Tutorial: Software implementation of Trigonometric Functions using a CORDIC algorithm

Christober Rayappan, Infineon - December 23, 2006

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

This tutorial describes software implementation of the following fixed point trigonometric routines using the CORDIC Algorithm on 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.

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 Equations (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.
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 tan is the power of 2, where Θ is the rotational angle. The first seven steps of the set of rotations are shown in Table 1 below.

<table>
<thead>
<tr>
<th></th>
<th>θ</th>
<th>Tan θ (Decimal) = 2^i</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>45</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>26.565</td>
<td>0.49999</td>
</tr>
<tr>
<td>2</td>
<td>14.036</td>
<td>0.24899</td>
</tr>
<tr>
<td>3</td>
<td>7.125</td>
<td>0.12499</td>
</tr>
<tr>
<td>4</td>
<td>3.516</td>
<td>0.06249</td>
</tr>
<tr>
<td>5</td>
<td>1.784</td>
<td>0.03114</td>
</tr>
<tr>
<td>6</td>
<td>0.894</td>
<td>0.01362</td>
</tr>
</tbody>
</table>

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

\[
X_{i+1} = \cos \theta \left[ X_i - Y_i \tan \theta \right] \\
Y_{i+1} = \cos \theta \left[ Y_i + X_i \tan \theta \right]
\]

Equations 5 and 6

From Table 1 above 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}

\[
Z_{i+1} = Z_i - \delta_i \times \tan^i (2^i) \quad \text{Where } \delta_i \text{ determines the direction of rotation.}
\]

The CORDIC Rotator is operated in one of the two modes

1) Rotation Mode
2) 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.

**Rotation Mode.** The CORDIC equations for rotation mode is

\[
X_{i+1} = X_i \cos \theta - Y_i \sin \theta \\
Y_{i+1} = Y_i \cos \theta + X_i \sin \theta
\]

(Equation 1) (Equation 2)

These can be expressed as

\[
X_{i+1} = \cos \theta \left[ X_i - Y_i \tan \theta \right] \\
Y_{i+1} = \cos \theta \left[ Y_i + X_i \tan \theta \right]
\]

(Equation 3) (Equation 4)
After ‘n’ Iterations,

\[ X_n = K^n [X_0 \cos Z_0 - Y_0 \sin Z_0] \]
\[ Y_n = K^n [Y_0 \cos Z_0 + X_0 \sin Z_0] \]
\[ Z_n = \theta \]

(Equations 7)

**Vectoring Mode.** The CORDIC equations for vectoring mode is

\[ X_{i+1} = X_i - Y_i \delta_i 2^{-i} \]
\[ Y_{i+1} = Y_i + X_i \delta_i 2^{-i} \]
\[ Z_{i+1} = Z_i - \delta_i \tan^{-1}(2^{-i}) \]

Where \( \delta_i = +1 \) if \( Y_i < 0 \), otherwise

(Equations 8)

After ‘n’ Iterations

\[ X_n = K \left( \sqrt{X_0^2 + Y_0^2} \right) \]
\[ Y_n = 0 \]
\[ Z_n = Z_0 + \tan^{-1} \left( \frac{Y_0}{X_0} \right) \]

(Equations 9)

**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 \( Z \), for aligning the result vector with the x axis (Figure 1, below), the result of the operation is a rotation angle and the scaled magnitude of the original vector.

![Figure 1](image)
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.

**Implementation.** The Library is C-callable, hand coded assembly routine written for Infineon's XC164CS Microcontroller with MAC unit. A 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 the 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 Equation (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:
```assembly
    MOV     R12,#0h
    MOV     R3,R1
    MOV     R5,R2
    ASHR    R5,#0fh
    CMP     R5,#0
    JMPA    cc_NZ, Label1
```

; Micro rotation 1
```assembly
    ;I=I+(Q>>K)
    MOV     R7,R2
    ASHR    R7,R13
    MOV     MAH,R1
    CoADD   R12,R7
    CoSTORE R1,MAS
```

```assembly
    ;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:
```assembly
    ;I=I-(Q>>K)
    MOV     R7, R2
    ASHR     R7,R13
    MOV     MAH,R1
    CoSUB      R12,R7
    CoSTORE R1,MAS
```

