Application Note, V1.0, Feb. 2007

AP16105

XC166 family

Software implementation of Trigonometric functions using CORDIC Algorithm

Microcontrollers

Never stop thinking

Edition 2007-06-21

Published by
Infineon Technologies AG
81726 München, Germany

AP16105		
Revision History:	2007-02	V1.0
Previous Version:	none	

Page	Subjects (major changes since last revision)

Table of Contents **Page**

1 Introduction

Trigonometric functions are often used in embedded systems. Motor drive control applications such as park transform, Clarke transform, PWM generation uses trigonometric functions extensively. Various methods exist to compute the trigonometric functions. These include Taylor series, Curve fitting algorithms, CORDIC algorithm.

This article describes software implementation of the following fixed point trigonometric routines using CORDIC Algorithm for Infineon's XC164CS Microcontroller with MAC unit. The implementation of the algorithm is examined concerning accuracy and efficiency.

- Complex Magnitude
- Sine
- Cosine

Routines are provided for signed two's complement arithmetic. First a brief description of the theory behind the algorithm is presented. Then the theory is extended to the implementation of algorithm in XC164CS Processor after which the numerical errors that occur in the fixed point implementation is discussed.

2 The CORDIC Arithmetic Technique

The Coordinate Rotation DIgital Computer (CORDIC) algorithm is an iterative technique proposed by Volder in 1956. This algorithm can be a very powerful tool in areas where arithmetic or trigonometric function evaluation is heavily utilized, such as digital signal processing, motor control.

The general vector rotational transform rotates a plane vector [X, Y] by an angle θ to produce a new vector point [X_{i+1}, Y_{i+1}] as in (1) and (2). The CORDIC Rotation is achieved by the same principle. It rotates the point [X, Y] in series of steps, which are smaller than θ. This rotation may be in anti clockwise direction (increase in θ) or clock wise direction (decrease in θ). Suppose if we wish to achieve a total rotation of 35°, we may rotate our point 30° anticlockwise, followed by 10° anticlockwise, followed by 5° clockwise.

$X_{i+1}=X_i\cos\theta - Y_i\sin\theta$ (1)

$Y_{i+1}= Y_i\cos\theta + X_i\sin\theta$ (2)

These can be expressed as

$X_{i+1}=\cos\theta * [X_i - Y_i\tan\theta]$ (3)

$Y_{i+1}=\cos\theta * [Yi + X_i\tan\theta]$ (4)

The reason for this simplification is to break down the rotation (θ) into many steps, each of decreasing size and has each step such that tanθ is the power of 2, where θ is the rotational angle. The first seven steps of the set of rotations are shown below.

Table 1 CORDIC Rotation

i	θ	**Tan θ (Decimal) = 2^{-i}**
0	45	1
1	26.565	0.49999
2	14.036	0.24999
3	7.125	0.12499
4	3.516	0.06249
5	1.784	0.03114
6	0.895	0.01562

This would allow us to implement the multiplication by tan θ as a simple bit shift operation (2^{-i}). Hence (3) and (4) reduces to

$X_{i+1}=\cos\theta*[X_i - Y_i\, 2^{-i}]$ (5)

$Y_{i+1}=\cos\theta*[Yi + X_i\, 2^{-i}]$ (6)

From Table (1) it is clear that the total rotation step is 99.88°. Since the rotation can be of clockwise or anticlockwise direction these steps are used to approximate angles between +99.88° to -99.88°. For mathematical simplicity the rotation angles are limited to –90 and +90°. For rotation angles greater than +/- 90° additional rotation is required. The cos θ term (K) is a constant which approaches to 0.6073 after 'n' iterations. The angle θ is accumulated in Z_{i+1}

$Z_{i+1}= Z_i - \delta_i * \tan^{-1}(2^{-i})$ Where δ_i determines the direction of rotation.

