REAL-TIME DIGITAL SYSTEMS DESIGN AND VERIFICATION WITH FPGAS
ECE 387 – LECTURE 12

PROF. DAVID ZARETSKY

DAVID.ZARETSKY@NORTHWESTERN.EDU

AGENDA

- Quantization
- Cordic Algorithm
- HW #5

TRIGONOMETRIC FUNCTIONS

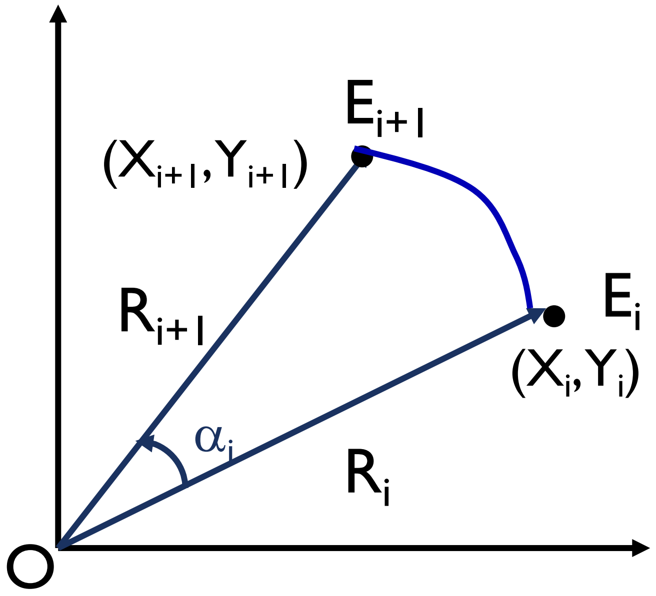
- **CO**ordinate **R**otational **DI**gital **C**omputer
- Jack E. Volder (1959)
- The CORDIC algorithm provides an iterative method of performing vector rotations by arbitrary angles using only shifts and adds.
- Very fast method for implementing SIN and COS in hardware.

CORDIC ALGORITHMS

- A convergence method for evaluating trigonometric (and other) functions
  - if a unit-length vector with end point at (X,Y) = (1,0) is rotated by an angle Z, its new end point will be at (X,Y) = (cos Z, sin Z)
  - simple hardware - shifters, adders, lookup table
- Family of algorithms: rotation, vector mode
  - circular rotations
  - linear rotations
  - hyperbolic rotations

REAL CORDIC ROTATIONS

- The variable Z allows us to keep track of the total rotation over several steps.
- If Z_0 is the initial rotation goal and if the α_i angles are selected at each step such that after n iterations Z_n tends to 0, then E_n will be the end point after rotation by angle Z_0



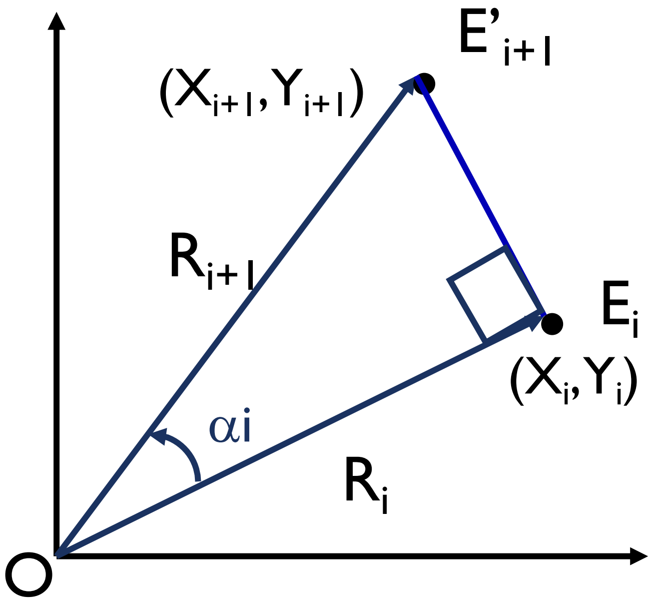
If vector OE_i is rotated about the origin by an angle α_i, the new vector OE_{i+1} will have the coordinates

Real rotation: E_{i+1}

$$X_{i+1} = X_i \cos \alpha_i - Y_i \sin \alpha_i$$
$$Y_{i+1} = Y_i \cos \alpha_i + X_i \sin \alpha_i$$
$$Z_{i+1} = Z_i - \alpha_i$$

PSEUDO CORDIC ROTATIONS

- Expansion factor $K = \prod(1 + \tan^2\alpha_i)^{1/2}$ depends on the rotation angles $\alpha_1, \alpha_2, \ldots, \alpha_n$.
- If we always rotate by the same angles, K is a constant.



