



Sahin Okur and Eyal Cohen

ABSTRACT

Trigonometric functions are commonly used in real-time control applications, particularly within the inner loops of control algorithms, where speed and accuracy is essential. The performance of trigonometric functions is a key careabout for designers of these systems as it can have a significant impact on the overall performance of the system. Until recently, trigonometric functions based on lookup tables were considered faster than the polynomial-based methods; however, with the inclusion of floating-point units (FPUs) and faster clock speeds, polynomial-based approximations have gained favor. TI has developed C functions of the most commonly used trigonometric functions using these polynomial-based methods and has optimized them for TI's Arm®-based microcontrollers (MCUs) and microprocessors (MPUs). This application note surveys the trigonometric functions that are available today and shares the optimization techniques used in these functions, as well as the results of our optimization efforts.

The TI-optimized trigonometric functions presented in this document can be found in MCU+ SDK v8.5 and later.

Table of Contents

1 Introduction	2
2 Trigonometric Optimizations	2
2.1 Lookup Table-Based Approximation.....	2
2.2 Polynomial Approximation.....	2
3 Trig Library Benchmarks	7
3.1 C Math.h Library.....	7
3.2 Arm “Fast Math Functions” in CMSIS.....	7
3.3 TI Arm Trig Library.....	8
3.4 Table of Results.....	8
4 Optimizations	9
4.1 Branch Prediction.....	9
4.2 Floating-Point Single-Precision Instructions.....	10
4.3 Memory Placement.....	11
4.4 Compiler.....	11
Revision History	12

List of Figures

Figure 2-1. Plot of Sine and Cosine Over the Range.....	3
Figure 2-2. Mapping of the Unit Circle for $\sin(x)$ for $0 \leq x \leq 2\pi$	4
Figure 2-3. Plot of Sine and Cosine Over the Range.....	4
Figure 2-4. Plot of $\arctan2(y,x)$	6
Figure 4-1. Condition Code Suffixes and Related Flags.....	9
Figure 4-2. Floating-Point Single-Precision Instructions.....	10
Figure 4-3. Memory Placement.....	11

List of Tables

Table 2-1. Sine.....	5
Table 2-2. Cosine.....	5
Table 2-3. Coefficients for the Arctangent.....	7
Table 3-1. Table of Results - Arm Cortex®-R5F.....	8

Trademarks

Arm® and Cortex® are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All trademarks are the property of their respective owners.

1 Introduction

Trigonometric functions are commonly used in real-time control applications, particularly within the inner loops of control algorithms, where speed and accuracy is essential. The performance of trigonometric functions is a key careabout for designers of these systems as it can have a significant impact on the overall performance of the system. Until recently, trigonometric functions based on lookup tables were considered faster than the polynomial-based methods; however, with the inclusion of floating-point units (FPUs) and faster CPU clock speeds, polynomial-based approximations are gaining favor.

2 Trigonometric Optimizations

The most commonly used trigonometric functions are sine, cosine, arcsine, arccosine, arctangent, and atan2. Trigonometric optimization techniques for these functions fall into two categories:

- Lookup table-based approximations
- Polynomial approximations

These techniques are described in the following sections and the polynomial-based optimization techniques used by the TI Arm® Trigonometric Library are discussed.

2.1 Lookup Table-Based Approximation

Up until recently, lookup table-based approximations were considered faster than polynomial-based approximations. Lookup table-based approximations pre-compute N values of the target function and then index them into the table to select the two closest points and perform an interpolation between them. When using lookup table-based approximations, the accuracy can be adjusted by changing either the size of the lookup or the order of the interpolation function. However, this comes with the tradeoff of more memory usage and longer function times. This is the technique used by the Fast Math Functions in the Arm CMSIS Library where they use a table size of 512 entries to cover 0-2PI and then do a linear interpolation between the closest values.

2.2 Polynomial Approximation

With the inclusion of floating-point co-processors and faster clock speeds, the polynomial-based methods are now quite fast and have the added benefit of not requiring any table storage. The TI Arm Trig Library uses this method.