The CORDIC Rotator is operated in one of the two modes

- Rotation Mode
- Vector mode

The rotation mode rotates the input vector to a specified angle. The vectoring mode rotates the input vector to x axis while recording the angle required to make the rotation (i.e.) the direction of rotation is opposite in both the modes. The complex magnitude is computed using vectoring mode, the sine and cosine of the input angle is computed using rotation mode.

2.1 Rotation Mode

The CORDIC equations for rotation mode is (7)

$X_{i+1}=\cos\theta*[X_i - Y_i\, \delta_i\, 2^{-i}]$

$Y_{i+1}=\cos\theta*[Yi + X_i\, \delta_i\, 2^{-i}]$

$Z_{i+1}= Z_i - \delta_i * \tan^{-1}(2^{-i})$

Where $\delta_i = -1$ if $Z_i<0$, +1 otherwise

After 'n' Iterations,

$X_n=K*[X_0 \text{ Cos } Z_0 - Y_0 \text{ Sin } Z_0]$

$Y_n=K*[Y_0 \text{ Cos } Z_0 + X_0 \text{ Sin } Z_0]$

$Z_n = 0$

2.2 Vectoring Mode

The CORDIC equations for vectoring mode is (8)

$X_{i+1}=X_i - Y_i\, \delta_i\, 2^{-i}$

$Y_{i+1}=Yi + X_i\, \delta_i\, 2^{-i}$

$Z_{i+1}= Z_i - \delta_i * \tan^{-1}(2^{-i})$

Where $\delta_i = -1$ if $Y_i<0$, +1 otherwise

After 'n' Iterations (9)

$$X_n = K\left(\sqrt{X_0^2+Y_0^2}\right)$$

$Y_n = 0$

$$Z_n = Z_0 + \tan^{-1}\left(Y_0 \big/ X_0 \right)$$

3 Computation of Complex magnitude

The Aim of the algorithm is to calculate the magnitude of a complex number C= X + jY.

Magnitude of this complex number is given by $|C| = \sqrt{X^2 + Y^2}$.The Cordic rotator rotates the input vector to angle Z_i for aligning the result vector with the x axis (Figure 1). The result of the operation is a rotation angle and the scaled magnitude of the original vector