Pseudo rotation: E'_{i+1}

$$X_{i+1} = X_i - Y_i \tan \alpha_i$$
$$Y_{i+1} = Y_i + X_i \tan \alpha_i$$
$$Z_{i+1} = Z_i - \alpha_i$$

Pseudo rotations increase the vector length to

$$R_{i+1} = R_i(1 + \tan^2\alpha_i)^{1/2}$$

CALCULATING K EXPANSION FACTOR

- K_i can be ignored in the iterative process and then applied afterward as a scaling factor

```
float K = 1.0;
for ( int i = 0; i < N; i++ )
{
    K *= sqrt(1.0 + pow(2,-2*i));
}
```

BASIC CORDIC ROTATIONS

- To simplify pseudo rotations, pick α_i such that
  $\tan \alpha_i = d_i 2^{-i}$ where $d_i \in \{-1, 1\}$.
- Then
  - $X_{i+1} = X_i - d_i Y_i 2^{-i}$
  - $Y_{i+1} = Y_i + d_i X_i 2^{-i}$
  - $Z_{i+1} = Z_i - d_i \tan^{-1} 2^{-i}$
- Computation of X_{i+1} and Y_{i+1} requires an i-bit right shift and an add/subtract;
- Z_{i+1} only requires an add/subtract and one table lookup
- Precompute and store the function $\tan^{-1} 2^{-i}$