The first step in using polynomial approximations is to reduce the range of the input. The bigger the range needed to approximate, the higher order polynomial needed to achieve a specified level of accuracy, which means more computations and slower function performance.

The second step is to compute the approximation for the reduced range and then finally restore the range depending on which quadrant the input was originally in by using trigonometric identities.

There are a number of methods of finding the best polynomials to achieve the lowest error with the fewest terms, but the most commonly used is the minimax approximation algorithm and one version in particular: the Remez algorithm. The Remez algorithm is used to find a polynomial with the least maximum error. This polynomial is called a minimax polynomial. The minimax polynomial is what is used in our optimized trigonometric functions as it is ideal for control applications where worst-case performance is a key care-about.

Note

To generate the coefficients for the minimax polynomials, the Sollya software tool is used: <https://www.sollya.org/>.

2.2.1 Optimizing Sine and Cosine

This section shows how the above techniques are applied to optimize the computations for a Sin + Cos function. Sin + Cos functions are commonly used in the transforms for control algorithms where both sin and cos are needed at the same time.

The first step in using polynomial approximations is to range reduce the input so that the segment of the function that needs to be modeled is smaller. The most common range reduction for sin/cos computation is to reduce the input to the range.

$$-\frac{\pi}{2} \leq x \leq \frac{\pi}{2} \tag{1}$$

Then, compute the approximation in this region and then adjust the sign depending on which quadrant the angle was in originally. Using the Chebyshev polynomials and the fact that $\sin(x)$ is an odd function and $\cos(x)$ is an even function, you will see the following:

$$\sin(x) \approx C_1X + C_3X^3 + C_5X^5 + \dots \tag{2}$$

$$\cos(x) \approx C_0 + C_2X^2 + C_4X^4 + \dots \tag{3}$$

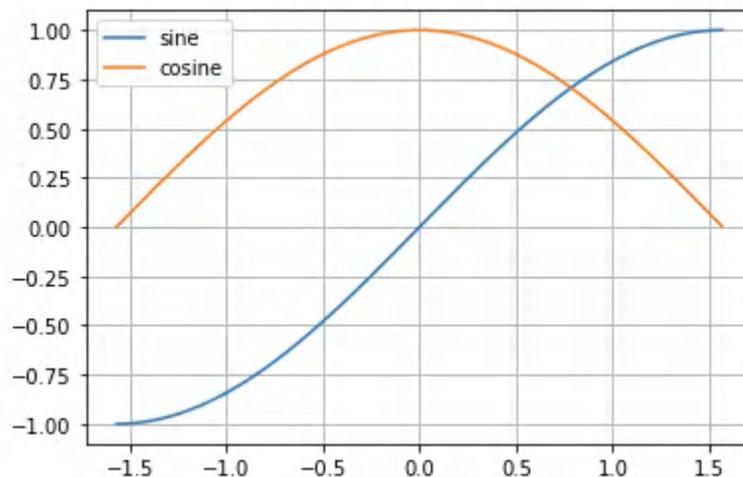


Figure 2-1. Plot of Sine and Cosine Over the Range

If you assume that the input to the functions are limited to $[0:2\pi]$, you need to map the input value to the range $-\frac{\pi}{2} \leq x \leq \frac{\pi}{2}$ before implementing the approximation. This can be done with a couple simple comparisons:

$$\text{if } x > \frac{3\pi}{2}, \text{ then } x = x - 2\pi \tag{4}$$

$$\text{else if } \frac{\pi}{2} < x < \frac{3\pi}{2}, \text{ then } x = \pi - x, \text{ and } \cos(x) = -\text{approx}(\cos(x)) \tag{5}$$

A further range reduction technique can be used to limit the input to $-\frac{\pi}{4} \leq x \leq \frac{\pi}{4}$ using the trigonometric identities:

$$\sin\left(\frac{\pi}{2} - \theta\right) = \cos(\theta) \tag{6}$$

$$\cos\left(\frac{\pi}{2} - \theta\right) = \sin(\theta) \tag{7}$$

Which yields the following mapping for $\sin(x)$ and a similar one for $\cos(x)$:

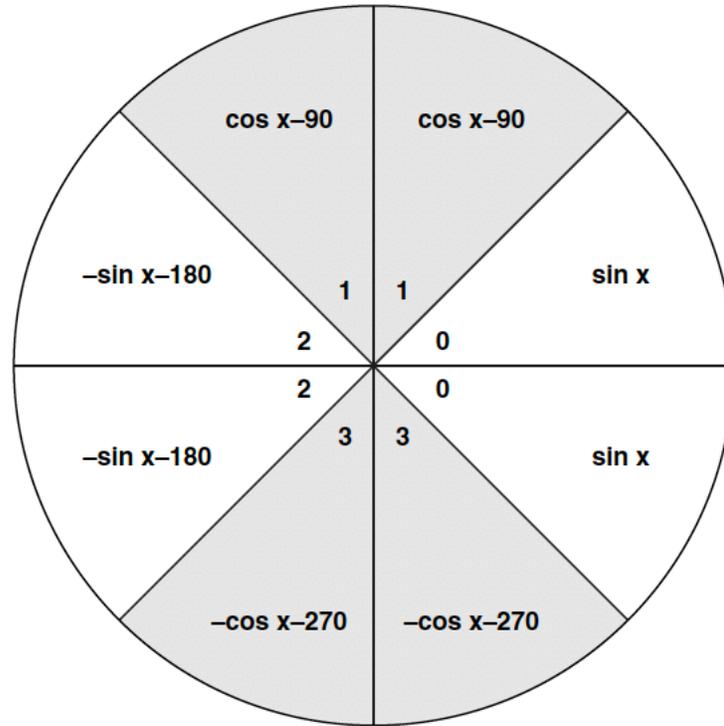


Figure 2-2. Mapping of the Unit Circle for $\sin(x)$ for $0 \leq x \leq 2\pi$

Since you only have to model the sine cosine from $-\frac{\pi}{4} \leq x \leq \frac{\pi}{4}$, you need fewer coefficients to achieve higher accuracy.

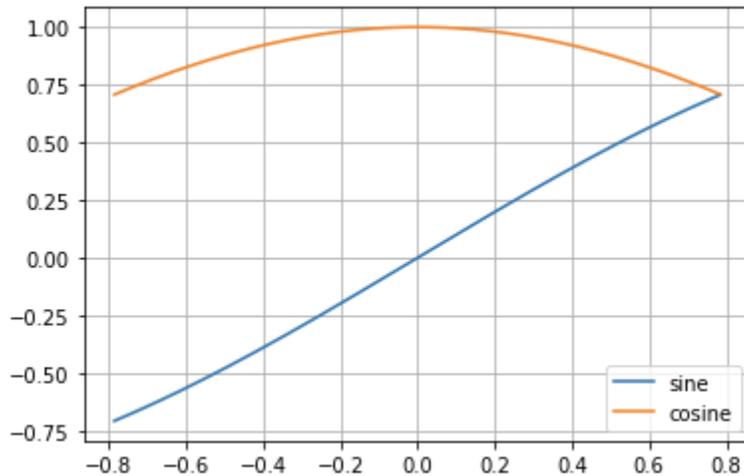


Figure 2-3. Plot of Sine and Cosine Over the Range

2.2.1.1 Sine Cosine Polynomials From Sollya

Table 2-1 and Table 2-2 show the coefficients for the sine and cosine approximation obtained from the Sollya program. The table shows the error expected given the range reduction and order of the polynomial. If you are trying to achieve full single precision floating-point accuracy, then you need to get to $\sim 1e-7$.