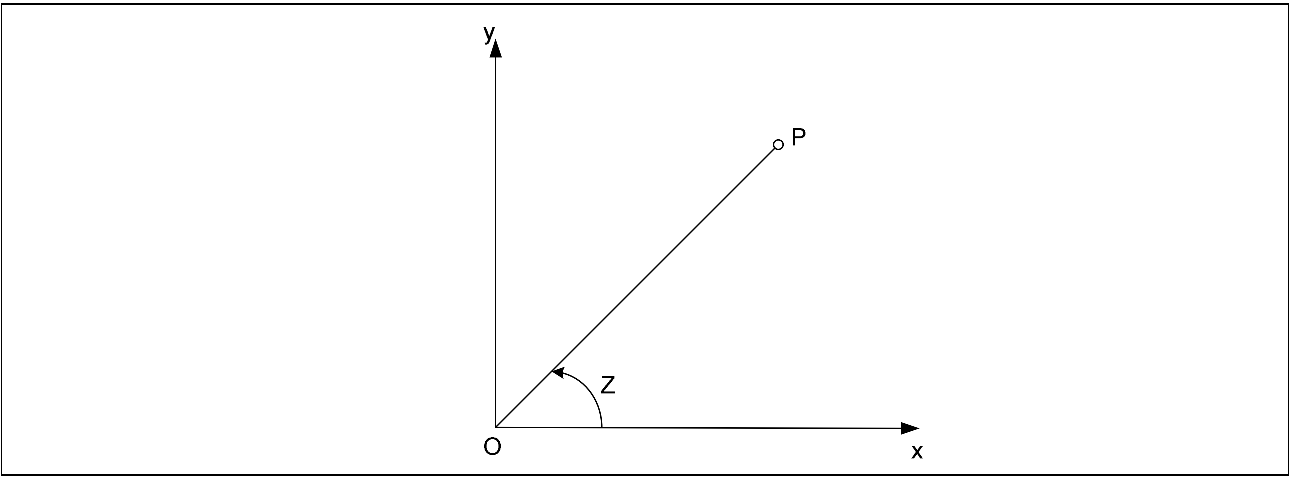


Figure 1 The Complex Plane

To extend the region of convergence greater than +/-90°, the phase is rotated by -90° if Y is positive and it is rotated by +90° if Y is negative. The CORDIC Rotation is done with successively smaller values of Z, starting with Z = 45°. The sign of Y decides to add or subtract the phase. In the rotation process C is multiplied with the Inverse CORDIC gain (K) of 0.6073.

3.1 Implementation

The Library is C-callable, hand coded assembly routine written for Infineon's XC164CS Microcontroller with MAC unit. Tasking tool is used for compilation. For the implementation we make the assumption that inputs are in 1Q15 format. The input data's are scaled down by 2 to avoid the overflow. The rotational gain K needs to be compensated at some stage (i.e.) compensation can be done before or after the iteration. In order to scale down the input further, the input X is multiplied by the inverse of gain before the iteration.

In Tasking tool chain the parameter transforms of the first four arguments of the function will be in R12 to R16. Using C166SV2 instruction set, the micro rotations according to (8) is given below as reference implementation. A fixed point number representation is used for the implementation. The registers R1, R2, R13 are assigned with X, Y, shift value respectively.

```
Label3:
   MOV          R12, #0h
   MOV          R3, R1
   MOV          R5, R2
   ASHR         R5, #0fh
   CMP          R5, #0
   JMPA         cc_NZ, Label1
```

```
; Micro rotation 1
;I= I+ (Q>>K)
  MOV           R7, R2
  ASHR          R7, R13
  MOV           MAH, R1
  CoADD         R12, R7
  CoSTORE       R1, MAS
; Q=Q-(I_tmp>>K)
  ASHR          R3,R13
  MOV           MAH,R2
  CoSUB         R12,R3
  CoSTORE       R2,MAS
  JMPA          cc_NZ,Label 2
; Micro rotation 2
Label1:
;I=I-(Q>>K)
   MOV          R7, R2
   ASHR         R7,R13
   MOV          MAH,R1
   CoSUB        R12,R7
   CoSTORE      R1,MAS
; Q=Q+ (I_tmp>>K)
   ASHR         R3, R13
   MOV          MAH, R2
   CoADD        R12, R3
   CoSTORE      R2,MAS
Label2:
    ADD         R13, #1h
    CMPD1       R6,#0h
    JMPR        cc_NZ, Label3
```

The number of iterations is fixed to 15 and the direction of rotation is depended on Y, therefore there is no need to record the degree of rotation (i.e.) the value of Z. This reduces the latency from n + 25 to n + 19 cycles, where 'n' is the number of iterations. In figure 2 the program flowchart of the full assembler program of the complex magnitude is shown.

3.2 Results and discussion

The result of complex magnitude is shown in Table 2. It is represented in 1Q15 format. For example complex input {30143, 21254} should be read as {0.91928, 0.64862}. This accuracy is obtained by keeping the number of iterations as 15. The error will be significantly reduced if this is implemented in a 32 bit processor with 31 iterations.

Table 2 Results of complex magnitude in 1Q15 format

Input {Real, imaginary}	**Output [Scaled down 0.5]**	**Expected Output [Scaled down 0.5]**	**Error [1Q15 Format]**
{12345,9728}	7856	7859	3
{-13254,-12543	9131	9124	7
{30123,21234}	18431	18427	4
{30143,21254}	18445	18442	3
{-30143,-21254}	18450	18442	8

Table 3 Cycle count and Code size

	Cycle count	**Code Size (Bytes)**
Store State	1	2
Read Input values and Initialization	11	22
Extension of region of convergence	8	16
CORDIC Rotation	n+19, where ‘n’ Number of Iterations	38
Restore state	1	2
Total	n+40 Cycles	80

This Algorithm should be better in code size compared to the traditional Taylor approximation series, but not good in terms of speed [8]. The speed is less due to the ‘n’ Number of iterations. Therefore this algorithm fits into an application which demands less code size.