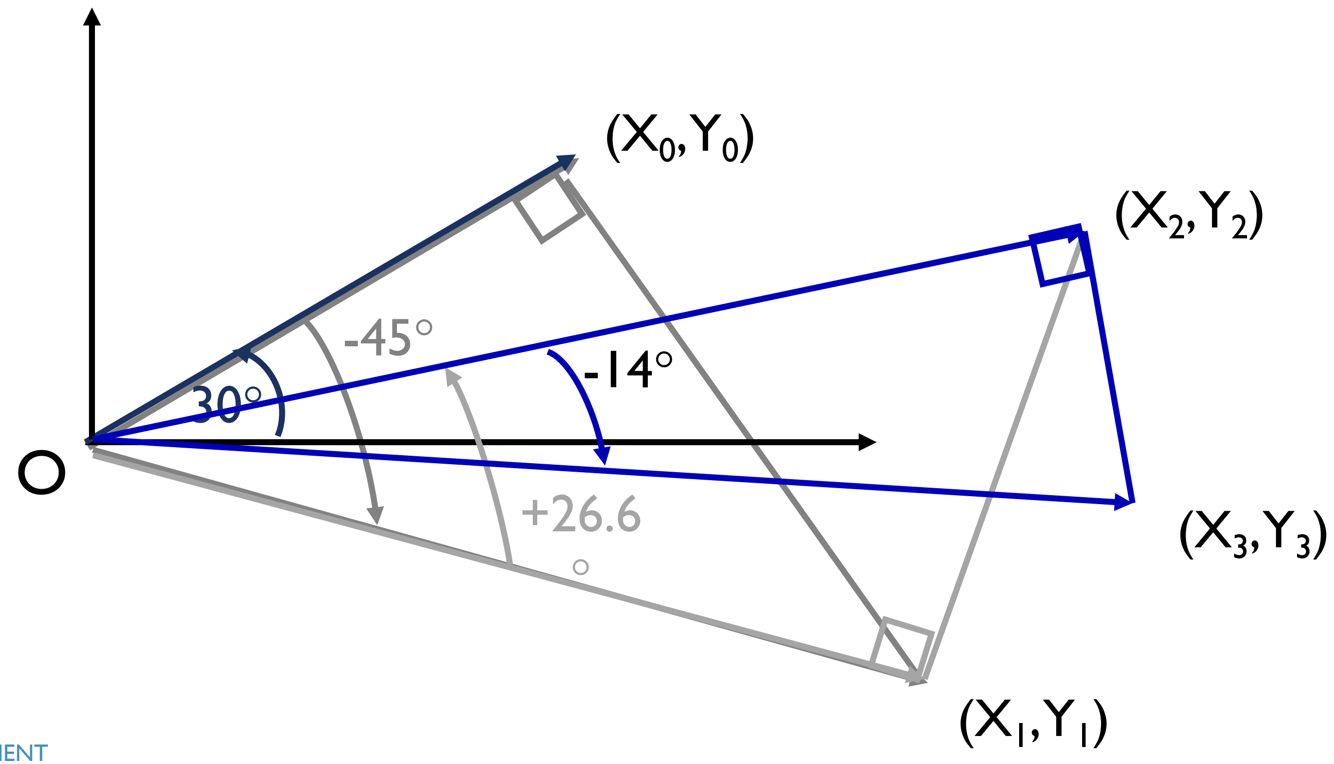
CHOOSING THE ANGLES

i	tan $\alpha_i = 2^{-i}$	$E_i = \tan^{-1}2^{-i}$	d_i	$Z_i - d_iE_i = Z_{i+1}$
0	1.000 000	45.000000	1	30.00 – 45.00 = -15.00
1	0.500000	26.565051	-1	-15.00 + 26.57 = 11.57
2	0.250000	14.036243	1	11.57 – 14.04 = -2.47
3	0.125000	7.125016	-1	-2.47 + 7.13 = 4.66
4	0.062500	3.576334	1	4.66 – 3.58 = 1.08
5	0.031250	1.789910	1	1.08 -1.79 = -0.71
6	0.015625	0.895174	-1	-0.71 + 0.90 = 0.19
7	0.007813	0.447614	1	0.19 - 0.45 = -0.26
8	0.003906	0.223811	-1	-0.26 + 0.22 = -0.04
9	0.001953	0.111906	-1	-0.04 + 0.11 = 0.07

$Z_{i+1} \rightarrow$ zero

ROTATING THE ANGLES

- Illustration of the first three rotations for a Z of 30°



CIRCULAR ROTATION MODE

- Choosing $d_i = \text{sign}(Z_i) \in \{-1,1\}$ to force Z to 0 gives the rotation mode Cordic iterations

  $X_{i+1} = X_i - d_i Y_i 2^{-i}$

  $Y_{i+1} = Y_i + d_i X_i 2^{-i}$

  $Z_{i+1} = Z_i - d_i E_i$ where $E_i = \tan^{-1} 2^{-1}$

- After n iterations, when Z_n is sufficiently close to 0, then we have $Z = \sum \alpha_i$ and

  $X_n = K(X \cos Z - Y \sin Z)$ where $K = 1.646\ 760\ 258 \ldots$

  $Y_n = K(Y \cos Z + X \sin Z)$

  $Z_n = 0$

  Rule: Choose $d_i \in \{-1,1\}$ such that $Z \to 0$

- Computes cos Z and sin Z by starting with
  $X = 1/K = 0.607\ 252\ 935 \ldots$ and $Y = 0$

- For k bits of precision, run it k iterations since for large i > k, $\tan^{-1} 2^{-i} \approx 2^{-i}$

CONVERGENCE DOMAIN

- Convergence of Z to 0 happens because each angle is more than half the previous angle.
- The domain of convergence is $0° \leq Z \leq 99.7°$

  which is the sum of all the angles
- Outside this range, we can use trig identities to convert to the range
  - $\cos(Z \pm 2j\pi) = \cos Z \qquad \sin(Z \pm 2j\pi) = \sin Z$
  - $\cos(Z - \pi) = -\cos Z \qquad \sin(Z - \pi) = -\sin Z$

ROTATION EXAMPLE

- Computing cos 30° (= 0.866 025) and sin 30° (=0.500 000)

i	d_i	$\tan \alpha_i = 2^{-i}$	$E_i=\tan^{-1}2^{-i}$	$X_{i+1} \rightarrow \cos$	$Y_{i+1} \rightarrow \sin$	$Z_{i+1} \rightarrow 0$
				1/K = 0.607 253	0.000 000	30.000 000
0	1	1.000 000	45.000000	0.607 253	0.607 253	-15.000 000
1	-1	0.500000	26.565051	0.910 880	0.303 627	11.565 051
2	1	0.250000	14.036243	0.834 973	0.531 347	-2.471 192
3	-1	0.125000	7.125016	0.901 391	0.426 975	4.653 824
4	1	0.062500	3.576334	0.874 705	0.483 312	1.077 490
5	1	0.031250	1.789910	0.859 602	0.510 647	-0.712 420
6	-1	0.015625	0.895174	0.867 581	0.497 216	0.182 754
7	1	0.007813	0.447614	0.863 697	0.503 994	-0.264 860
8	-1	0.003906	0.223811	0.865 666	0.500 620	-0.041 049
...	...	...	...	...	...	...