Table 2-1. Sine

Range	Number of Terms	Absolute Error	Polynomial
$-\pi/2 : \pi/2$	3	1.00E-04	$x * (0.999891757965087890625 + x^2 * (-0.165960013866424560546875 + x^2 * 7.602870464324951171875e-3))$ (8)
$-\pi/2 : \pi/2$	4	6.00E-07	$x * (0.999996483325958251953125 + x^2 * (-0.166647970676422119140625 + x^2 * (8.306086063385009765625e-3 + x^2 * (-1.83582305908203125e-4))))$ (9)
$-\pi/2 : \pi/2$	5	6.00E-09	$x * (1 + x^2 * (-0.1666665971279144287109375 + x^2 * (8.333069272339344024658203125e-3 + x^2 * (-1.98097783140838146209716796875e-4 + x^2 * 2.6061034077429212629795074462890625e-6))))$ (10)
$-\pi/4 : \pi/4$	2	1.50E-04	$x * (0.99903142452239990234375 + x^2 * (-0.16034401953220367431640625))$ (11)
$-\pi/4 : \pi/4$	3	5.60E-07	$x * (0.9999949932098388671875 + x^2 * (-0.166601598262786865234375 + x^2 * 8.12153331935405731201171875e-3))$ (12)
$-\pi/4 : \pi/4$	4	1.80E-09	$x * (1 + x^2 * (-0.166666507720947265625 + x^2 * (8.331983350217342376708984375e-3 + x^2 * (-1.94961365195922553539276123046875e-4))))$ (13)
$-\pi/4 : \pi/4$	5	6.00E-11	$x * (1 + x^2 * (-0.16666667163372039794921875 + x^2 * (8.33337195217609405517578125e-3 + x^2 * (-1.98499110410921275615692138671875e-4 + x^2 * 2.800547008519060909748077392578125e-6))))$ (14)

Table 2-2. Cosine

Range	Number of Terms	Absolute Error	Polynomial
$-\pi/2 : \pi/2$	3	6.00E-04	$0.9994032382965087890625 + x^2 * (-0.495580852031707763671875 + x^2 * 3.679168224334716796875e-2)$ (15)
$-\pi/2 : \pi/2$	4	6.70E-06	$0.99999332427978515625 + x^2 * (-0.4999125301837921142578125 + x^2 * (4.1487820446491241455078125e-2 + x^2 * (-1.27122621051967144012451171875e-3)))$ (16)
$-\pi/2 : \pi/2$	5	6.00E-08	$0.999999940395355224609375 + x^2 * (-0.499998986721038818359375 + x^2 * (4.1663490235805511474609375e-2 + x^2 * (-1.385320327244699001312255859375e-3 + x^2 * 2.31450176215730607509613037109375e-5)))$ (17)

Table 2-2. Cosine (continued)

Range	Number of Terms	Absolute Error	Polynomial
$-\pi/4 : \pi/4$	3	1.00E-05	$0.999990046024322509765625 + x^2 * (-0.4997082054615020751953125 + x^2 * 4.03986163437366485595703125e-2)$ (18)
$-\pi/4 : \pi/4$	4	3.30E-08	$1 + x^2 * (-0.49999892711639404296875 + x^2 * (4.16561998426914215087890625e-2 + x^2 * (-1.35968066751956939697265625e-3)))$ (19)
$-\pi/4 : \pi/4$	5	1.00E-10	$1 + x^2 * (-0.5 + x^2 * (4.16666455566883087158203125e-2 + x^2 * (-1.388731296174228191375732421875e-3 + x^2 * 2.4432971258647739887237548828125e-5)))$ (20)

2.2.2 Optimizing Arctangent and Arctangent2

The arctangent function, and in particular, the arctangent2 function is a critical function in control applications. For example, in motor control, there may be sensors used to get the x and y position of a motor. Then, the application needs to translate that into an angular value. A standard arctan function would be called as $\arctan(y/x)$, but this loses the quadrant information as Q1 and Q3 are both positive and Q2 and Q4 are both negative. The arctan2 function accepts both the x and the y input to return a value in the full $-\pi$ to π range versus the arctan function that returns values only in Q1 or Q4, $-\pi/2$ to $\pi/2$.

