CMSC-411 Final Report: Cordic Algorithm in ARM

Stephanie Tam, Evan Andre, Ben Hazlett

**Total Computer Cycles**

186 cycles to calculate sin(x) and cos(x)

**Estimated CPI**   
CPI = # of instructions

**Estimated Total Processing Time**

Assuming a system clock of the following:

* 32kHz:
* 1MHz:
* 1GHz

**Description of Implemented Algorithms**   
To calculate *sin(x)* and *cos(x)* we used the CORDIC algorithm.

**The Cordic Algorithm for Sine and Cosine**

Sine and Cosine are estimated through this algorithm using only shifts, adds, and subtracts. The key for this is the convergence on a specific angle by either adding to our target angle when our target angle is negative and subtracting from our target angle when our target angle is positive. The values for sine and cosine are initialized to 0 and 0.6072529350 respectively. To set up our table of angles we took the arctan(2i) where i started at 0 and counted up to 11. Then from there the new cosine and new sine values are calculated as such:

If current angle < 0,

current angle += angleTable[i]

new cosine = current cosine + (current sine >> i)

new Sine = current sine  - (current cosine >> i)

current cosine = new cosine

current sine  = new sine

If current angle >= 0,

current angle -= angleTable[i]

new cosine = current cosine - (current sine >> i)

new sine   = current sine  + (current cosine >> i)

current cosine = new Cosine

current sine = new Sine

This was then iterate for the whole angleTable.

After a twelve iterations the current cosine and current sine values will result in our estimate for our cosine and sine values for that angle. One important method we used when implementing this algorithm in ARM is

In implementing division in ARM LEGv8,

**Sample Input & Output**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Angle, z** | **Cos(z)**  **Hex** | **Sin(z)**  **Hex** | **Cos(z)**  **Approx.** | **Sin(z)**  **Approx.** | **Actual**  **Cos(z)** | **Actual**  **Sin(z)** |
| **5.2954** | **0000.fee3** | **0000.17be** | **0.9956512** | **0.0927429** | **0.9957321** | **0.0922906** |
| **80.1** | **0000.2c0a** | **0000.fc2e** | **0.1720275** | **0.9850769** | **0.1719291** | **0.9851093** |
| **24.23** | **0000.e989** | **0000.68d4** | **0.9122467** | **0.4094848** | **0.9119053** | **0.4104005** |
| **48.52** | **0000.a96f** | **0000.bfe9** | **0.6618499** | **0.7496490** | **0.6623585** | **0.7491869** |
| **10.001** | **0000.fc17** | **0000.2c88** | **0.9847259** | **0.1739501** | **0.9848047** | **0.1736653** |
| **70.86159** | **0000.53f1** | **0000.f1d5** | **0.3278961** | **0.9446563** | **0.3278513** | **0.9447293** |
| **30.0** | **0000.ddbd** | **0000.7ff0** | **0.8661651** | **0.4997558** | **0.8660254** | **0.5** |
| **86.5** | **.0b043615** | **.efe84526** | **0.0430330** | **0.9371379** | **0.0610485** | **0.9981347** |
| **90** | **00000000** | **00000001** | **0** | **1** | **0** | **1** |
| **0** | **00000001** | **00000000** | **1** | **0** | **1** | **0** |
| **4.3667** | **.efb4bda** | **.0ec37009** | **0.9363511** | **0.0576696** | **0.9970970** | **0.0761407** |
| **88.667** | **.021bd47a** | **.f0269e12** | **0.0082371** | **0.9380832** | **0.0232631** | **0.9997293** |
| **45** | **0000.b4f6** | **0000.b513** | **0.7068786** | **0.7073211** | **0.7071067** | **0.7071067** |
| **60** | **0000.7ff0** | **0000.ddbd** | **0.4997558** | **0.8661651** | **0.5** | **0.8660254** |