```assembly
    ; Q=Q+ (I_tmp>>K)
    ASHR     R3, R13
```

```assembly
```
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 (in the attached PDF file), the program flowchart of the full assembler program of the complex magnitude is shown.

<table>
<thead>
<tr>
<th>Input (Real, imaginary) [IQ15 Format]</th>
<th>Output [Scaled down 0.5]</th>
<th>Expected Output [Scaled down 0.5]</th>
<th>Error [IQ15 Format]</th>
</tr>
</thead>
<tbody>
<tr>
<td>(12345, 9728)</td>
<td>7856</td>
<td>7859</td>
<td>3</td>
</tr>
<tr>
<td>(.13254, -12543)</td>
<td>9131</td>
<td>9124</td>
<td>7</td>
</tr>
<tr>
<td>(30123, 21234)</td>
<td>18431</td>
<td>18427</td>
<td>4</td>
</tr>
<tr>
<td>(30143, 21254)</td>
<td>18445</td>
<td>18442</td>
<td>3</td>
</tr>
<tr>
<td>(-30143, -21254)</td>
<td>18459</td>
<td>18442</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2 Output

<table>
<thead>
<tr>
<th>Store state</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store inputs and initialization</td>
<td>1</td>
</tr>
<tr>
<td>Extension of region of convergence</td>
<td>8</td>
</tr>
<tr>
<td>CORDIC Rotation</td>
<td>n+19, where ‘n’ Number of Iterations</td>
</tr>
<tr>
<td>Restore state</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>n+40 Cycles</td>
</tr>
</tbody>
</table>

Table 3 Cycle count

<table>
<thead>
<tr>
<th>Store state</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read input values and initialization</td>
<td>22</td>
</tr>
<tr>
<td>Extension of region of convergence</td>
<td>16</td>
</tr>
<tr>
<td>CORDIC Rotation</td>
<td>38</td>
</tr>
<tr>
<td>Restore state</td>
<td>2</td>
</tr>
<tr>
<td>Total</td>
<td>80 Bytes</td>
</tr>
</tbody>
</table>

Table 4 Code Size (Bytes)

Computation of Sine and Cosine 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:

\[
\begin{align*}
X_{i+1} &= K \cdot X_i \cos Z_i \\
Y_{i+1} &= K \cdot X_i \sin Z_i
\end{align*}
\]

(Equations 10)

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.

**Implementation.** The input vector Z contains the angle in radians between [-,] which is normalized between (-1, 1) in 1Q15 format (Z=Z rad/). For example, 45° = /4 is equivalent to Z =1/4 =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 Cos Θ constant for this algorithm is 0.60725. We also assume that the 12 values of tan-1 (1/2i) are stored as a look up table in 4Q12 format.

Using C166SV2 instruction set, the micro rotations according to Equation (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.

**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.

```cpp
{
    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
        }
    }
}
```
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.*

<table>
<thead>
<tr>
<th>i</th>
<th>Output</th>
<th>Expected Output</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8192</td>
<td>23169</td>
<td>23170</td>
<td>1</td>
</tr>
<tr>
<td>16384</td>
<td>32766</td>
<td>32767</td>
<td>1</td>
</tr>
<tr>
<td>24576</td>
<td>23172</td>
<td>23170</td>
<td>2</td>
</tr>
<tr>
<td>32767</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-8192</td>
<td>-23169</td>
<td>-23171</td>
<td>2</td>
</tr>
<tr>
<td>-16384</td>
<td>-32766</td>
<td>-32768</td>
<td>2</td>
</tr>
<tr>
<td>-24576</td>
<td>-23172</td>
<td>-23171</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5 Output

<table>
<thead>
<tr>
<th>Store state</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Input values and Initialization</td>
<td>4</td>
</tr>
<tr>
<td>Extension of region of convergence</td>
<td>10</td>
</tr>
<tr>
<td>CORDIC Rotation</td>
<td>( \pi + 2 \alpha ), where ( \alpha ) is Number of Iterations</td>
</tr>
<tr>
<td>Restore state</td>
<td>2</td>
</tr>
<tr>
<td>Total</td>
<td>( \pi + 43 ) Cycles</td>
</tr>
</tbody>
</table>

Table 6 Cycle count

<table>
<thead>
<tr>
<th>Store state</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Input values and Initialization</td>
<td>8</td>
</tr>
<tr>
<td>Extension of region of convergence</td>
<td>20</td>
</tr>
<tr>
<td>CORDIC Rotation</td>
<td>50</td>
</tr>
<tr>
<td>Restore state</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>86 Bytes</td>
</tr>
</tbody>
</table>

Table 7 Code Size (Bytes)

**Numerical Error in CORDIC**

The error in CORDIC is split to different factors as approximation error and truncation error (Equation 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 \( Q[x_i] = 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 1 + 0.125 = 1.125 \).

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.

![Graph showing the magnitude of the difference between Ideal CORDIC and implemented CORDIC](image)

The magnitude of the difference between the Ideal CORDIC and the implemented CORDIC is shown in Figure 3 above. The X axis represents the input angle which is normalized and the Y axis represents the approximation and the truncation error.

Acknowledgements: Thanks to Samuel Ginsberg and Richard Armstrong for helping me to understand the concept. Thanks to Manoj Palat, Raghunath Iolur, and Sonali Nath for their valuable ideas and code review.

Christopher Rayappan is a software engineer at Infineon Technologies, with a particular focus on the design and development of DSP Algorithms in embedded processors and FPGAs, and finite word length effects in signal processing. He is currently involved in the development of DSP libraries for Infineon 16-bit microcontrollers.

References

http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1275604

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