CORDIC IN SOFTWARE

```
// Constants
#define K          1.646760258121066
#define CORDIC_1K QUANTIZE_F(1/K)
#define PI QUANTIZE_F(M_PI)
#define HALF_PI QUANTIZE_F(M_PI/2)

void cordic_stage(short k, short c, short *x, short *y, short *z)
{
    // inputs
    short xk = *x;
    short yk = *y;
    short zk = *z;

    // cordic stage
    short d = (zk >= 0) ? 0 : -1;
    short tx = xk - (((yk >> k) ^ d) - d);
    short ty = yk + (((xk >> k) ^ d) - d);
    short tz = zk - ((c ^ d) - d);

    // outputs
    *x = tx;
    *y = ty;
    *z = tz;
}
```

```
void cordic(int rad, short *s, short *c)
{
    short x = CORDIC_1K, y = 0;
    int r = rad;

    while ( r > PI  ) r -= 2*PI;
    while ( r < -PI ) r += 2*PI;

    if ( r > HALF_PI ) {
         r -= PI;   x = -x;   y = -y;
    }
    else if ( r < -HALF_PI ) {
        r += PI;  x = -x;   y = -y;
    }

    short z = r;

    for ( int k = 0; k < CORDIC_NTAB; k++ ) {
       cordic_stage(k, CORDIC_TABLE[k], &x, &y, &z);
    }

    *c = x;
    *s = y;
}
```

PERFORMANCE COMPARISON

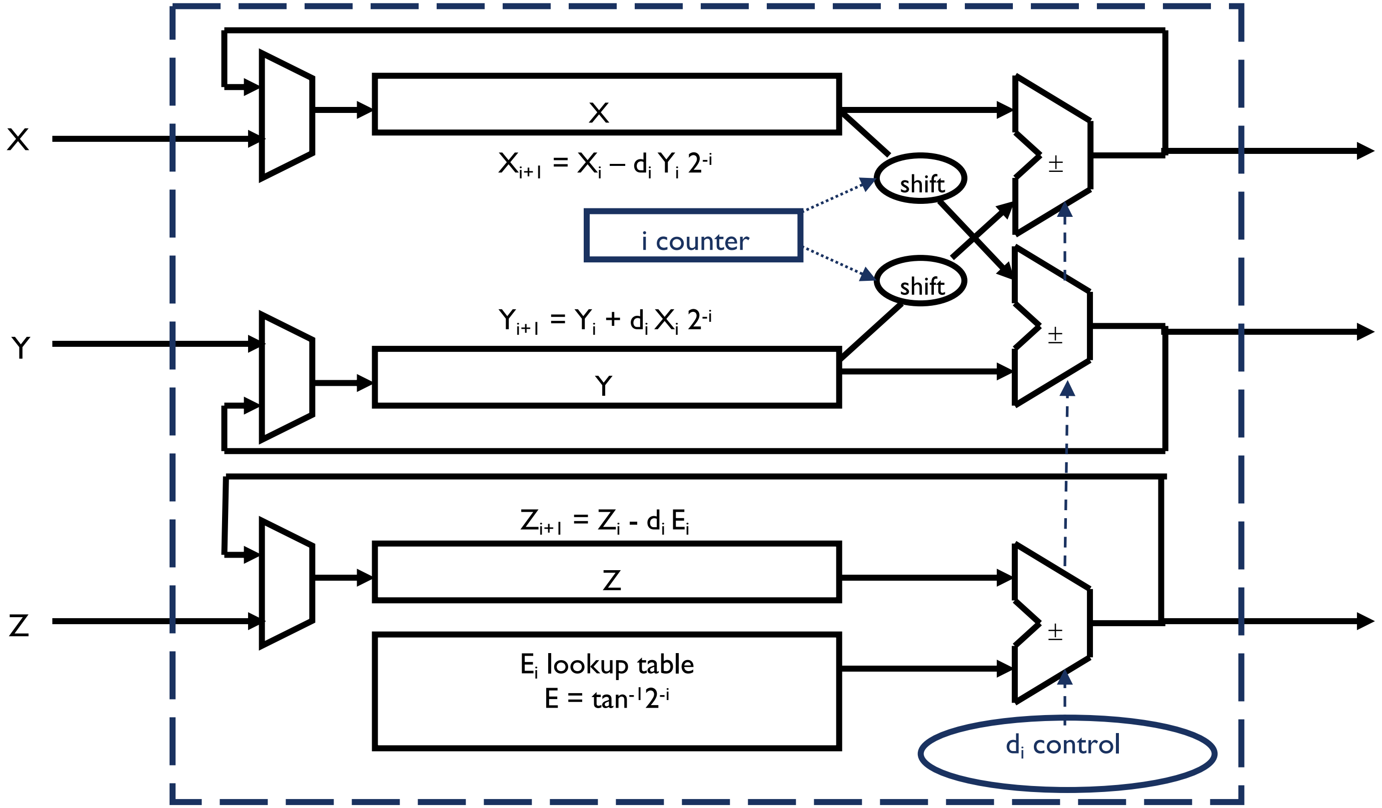
```
int main(int argc, char **argv)
{
    for ( int i = 0; i < CORDIC_NTAB; i++ )
    {
        CORDIC_TABLE[i] = QUANTIZE_F( atan(pow(2, -i)) ));
    }

    for ( int i = -360; i <= 360; i++ )
    {
        float p = i * M_PI / 180;
        int p_fixed = QUANTIZE_F(p);
        short s = 0, c = 0;
        cordic(p_fixed, &s, &c);

        printf("theta %d = %08x --> sin: %8.4f --> cos: %8.4f\n", i, p_fixed,
        DEQUANTIZE_F(s) - sin(p), DEQUANTIZE_F(c) - cos(p));
    }

    return 0;
}
```