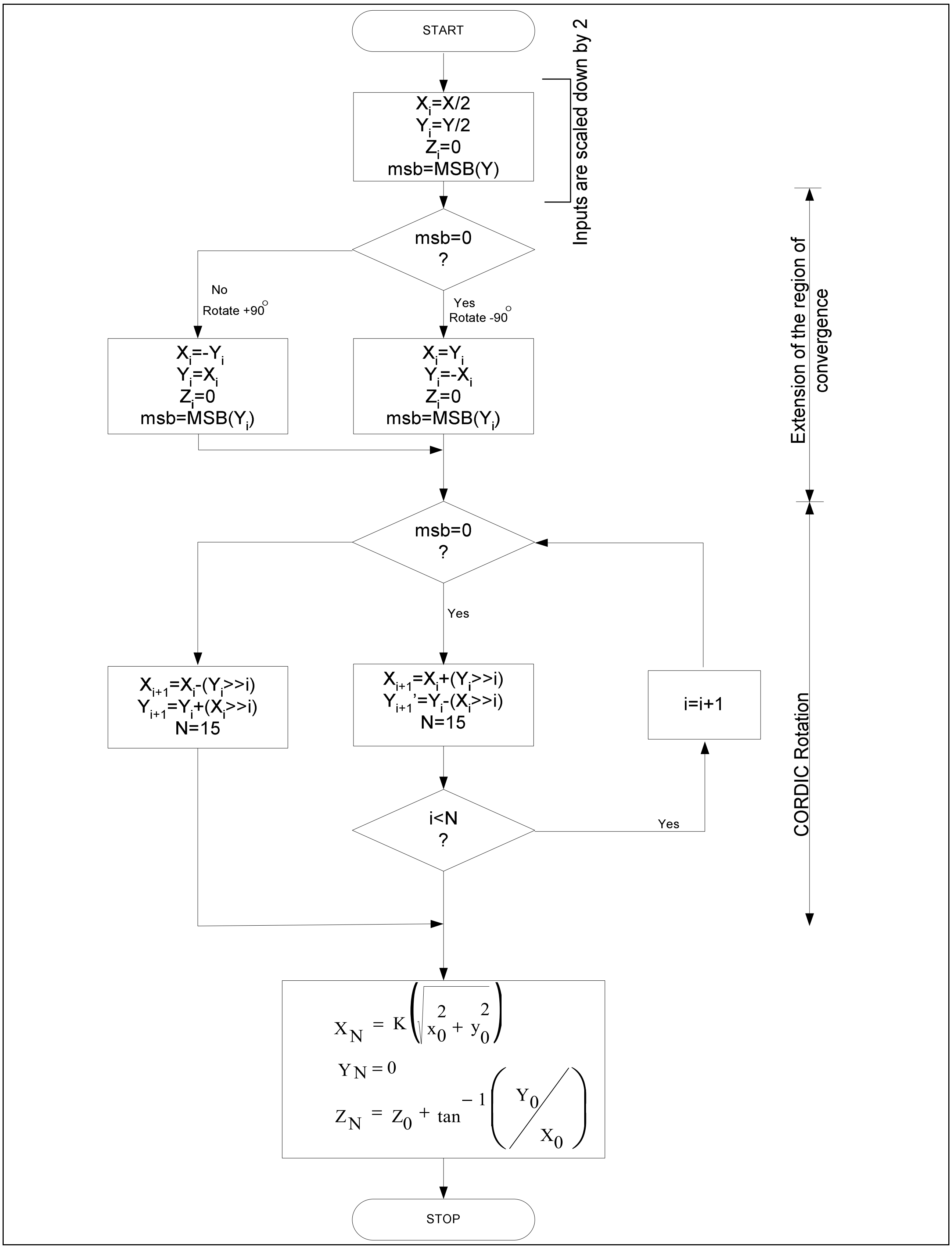


Figure 2 Flowchart for the implementation of complex magnitude using CORDIC

4 Computation of Sine and Cos for an input angle

Sine and cosine of the input angles is calculated using CORDIC. If the initial Y component of rotation transform is set to zero the rotation mode reduces to

X_{i+1}=K* X_i cos Z_i (10)

Y_{i+1}= K* X_i sin Z_i

Where, K is the CORDIC Gain. By setting initial X component to 0.60725 the rotation process produces an unscaled version of sine and cosine term. Since the rotational angle is limited to –90° and +90° additional rotation is required. This is done by exploiting the symmetry property of the sine wave. The values in other Quadrants are computed by using the relations, Sine (-Z) = -Sine (Z) and Sine (180-Z) = Sine (Z). The absolute value of the input is calculated. If the input is negative (III/IV Quadrant), then sign=1. If absolute value of the input is greater than 1/2 (II/III Quadrant), it is subtracted from 1. If sign=1, the result is negated to give the final sine result.

4.1 Implementation

The input vector Z contains the angle in radians between [-π,π] which is normalized between (-1, 1) in 1Q15 format (Z=Z rad/π). For example, 45°=π/4 is equivalent to Z = ¼ =0.25 (8192 in 1Q15 format. Denormalisation is done in the algorithm.

The algorithm is presented in a 'C' like pseudo code. Note that the Cosθ. constant for this algorithm is 0.60725. We also assume that the 12 values of $\tan^{-1}$ ($1/2^i$) are stored as a look up table in 4Q12 format.

