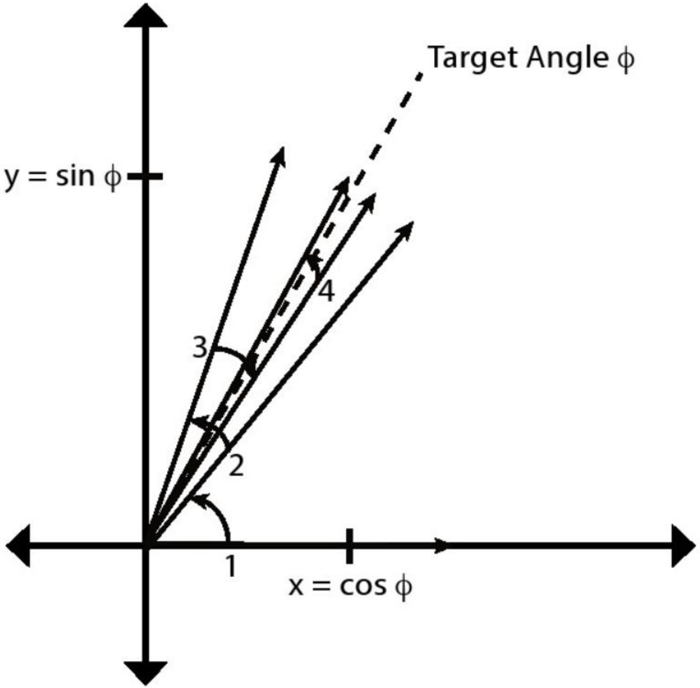
LAB_B

CORDIC

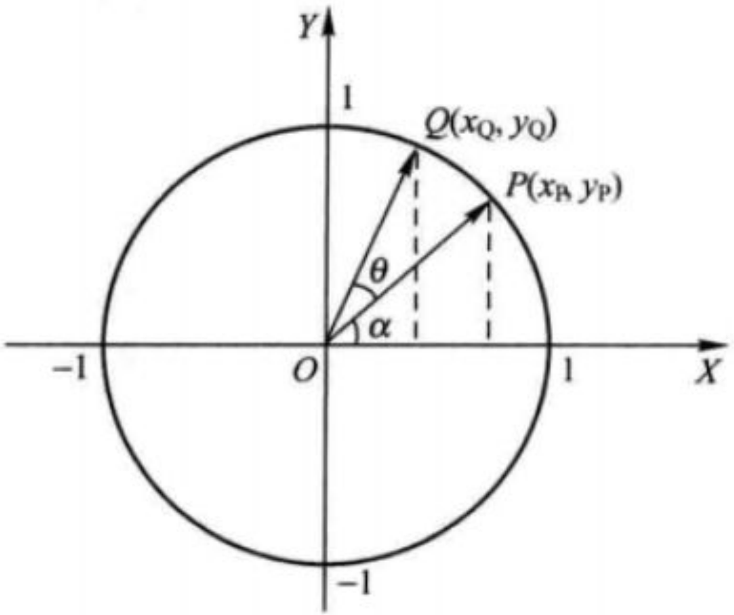
CORDIC

- Coordinate Rotation Digital Computer
- CORDIC is a simple and efficient algorithm to calculate trigonometric functions
- Only requires are additions, subtractions, bitshift and lookup tables

CORDIC



Rotation Mode



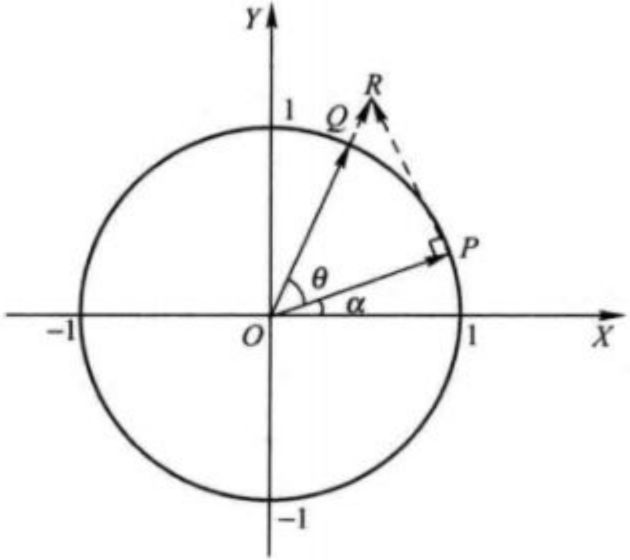
$$\begin{cases} x_Q = \cos(\alpha+\theta) \\ y_Q = \sin(\alpha+\theta) \end{cases}$$