PERFORMANCE RESULTS

```
theta -45 = ffffcdbd --> sin:  -0.0000  --> cos:   0.0002
theta -44 = ffffceda --> sin:  -0.0002  --> cos:  -0.0003
theta -43 = ffffcff8 --> sin:  -0.0001  --> cos:  -0.0002
theta -42 = ffffd116 --> sin:  -0.0001  --> cos:   0.0001
theta -41 = ffffd234 --> sin:   0.0001  --> cos:   0.0002
theta -40 = ffffd352 --> sin:   0.0000  --> cos:   0.0001
theta -39 = ffffd470 --> sin:  -0.0000  --> cos:  -0.0004
theta -38 = ffffd58e --> sin:  -0.0000  --> cos:  -0.0001
theta -37 = ffffd6ac --> sin:   0.0000  --> cos:  -0.0001
theta -36 = ffffd7ca --> sin:  -0.0000  --> cos:  -0.0006
theta -35 = ffffd8e8 --> sin:  -0.0001  --> cos:  -0.0004
theta -34 = ffffda06 --> sin:  -0.0000  --> cos:  -0.0003
theta -33 = ffffdb24 --> sin:  -0.0000  --> cos:  -0.0001
theta -32 = ffffdc42 --> sin:  -0.0002  --> cos:  -0.0003
theta -31 = ffffdd60 --> sin:  -0.0000  --> cos:   0.0003
theta -30 = ffffde7e --> sin:   0.0000  --> cos:   0.0002
theta -29 = ffffdf9c --> sin:   0.0001  --> cos:   0.0003
theta -28 = ffffe0ba --> sin:  -0.0003  --> cos:  -0.0001
theta -27 = ffffe1d8 --> sin:  -0.0002  --> cos:  -0.0000
theta -26 = ffffe2f6 --> sin:  -0.0002  --> cos:   0.0001
theta -25 = ffffe414 --> sin:  -0.0001  --> cos:  -0.0002
theta -24 = ffffe532 --> sin:   0.0001  --> cos:  -0.0002
theta -23 = ffffe650 --> sin:   0.0001  --> cos:  -0.0000
theta -22 = ffffe76d --> sin:   0.0003  --> cos:   0.0001
theta -21 = ffffe88b --> sin:  -0.0000  --> cos:   0.0003
theta -20 = ffffe9a9 --> sin:  -0.0000  --> cos:   0.0005
theta -19 = ffffeac7 --> sin:   0.0000  --> cos:  -0.0000
theta -18 = ffffebe5 --> sin:   0.0002  --> cos:  -0.0004
theta -17 = ffffed03 --> sin:   0.0003  --> cos:  -0.0002
theta -16 = ffffee21 --> sin:  -0.0002  --> cos:   0.0002
theta -15 = ffffef3f --> sin:  -0.0001  --> cos:  -0.0003
theta -14 = fffff05d --> sin:  -0.0000  --> cos:  -0.0001
theta -13 = fffff17b --> sin:   0.0000  --> cos:  -0.0002
theta -12 = fffff299 --> sin:   0.0002  --> cos:   0.0001
theta -11 = fffff3b7 --> sin:  -0.0000  --> cos:  -0.0002
theta -10 = fffff4d5 --> sin:  -0.0001  --> cos:   0.0002
theta -9 = fffff5f3 --> sin:   0.0001  --> cos:   0.0001
theta -8 = fffff711 --> sin:   0.0001  --> cos:   0.0005
theta -7 = fffff82f --> sin:  -0.0001  --> cos:   0.0002
theta -6 = fffff94d --> sin:  -0.0001  --> cos:   0.0002
theta -5 = fffffa6b --> sin:  -0.0000  --> cos:   0.0003
theta -4 = fffffb89 --> sin:   0.0001  --> cos:   0.0001
theta -3 = fffffca7 --> sin:   0.0001  --> cos:  -0.0001
theta -2 = fffffdc5 --> sin:   0.0001  --> cos:  -0.0001
theta -1 = fffffee3 --> sin:   0.0001  --> cos:  -0.0000