Using C166SV2 instruction set, the micro rotations according to (7) is given below as reference implementation. A fixed point number representation is used for the implementation. The registers R1, R2, R11 and R13 are assigned with X, Y, Z, shift value respectively.

```
Label3:
   MOV          R12, #0h
   MOV          R3, R1
   MOV          R5, R2
   ASHR         R5, #0fh
   CMP          R5, #0
   JMPA         cc_NZ, Label1
; Micro rotation 1
; I=I+ (Q>>K)
  MOV           R7, R2
  ASHR          R7, R13
  MOV           MAH, R1
  CoADD         R12,R7
  CoSTORE       R1,MAS
; Q=Q-(I_tmp>>K)
  ASHR          R3, R13
  MOV           MAH, R2
  CoSUB         R12,R3
  CoSTORE       R2,MAS
 ; Z=Z+ atan(K[L])
  MOV           MAH, R11
  CoADD         R12,[R4+]
  CoSTORE       R11,MAS
  JMPA          cc_NZ, Label 2
; Micro rotation 2
Label1:
;I=I-(Q>>K)
   MOV          R7, R2
   ASHR         R7,R13
   MOV          MAH,R1
   CoSUB        R12,R7
   CoSTORE      R1,MAS
; Q=Q+ (I_tmp>>K)
   ASHR         R3, R13
   MOV          MAH, R2
   CoADD        R12, R3
   CoSTORE      R2,MAS
;Z=Z-atan(K[L])
   MOV          MAH,R11
   CoSUB        R12,[R4+]
   CoSTORE      R11,MAS
Label2:
    ADD         R13, #1h
    CMPD1       R6,#0h
    JMPR        cc_NZ, Label3
```

We cannot neglect the angle information as in complex magnitude, since the direction of rotation is dependent on Z Parameter. To denormalise, the input is multiplied by 4DBAh (4Q12 format). Therefore the lookup table value (tan-1 (1/2i)) has to be in 4Q12 format. Due to this the number of iterations is reduced to 12.