$$\begin{cases} x_Q = \cos\alpha\cos\theta - \sin\alpha\sin\theta \\ y_Q = \sin\alpha\cos\theta + \cos\alpha\sin\theta \end{cases}$$

$$\begin{cases} x_Q = x_P\cos\theta - y_P\sin\theta \\ y_Q = y_P\cos\theta + x_P\sin\theta \end{cases}$$

$$\begin{cases} x_Q = \cos\theta(x_P - y_P\tan\theta) \\ y_Q = \cos\theta(y_P + x_P\tan\theta) \end{cases}$$

Rotation Mode



$$\begin{cases} x_Q = \cos\theta(x_P - y_P \tan\theta) \\ y_Q = \cos\theta(y_P + x_P \tan\theta) \end{cases}$$

$$\begin{cases} x_R = x_P - y_P \tan\theta \\ y_R = y_P + x_P \tan\theta \end{cases}$$

$$K_i = \cos\theta_i = \cos(\tan^{-1} 2^{-i}) = \frac{1}{\sqrt{1+2^{-2i}}}$$

$$K = \prod_{i=0}^{n-1} \frac{1}{\sqrt{1+2^{-2i}}}$$

Rotation Mode

Decompose θ into the sum of a series of tiny angles $\theta = \sum_{i=0}^{\infty} \theta_i$

The result after the i+1 rotation $\begin{cases} x_R = x_P - y_P \tan\theta \\ y_R = y_P + x_P \tan\theta \end{cases} \longrightarrow \begin{cases} x_{i+1} = x_i - y_i \tan\theta_i \\ y_{i+1} = y_i + x_i \tan\theta_i \end{cases}$

Rotation Mode

Z used to determine the remaining rotation angle
Initial value : z = θ
Target : z = 0

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

i-th rotation angle

$$\theta_i = \tan^{-1}(d_i 2^{-i})$$

$$d_i = \begin{cases} +1 & z_i \geqslant 0 \\ -1 & z_i < 0 \end{cases}$$

$$\begin{cases} x_{i+1} = x_i - y_i \tan\theta_i \\ y_{i+1} = y_i + x_i \tan\theta_i \end{cases} \longrightarrow \begin{cases} x_{i+1} = x_i - d_i y_i 2^{-i} \\ y_{i+1} = y_i + d_i x_i 2^{-i} \end{cases}$$

Lookup tables

i	2^-i	tan(2^-i)°	tan(2^-i) radian
0	1	45°	0.785398163
1	0.5	26.6°	0.463647609
2	0.25	14°	0.244978663
3	0.125	7.1°	0.124354995
4	0.0625	3.6°	0.062418810
5	0.03125	1.8°	0.031239833

Example

$$55° = 45° + 26.6° - 14° - 7.1° + 3.6° + 1.8° - 0.9°$$

variable i	rotation angle	Cumulative rotation angle	compare	Target angle	Remaining angle	Rotation direction
initialize				55	55	Counterclockwise
0	+45	45	<	55	10	Counterclockwise
1	+26.6	71.6	>	55	-16.6	Clockwise
2	-14	57.6	>	55	-2.6	Clockwise
3	-7.1	50.5	<	55	4.5	Counterclockwise
4	+3.6	54.1	<	55	0.9	Counterclockwise
5	+1.8	55.9	>	55	-0.9	Counterclockwise
6	-0.9	55	=	55	0	