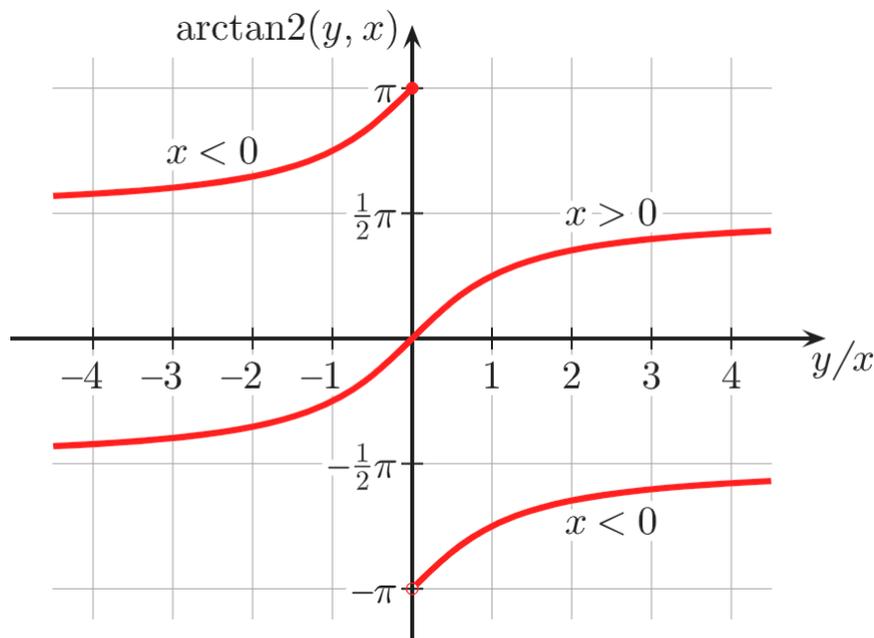


Figure 2-4. Plot of $\arctan2(y,x)$

As the input to $\arctan(z)$ can be any number from $-\infty$ to ∞ , use trig identities to reduce the range and get an approximation function that is relatively low complexity.

Start with:

- $\arctan(x) = \pi/2 - \arctan(1/x)$ (1a)
- $\arctan(x) = -\arctan(-x)$ (2a)

These identities allow you to restrict the approximation range to $\text{abs}(x) \leq 1$.

The next step is to find a simple polynomial that approximates arctan in the range 0-1, or -1 to 1. Using sollya, try some different polynomial lengths to get the approximation error. This is shown in [Table 2-3](#). You can see that the error is not falling that fast as a function of the number of terms in the polynomial, so even at 6 terms you are still at 3.4e-6 in the range (-1,1). Also note that arctan is an odd function where all the terms are odd powers.

2.2.2.1 Arctangent Polynomials

[Table 2-3](#) shows the coefficients for the arctangent approximation obtained from the Sollya program. The table shows the error expected given the range reduction and order of the polynomial.

Table 2-3. Coefficients for the Arctangent

Range	Terms	Abs err	Polynomial
-1 : 1	4	8.00E-05	$x * (0.99921381473541259765625 + x^2 * (-0.321175038814544677734375 + x^2 * (0.146264731884002685546875 + x^2 * (-3.8986742496490478515625e-2))))$ (21)
-1 : 1	5	2.30E-05	$x * (0.999970018863677978515625 + x^2 * (-0.3317006528377532958984375 + x^2 * (0.1852150261402130126953125 + x^2 * (-9.1925732791423797607421875e-2 + x^2 * 2.386303804814815521240234375e-2))))$ (22)
-1 : 1	6	3.40E-06	$x * (0.999995648860931396484375 + x^2 * (-0.3329949676990509033203125 + x^2 * (0.19563795626163482666015625 + x^2 * (-0.121243648231029510498046875 + x^2 * (5.7481847703456878662109375e-2 + x^2 * (-1.3482107780873775482177734375e-2))))))$ (23)
tan(pi/12)	3	2.00E-07	$x * (0.999994814395904541015625 + x^2 * (-0.3327477872371673583984375 + x^2 * 0.18327605724334716796875))$ (24)
tan(pi/12)	4	3.00E-09	$x * (0.999999940395355224609375 + x^2 * (-0.333319008350372314453125 + x^2 * (0.19920165836811065673828125 + x^2 * (-0.12685041129589080810546875))))$ (25)
tan(pi/12)	5	8.70E-11	$x * (1 + x^2 * (-0.333333194255828857421875 + x^2 * (0.19998063147068023681640625 + x^2 * (-0.14202083647251129150390625 + x^2 * 9.6703059971332550048828125e-2))))$ (26)

3 Trig Library Benchmarks

3.1 C Math.h Library

Math.h includes the standard functions for single-precision floating-point `sinf()`, `cosf()`, `atanf()`, and `atan2f()`, as well as the double-precision `sin()`, `cos()`, `atan()`, and `atan2()`. These functions use polynomial-based approximation methods and provide high accuracy with no input limitations, but at the cost of more cycles. These functions are benchmarked and the results are shared at the end of this section.