4.2 Pseudo code

Input vector Z is initialized to the desired angle, Y=0 and X=0.60725.The initialization of X specifies the constant 0.60725 which results from the Cos θ term.

```
 {
  short L;
  short I, Q;
  short Z;
  short tmp_I;
  short sign;
 Q=0;
  I=0.60725;

  Z=P*pi;    //denormalization of input
  If (Z<0) {
         sign =1;
  }
//If X is in III/IV Quadrant (Extension of region of convergence)
 If (abs (Z)>0.5)
  {
         Z=1-abs (Z);
  }
//CORDIC Rotation
 For (L = 0; L < 15; L++) {
     tmp_I = I;
 If (Z < 0.0) {
      I += Q >>L;
      Q -= tmp_I >>L;
      Z=Z+ tan$^{-1}$ (2$^{-L}$);  // value of tan$^{-1}$ (2$^{-L}$) is stored in lookup table
    } else {

      I -= Q >>L;
      Q += tmp_I >>L;
      Z=Z- tan$^{-1}$ (2$^{-L}$);
    }
 }
 if (sign==0)
 {
      I=I;
      Q=Q;
 } else {
      I=-I;
      Q=-Q;
 }
}
```

The Sine of the desired angle is now present in the variable I and the Cosine of the desired angle is in the variable Q. These outputs are within the integer range –32768 to +32767.

4.3 Results and Discussion

The result of complex magnitude is shown in Table 2. It is represented in 1Q15 format. For example complex input 8192 should be read as 0.25. This accuracy is obtained by keeping the iterations as 12. So we can expect an accurate result in a 32 bit processor.

Table 4 Results of Sine computation in 1Q15 format

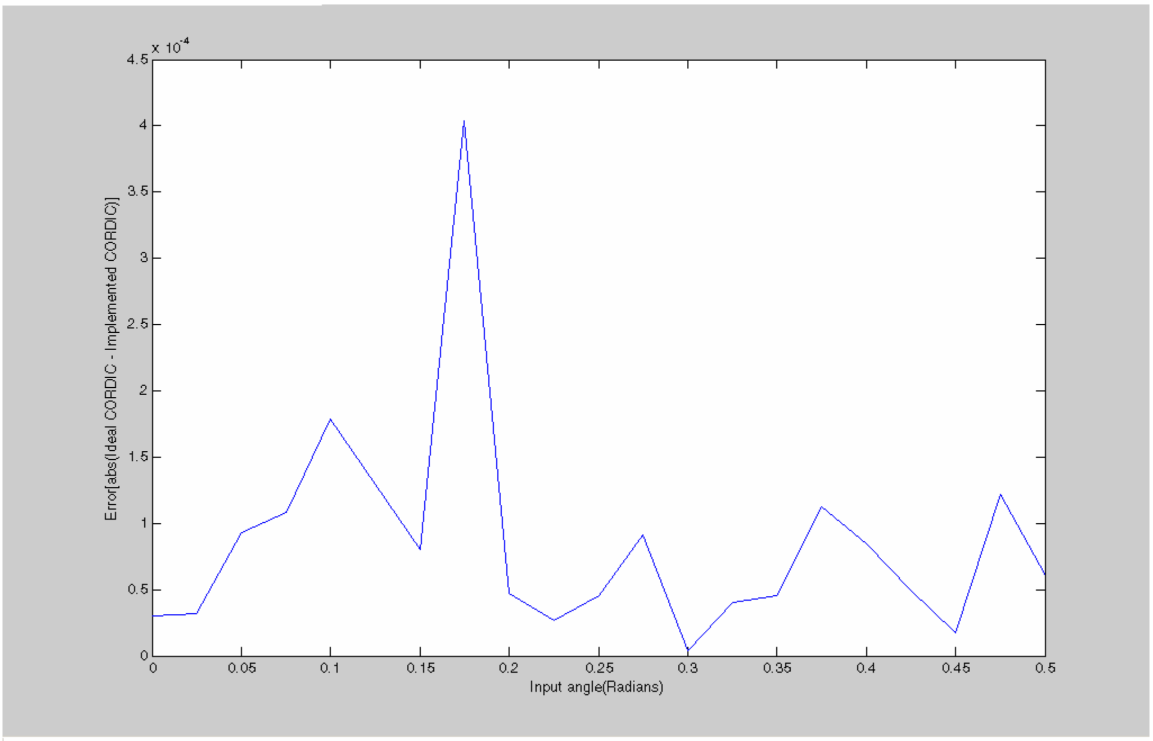
Input	Output	Expected Output	Error
0	1	0	1
8192	23169	23170	1
16384	32766	32767	1
24576	23172	23170	2
32767	1	0	1
-8192	-23169	-23171	2
-16384	-32766	-32768	2
-24576	-23172	-23171	1