Crodic.h

```
#ifndef CORDIC_H
#define CORDIC_H
#include "ap_fixed.h"

typedef unsigned int UINTYPE_12;
typedef ap_fixed<12,2> THETA_TYPE;
typedef ap_fixed<12,2> COS_SIN_TYPE;
const int NUM_ITERATIONS=32;

const int NUM_DEGREE=90;
static THETA_TYPE cordic_phase[64]={0.78539816339744828000,0.46364760900080609000,0.24497866312686414000,0.12435499454676144000
        ,0.06241880999595735000,0.03123983343026827700,0.01562372862047683100,0.00781234106010111110,0.00390623013196697180
        ,0.00195312251647881880,0.00097656218955931946,0.00048828121119489829,0.00024414062014936177,0.00012207031189367021
        ,0.00006103515617420877,0.00003051757811552610,0.00001525878906131576,0.00000762939453110197,0.00000381469726560650
        ,0.00000190734863281019,0.00000095367431640596,0.00000047683715820309,0.00000023841857910156,0.00000011920928955078
        ,0.00000005960464477539,0.00000002980232238770,0.00000001490116119385,0.00000000745058059692,0.00000000372529029846
        ,0.00000000186264514923,0.00000000093132257462,0.00000000046566128731,0.00000000023283064365,0.00000000011641532183
        ,0.00000000005820766091,0.00000000002910383046,0.00000000001455191523,0.00000000000727595761,0.00000000000363797881
        ,0.00000000000181898940,0.00000000000090949470,0.00000000000045474735,0.00000000000022737368,0.00000000000011368684
        ,0.00000000000005684342,0.00000000000002842171,0.00000000000001421085,0.00000000000000710543,0.00000000000000355271
        ,0.00000000000000177636,0.00000000000000088818,0.00000000000000044409,0.00000000000000022204,0.00000000000000011102
        ,0.00000000000000005551,0.00000000000000002776,0.00000000000000001388,0.00000000000000000694,0.00000000000000000347
        ,0.00000000000000000173,0.00000000000000000087,0.00000000000000000043,0.00000000000000000022,0.00000000000000000011};

void cordic(THETA_TYPE theta, COS_SIN_TYPE &s, COS_SIN_TYPE &c);
#endif
```

Cordic.cpp

```
#include "cordic.h"

void cordic(THETA_TYPE theta, COS_SIN_TYPE &s, COS_SIN_TYPE &c)
{
  COS_SIN_TYPE current_cos = 0.60735; x
  COS_SIN_TYPE current_sin = 0.0; y
  COS_SIN_TYPE factor = 1.0;2^-i
  // This loop iteratively rotates the initial vector to find the
  // sine and cosine values corresponding to the input theta angle
  for (int j = 0; j < NUM_ITERATIONS; j++) {
      // Determine if we are rotating by a positive or negative angle
      int sigma = (theta < 0) ? -1 : 1;

      // Multiply previous iteration by 2^(-j)
      COS_SIN_TYPE cos_shift = current_cos * sigma * factor;
      COS_SIN_TYPE sin_shift = current_sin * sigma * factor;

      // Perform the rotation
      current_cos = current_cos - sin_shift;
      current_sin = current_sin + cos_shift;

      // Determine the new theta
      theta = theta - sigma * cordic_phase[j];

      factor = factor / 2;
  }

  // Set the final sine and cosine values
  s = current_sin;  c = current_cos;
}
```

$$d_i = \begin{cases} +1 & z_i \geqslant 0 \\ -1 & z_i < 0 \end{cases}$$

$$\begin{cases} x_{i+1} = x_i - d_i y_i 2^{-i} \\ y_{i+1} = y_i + d_i x_i 2^{-i} \end{cases}$$

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

Testbench

```
#include <math.h>
#include"cordic.h"
#include <stdio.h>
#include <stdlib.h>
using namespace std;
double abs_double(double var){
    if ( var < 0)
    var = -var;
    return var;
}
int main(int argc, char **argv)
{
    FILE *fp;
    COS_SIN_TYPE s;         //sine
    COS_SIN_TYPE c;         //cos
    THETA_TYPE radian;      //radian versuin of degree
    //zs=sin, zc=cos using math.h in VivadoHLS
    double zs, zc;          // sine and cos values calculated from math.
    //Error checking
    double Total_Error_Sin=0.0;
    double Total_error_Cos=0.0;
    double error_sin=0.0, error_cos=0.0;
```

```
    fp=fopen("out.dat","w");
    for(int i=1;i<NUM_DEGREE;i++) {
            radian = i*3.14/180;
            cordic(radian, s, c);
            zs = sin((double)radian);
            zc = cos((double)radian);
            error_sin=(abs_double((double)s-zs)/zs)*100.0;
            error_cos=(abs_double((double)c-zc)/zc)*100.0;
            Total_Error_Sin=Total_Error_Sin+error_sin;
            Total_error_Cos=Total_error_Cos+error_cos;
            fprintf(fp, "degree=%d, radian=%f, cos=%f, sin=%f\n"
                    , i, (double)radian, (double)c, (double)s);
    }
    fclose(fp);
    printf ("Total_Error_Sin=%f, Total_error_Cos=%f, \n"
            , Total_Error_Sin, Total_error_Cos);
    return 0;
}
```