theta 0 = 00000000 --> sin:   0.0001  --> cos:  -0.0001
theta 1 = 0000011d --> sin:   0.0000  --> cos:  -0.0002
theta 2 = 0000023b --> sin:   0.0000  --> cos:  -0.0001
theta 3 = 00000359 --> sin:   0.0000  --> cos:  -0.0000
theta 4 = 00000477 --> sin:   0.0002  --> cos:  -0.0000
theta 5 = 00000595 --> sin:   0.0000  --> cos:  -0.0002
theta 6 = 000006b3 --> sin:   0.0001  --> cos:  -0.0001
theta 7 = 000007d1 --> sin:   0.0001  --> cos:   0.0000
theta 8 = 000008ef --> sin:  -0.0001  --> cos:  -0.0001
theta 9 = 00000a0d --> sin:  -0.0000  --> cos:   0.0000
theta 10 = 00000b2b --> sin:   0.0001  --> cos:  -0.0001
theta 11 = 00000c49 --> sin:   0.0002  --> cos:   0.0000
theta 12 = 00000d67 --> sin:  -0.0000  --> cos:  -0.0002
theta 13 = 00000e85 --> sin:  -0.0001  --> cos:  -0.0001
theta 14 = 00000fa3 --> sin:   0.0001  --> cos:  -0.0001
theta 15 = 000010c1 --> sin:   0.0001  --> cos:   0.0000
theta 16 = 000011df --> sin:   0.0002  --> cos:  -0.0002
theta 17 = 000012fd --> sin:  -0.0003  --> cos:  -0.0000
theta 18 = 0000141b --> sin:   0.0002  --> cos:  -0.0002
theta 19 = 00001539 --> sin:  -0.0001  --> cos:  -0.0001
theta 20 = 00001657 --> sin:  -0.0000  --> cos:  -0.0002
theta 21 = 00001775 --> sin:  -0.0000  --> cos:  -0.0001
theta 22 = 00001893 --> sin:   0.0001  --> cos:  -0.0001
theta 23 = 000019b0 --> sin:   0.0002  --> cos:  -0.0001
theta 24 = 00001ace --> sin:  -0.0002  --> cos:   0.0002
theta 25 = 00001bec --> sin:  -0.0000  --> cos:   0.0001
theta 26 = 00001d0a --> sin:   0.0001  --> cos:   0.0001
theta 27 = 00001e28 --> sin:   0.0001  --> cos:   0.0000
theta 28 = 00001f46 --> sin:   0.0002  --> cos:   0.0001
theta 29 = 00002064 --> sin:  -0.0001  --> cos:  -0.0001
theta 30 = 00002182 --> sin:  -0.0001  --> cos:   0.0001
theta 31 = 000022a0 --> sin:  -0.0000  --> cos:   0.0001
theta 32 = 000023be --> sin:   0.0001  --> cos:  -0.0001
theta 33 = 000024dc --> sin:  -0.0001  --> cos:  -0.0001
theta 34 = 000025fa --> sin:  -0.0001  --> cos:  -0.0001
theta 35 = 00002718 --> sin:  -0.0000  --> cos:  -0.0001
theta 36 = 00002836 --> sin:  -0.0001  --> cos:   0.0000
theta 37 = 00002954 --> sin:   0.0001  --> cos:  -0.0001
theta 38 = 00002a72 --> sin:   0.0001  --> cos:  -0.0000
theta 39 = 00002b90 --> sin:   0.0001  --> cos:  -0.0002
theta 40 = 00002cae --> sin:  -0.0000  --> cos:   0.0000
theta 41 = 00002dcc --> sin:  -0.0001  --> cos:   0.0001
theta 42 = 00002eea --> sin:   0.0001  --> cos:   0.0000
theta 43 = 00003008 --> sin:   0.0001  --> cos:  -0.0001
theta 44 = 00003126 --> sin:   0.0002  --> cos:  -0.0002
theta 45 = 00003243 --> sin:  -0.0000  --> cos:   0.0000
```

