because the lesser bits allocated for the table, the LUT table will be smaller, the good part is that would take less resources.
- Value ranges per input & outputs:
  - X & Y value range is (-1 to +1)
  - R value range is (-1 to +1)
  - Theta value range is (-2pi to +2pi)

```
typedef ap_fixed<4,3> data_t;

#define W           5
#define I           2
```

- Reducing the number of fractional bits reduces significantly especially the number of LUT resources
- The advantage of having the LUT tables is that we saved in advance data tables that can be re-used instead of doing the CORDIC rotations calculations


rst_ps7_0_50M
slowest_sync_clk
ext_reset_in
aux_reset_in
mb_debug_sys_rst
dcm_locked
mb_reset
bus_struct_reset[0:0]
peripheral_reset[0:0]
interconnect_aresetn[0:0]
peripheral_aresetn[0:0]
Processor System Reset
processing_system7_0
M_AXI_GP0_ACLK
ZYNQ
DDR
FIXED_IO
M_AXI_GP0
FCLK_CLK0
FCLK_RESET0_N
ZYNQ7 Processing System
DDR
FIXED_IO
ps7_0_axi_periph
S00_AXI
ACLK
ARESETN
S00_ACLK
S00_ARESETN
M00_ACLK
M00_ARESETN
M00_AXI
AXI Interconnect
cordiccart2pol_0
s_axi_control
ap_clk
ap_rst_n
Vitis™ HLS
interrupt
Cordiccart2pol (Pre-Production)

```
//
//------------------------Address Info-------------------
// 0x00 : Control signals
//        bit 0  - ap_start (Read/Write/COH)
//        bit 1  - ap_done (Read/COR)
//        bit 2  - ap_idle (Read)
//        bit 3  - ap_ready (Read/COR)
//        bit 7  - auto_restart (Read/Write)
//        bit 9  - interrupt (Read)
//        others - reserved
// 0x04 : Global Interrupt Enable Register
//        bit 0  - Global Interrupt Enable (Read/Write)
//        others - reserved
// 0x08 : IP Interrupt Enable Register (Read/Write)
//        bit 0 - enable ap_done interrupt (Read/Write)
//        bit 1 - enable ap_ready interrupt (Read/Write)
//        others - reserved
// 0x0c : IP Interrupt Status Register (Read/TOW)
//        bit 0 - ap_done (Read/TOW)
//        bit 1 - ap_ready (Read/TOW)
//        others - reserved
// 0x10 : Data signal of x
//        bit 31~0 - x[31:0] (Read/Write)
// 0x14 : reserved
// 0x18 : Data signal of y
//        bit 31~0 - y[31:0] (Read/Write)
// 0x1c : reserved
// 0x20 : Data signal of r
//        bit 31~0 - r[31:0] (Read)
// 0x24 : Control signal of r
//        bit 0  - r_ap_vld (Read/COR)
//        others - reserved
// 0x30 : Data signal of theta
//        bit 31~0 - theta[31:0] (Read)
// 0x34 : Control signal of theta
//        bit 0  - theta_ap_vld (Read/COR)
//        others - reserved
// (SC = Self Clear, COR = Clear on Read, TOW = Toggle on Write, COH = Clear on Handshake)
```

CORDIC Testbench

This notebook is to test the implementation of a CORDIC running on the programmable logic. The CORDIC is used to convert cartesian to polar coordinates. The output is compared with a Python calculation of the coordinate transform. It takes in x and y and gives out r and theta where r is the radius and theta is the angle.

In [1]:
```
from pynq import Overlay
from pynq import MMIO
import numpy as np
import struct
import binascii
import cmath
import random
import matplotlib.pyplot as plt

NUM_SAMPLES = 50
```

In [2]:
```
ol=Overlay('./design_1_wrapper.bit') #Change name of bitstream as required
```

In [3]:
```
cordic_ip=MMIO(0x43C00000,10000) #Change base address as required
```

In [4]:
```
r_error=np.zeros(NUM_SAMPLES)
theta_error=np.zeros(NUM_SAMPLES)
ind=np.arange(NUM_SAMPLES)
r_rmse=np.zeros(NUM_SAMPLES)
theta_rmse=np.zeros(NUM_SAMPLES)
```

In [5]:
```
for i in range(NUM_SAMPLES):
    #Generating random inputs
    x=random.uniform(-1,1)
    y=random.uniform(-1,1)

    #Computing golden output
    cn=complex(x,y)
    cn=cmath.polar(cn)

    #Converting input to bytes to be sent to FPGA
    x=(struct.unpack('<I', struct.pack('<f', x))[0])
    y=(struct.unpack('<I', struct.pack('<f', y))[0])

    #Writing values to the FPGA
    cordic_ip.write(0x10,x)                          #Change the offset as mentioned in vivado file
    cordic_ip.write(0x18,y)                          #Change the offset as mentioned in vivado file

    #Starting and stopping the IP (Don't change this)
    cordic_ip.write(0x00,1)
    cordic_ip.write(0x00,0)

    #Reading from IP
    r=hex(cordic_ip.read(0x20))                      #Change the offset as mentioned in vivado file
    r=r[2:]
    theta=hex(cordic_ip.read(0x30))                  #Change the offset as mentioned in vivado file
    theta=theta[2:]

    #Converting to float
    if r!=0:
        r=struct.unpack('>f', binascii.unhexlify(r))
        r=r[0]
    if theta!=0:
        theta=struct.unpack('>f', binascii.unhexlify(theta))
        theta=theta[0]

    #Comparing with golden output
    r_error[i]="{0:.6f}".format(abs(r-cn[0]))
    theta_error[i]="{0:.6f}".format(abs(theta-cn[1]))
```

Verifying Functionality

In [6]:

```
sum_sq_r=0
sum_sq_theta=0
for i in range(NUM_SAMPLES):
    sum_sq_r =sum_sq_r+(r_error[i]*r_error[i])
    r_rmse = np.sqrt(sum_sq_r / (i+1))
    sum_sq_theta =sum_sq_theta+(theta_error[i]*theta_error[i])
    theta_rmse = np.sqrt(sum_sq_theta / (i+1))
print("Radius RMSE: ", r_rmse, "Theta RMSE:", theta_rmse)
if r_rmse<0.001 and theta_rmse<0.001:
    print("PASS")
else:
    print("FAIL")
```

```
Radius RMSE:  0.0 Theta RMSE: 1.7230786401090347e-05
PASS
```

Displaying Errors

In [8]:

```
plt.figure(figsize=(10, 5))
plt.subplot(1,2,1)
plt.bar(ind,r_error)
plt.title("Radius Error")
plt.xlabel("Index")
plt.ylabel("Error")
#plt.xticks(ind)
plt.tight_layout()

plt.subplot(1,2,2)
plt.bar(ind,theta_error)
plt.title("Theta Error")
plt.xlabel("Index")
plt.ylabel("Error")
#plt.xticks(ind)
plt.tight_layout()
```