Pragama

1. pipeline

   -The PIPELINE pragma reduces the initiation interval (II) for a function or loop by allowing the concurrent execution of operations.

2. unroll

   -Unroll loops to create multiple independent operations rather than a single collection of operations

3. allocation

   -This defines and can limit the number of register transfer level (RTL) instances and hardware resources used to implement specific functions, loops, operations or cores

Pragama pipeline

```
#include "cordic.h"

void cordic(THETA_TYPE theta, COS_SIN_TYPE &s, COS_SIN_TYPE &c)
{
  COS_SIN_TYPE current_cos = 0.60735;
  COS_SIN_TYPE current_sin = 0.0;
  COS_SIN_TYPE factor = 1.0;

  #pragma HLS pipeline II=2

  for (int j = 0; j < NUM_ITERATIONS; j++) {

      int sigma = (theta < 0) ? -1 : 1;

      COS_SIN_TYPE cos_shift = current_cos * sigma * factor;
      COS_SIN_TYPE sin_shift = current_sin * sigma * factor;

      current_cos = current_cos - sin_shift;
      current_sin = current_sin + cos_shift;

      theta = theta - sigma * cordic_phase[j];

      factor = factor / 2;
  }
  s = current_sin;  c = current_cos;
}
```

Pragama unroll

```
#include "cordic.h"

void cordic(THETA_TYPE theta, COS_SIN_TYPE &s, COS_SIN_TYPE &c)
{
  COS_SIN_TYPE current_cos = 0.60735;
  COS_SIN_TYPE current_sin = 0.0;
  COS_SIN_TYPE factor = 1.0;

  for (int j = 0; j < NUM_ITERATIONS; j++) {

    #pragma HLS unroll

    int sigma = (theta < 0) ? -1 : 1;

    COS_SIN_TYPE cos_shift = current_cos * sigma * factor;
    COS_SIN_TYPE sin_shift = current_sin * sigma * factor;

    current_cos = current_cos - sin_shift;
    current_sin = current_sin + cos_shift;

    theta = theta - sigma * cordic_phase[j];

    factor = factor / 2;
  }
  s = current_sin;  c = current_cos;
}
```

Pragama allocation

```
#include "cordic.h"

void cordic(THETA_TYPE theta, COS_SIN_TYPE &s, COS_SIN_TYPE &c)
{
  COS_SIN_TYPE current_cos = 0.60735;
  COS_SIN_TYPE current_sin = 0.0;
  COS_SIN_TYPE factor = 1.0;

  #pragma HLS allocation instances=mul limit=1 operation

  for (int j = 0; j < NUM_ITERATIONS; j++) {

      int sigma = (theta < 0) ? -1 : 1;

      COS_SIN_TYPE cos_shift = current_cos * sigma * factor;
      COS_SIN_TYPE sin_shift = current_sin * sigma * factor;

      current_cos = current_cos - sin_shift;
      current_sin = current_sin + cos_shift;

      theta = theta - sigma * cordic_phase[j];

      factor = factor / 2;
  }
  s = current_sin;  c = current_cos;
}
```

Timing

	original	pipeline	unrool	allocation
Latancy(cycle)	161	9	9	161

Utilization

	original	pipeline	unrool	allocation
DPS48E	2	0	0	1
FF	181	293	292	238
LUT	280	891	913	312
BRAM	0	0	0	0

Github Link

https://github.com/405410605/LAB_B_CRODIC

Question

1. What are the benefits of specifying the total number of rotation
2. Why do we need a lookup table?