Table 5 Cycle count and Code size

	Cycle count	Code Size (Bytes)
Store State	2	4
Read Input values and Initialization	4	8
Extension of region of convergence	10	20
CORDIC Rotation	n+25, where ‘n’ Number of Iterations	50
Restore state	2	4
Total	n+43 Cycles	86

5 Numerical Error in CORDIC

The error in CORDIC is split to different factors as approximation error and truncation error [1]. Theoretical realization of CORDIC has infinite iterations which produce the accurate result. But, the practical implementation of CORDIC has finite number of iterations. This is the cause for approximation error. In general CORDIC Algorithm produces one additional bit of accuracy for each iteration. The truncation error is due to the finite word length effect. For example, consider a fixed-point representation of *5* bits, with the lower order 3 bits after the binary point. If $x_i = 1.2345678$, then the approximate representation of this number is 01001. Hence $Q[x_i] = 0 * 2^1 + 1 * 2^0 + 0 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3} = 1 + 0.125 = 1.125$. Hence the quantization error due to finite word length is $E_i = 1.2345678 - 1.125 = 0.1095678 < 2^{-3}$ (0.125). Due to these errors the precision of CORDIC is affected. A Sine wave of 21 samples with input frequency 50 Hz and sampling frequency of 4000 Hz is generated using Ideal CORDIC and the implemented CORDIC. The magnitude of the difference between the Ideal CORDIC and the implemented CORDIC is shown in figure 3. The X axis represents the input angle which is normalized and the Y axis represents the approximation and the truncation error.



6 Conclusions

In this paper, fixed point software implementation of CORDIC has been presented. The accuracy has been discussed using the error and the efficiency is calculated using the cycle count and code size.

7 Acknowledgements

Thanks to Samuel Ginsberg, Richard for helping me to understand the concept. Thanks to Manoj palat, Raghunath lolur, Sonali nath for their valuable ideas and code review.

8 References

[1] Ray Andraka, A Survey of CORDIC algorithms for FPGA based computers

[2] Y. H. Hu, "The quantization effects of the CORDIC algorithm," *IEEE Trans. Signal Processing,* pp. 834-844, Apr. 1992.

[3] Sang Yoon Park and Nam Ik Cho, "fixed point error analysis of CORDIC processor based on the variance propagation," *IEEE Transactions on Circuits and Systems I-Fundamental Theory and Applicat*, vol. 51 no. 3 pp.573-584, Mar. 2004

[4] Samuel Ginsberg, "Compact and Efficient Generation of Trigonometric Functions using a CORDIC algorithm"

[5] http://www.dspguru.com/info/faqs/cordic.htm

[6] Infineon Technologies, C166S V2 User manual, 16-Bit Microcontroller V 1.7

[7] Guangyu Wang, C166S V2 Lib A DSP Librarary for C166S V2 Core, User’s Manual, V1.1, Sep 2002

[8] TriLib, A DSP Library for Tricore, User’s Manual, V1.2, Jan 2001