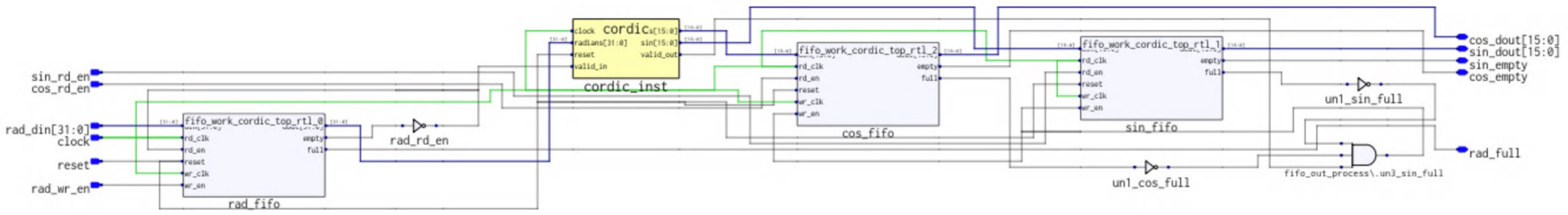
ERROR IS LESS THAN +/-0.001

CIRCULAR CORDIC HARDWARE

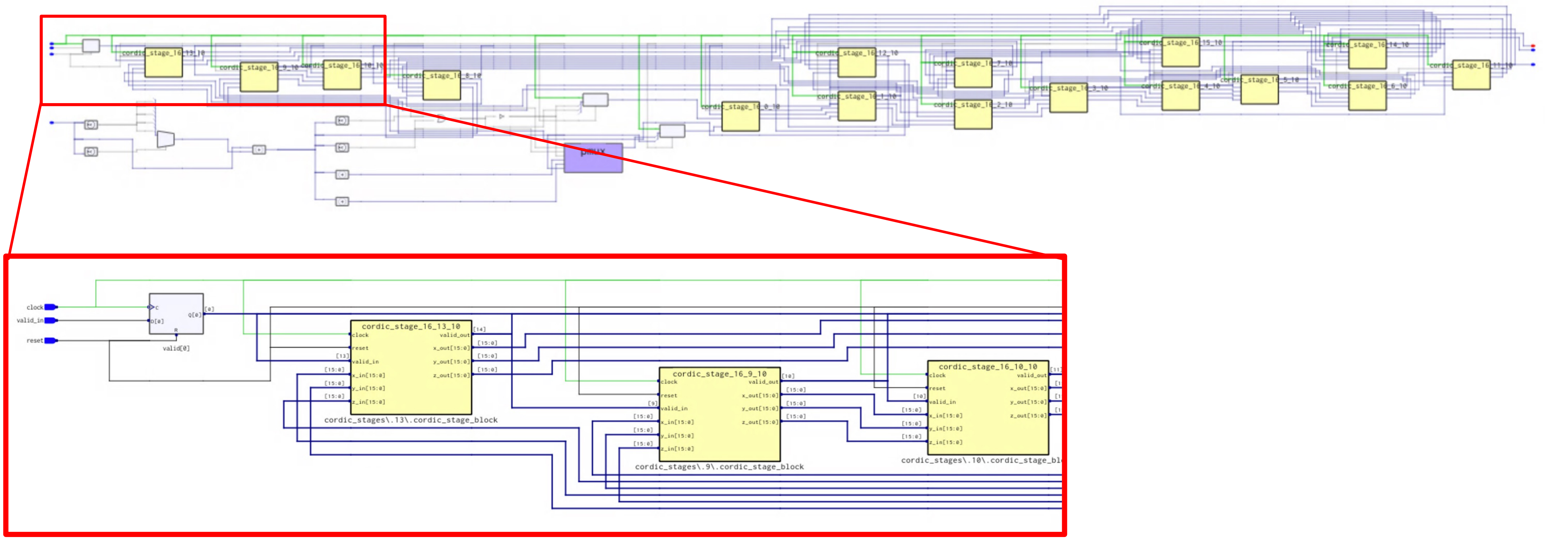


CORDIC TOP-LEVEL IMPLEMENTATION

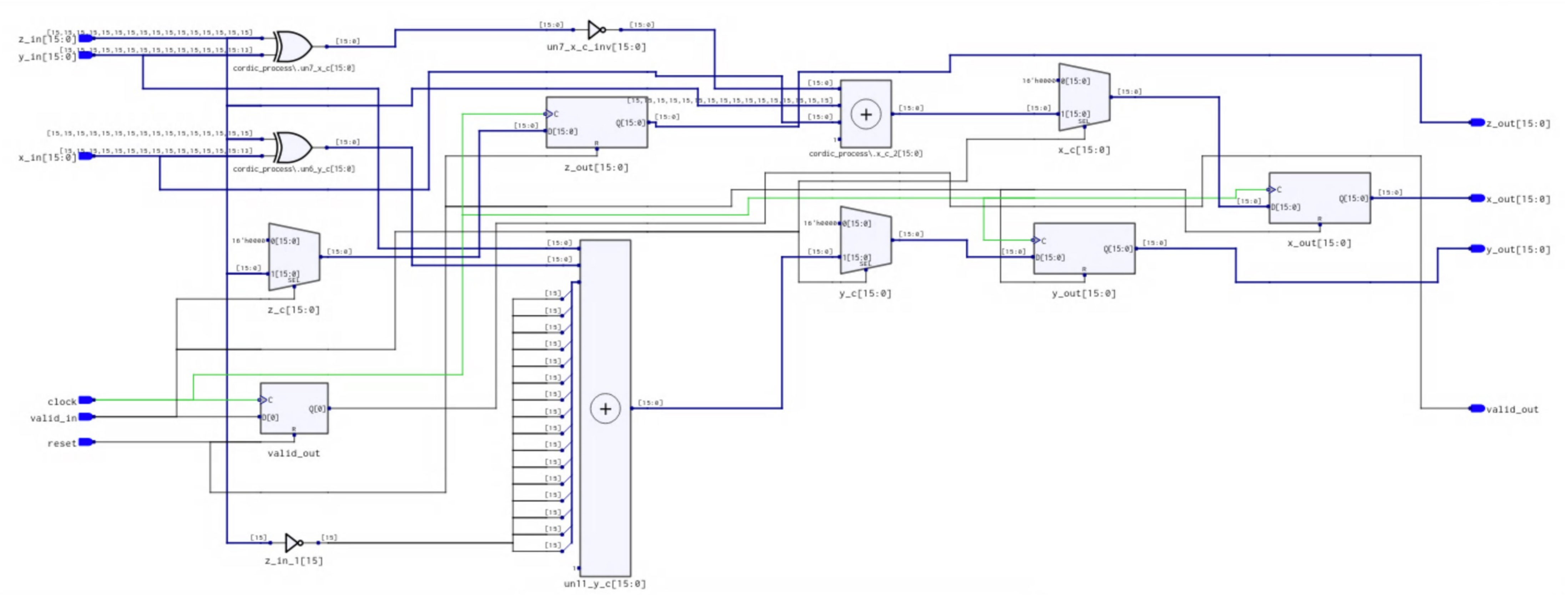
- Input: Radians
- Output: Sin & Cos



CORDIC 16-STAGE PIPELINE



CORDIC STAGE



CORDIC SUMMARY

- Can compute virtually all trig functions of common interest
- Using approximations, we can simplify trigonometric computations using adders, shifters, and lookup tables
- When the number of iterations is fixed, K is constant
- In hardware Cordic can be easily pipelined
- We always need k iterations for k digits of precision
- Cordic can be extended to higher radices
  - for base 4, $d_i \in \{-2,-1,1,2\}$ and the number of iterations will be cut in half with essentially the same hardware

PROGRAM ASSIGNMENT

- Build a quantized Cordic algorithm that generates the Sin & Cos values
- Implement 16-stage hardware pipelined architecture
- The streaming Cordic implementation should produce a new value every cycle
- Simulate in software for theta in range -360 to 360 degrees, and generate quantized outputs for sin and cos
- Compare fixed point results to the software implementation, and determine the precision of quantization error.

NEXT…

- Digital Signal Processing Applications
- Final Project: FM Radio
- Form Groups for Final Project