3.2 Arm “Fast Math Functions” in CMSIS

Arm provides a library titled *FastMathFunctions* that contains single-precision floating-point `sin()` and `cos()` functions. These functions use lookup table-based methods to approximate the functions with lookup tables of size 2 KB. Arm also provides a `sincos` function that computes both the `sin()` and `cos()` simultaneously in *ControllerFunctions* as well. These functions were benchmarked and the results are shared at the end of this section.

3.3 TI Arm Trig Library

The TI Arm Trig library provides the single-precision floating-point functions `ti_arm_sin()`, `ti_arm_cos()`, `ti_arm_sincos()`, `ti_arm_asin()`, `ti_arm_acos()`, `ti_arm_atan()`, and `ti_arm_atan2()`. These functions use polynomial approximation-based methods.

This library can be found in MCU+ SDK.

3.4 Table of Results

Hardware

AM243x LaunchPad

Software

- TI Arm Clang Compiler v2.0.0.STS
- MCU+ SDK for AM243x v8.2.0.31

Table 3-1. Table of Results - Arm Cortex®-R5F

Trig Function	Library	C Function	Input Range [Rad]	Max Error	Max Cycles	Avg Cycles	Approximation Type
Sine	C <Math.h>	<code>sinf()</code>	Any	0.0000000296	179	150	Polynomial
	CMSIS	<code>arm_sin_f32()</code>	Any	0.0000181917	48	48	Lookup table (2 KB)
	TI Arm Trig	<code>ti_arm_sin()</code>	0:2 π	0.0000007225	29	29	Polynomial
Cosine	C <Math.h>	<code>cosf()</code>	Any	0.0000000297	179	150	Polynomial
	CMSIS	<code>arm_cos_f32()</code>	Any	0.0000183477	50	50	Lookup table
	TI Arm Trig	<code>ti_arm_cos()</code>	0:2 π	0.0000002863	37	37	Polynomial
Sine + Cosine	C <Math.h>	NA	-	-	-	-	-
	CMSIS	<code>arm_sin_cos_f32()</code>	Any	0.0000006100	83	83	Lookup table
	TI Arm Trig	<code>ti_arm_sincos()</code>	0:2 π	0.0000001925	54	54	Polynomial
Arcsine	C <Math.h>	<code>asinf()</code>	Any	0.0000000590	213	132	Polynomial
	CMSIS	NA	-	-	-	-	-
	TI Arm Trig	<code>ti_arm_asin()</code>	Any	0.0000003428	59	59	Polynomial
Arccosine	C <Math.h>	<code>acosf()</code>	Any	0.0000001792	128	87	Polynomial
	CMSIS	NA	-	-	-	-	-
	TI Arm Trig	<code>ti_arm_acos()</code>	Any	0.0000004295	64	64	Polynomial
Arctangent	C <Math.h>	<code>atanf()</code>	Any	0.0000001748	128	87	Polynomial
	CMSIS	NA	-	-	-	-	-
	TI Arm Trig	<code>ti_arm_atan</code>	Any	0.0000001748	64	64	Polynomial
Arctangent2	C <Math.h>	<code>atan2f()</code>	Any	0.0000002021	222	148	Polynomial
	CMSIS	NA	-	-	-	-	-
	TI Arm Trig	<code>ti_arm_atan2</code>	Any	0.0000002957	59	49	Polynomial

4 Optimizations

4.1 Branch Prediction

Using branches in functions creates unpredictability in the exact cycle count as the branch predictor may not predict correctly and any missed predictions cost approximately 8 cycles/miss. Arm provides conditional instructions that can be used in place of branch statements, ensuring that the functions always execute in the same number of cycles. Figure 4-1 shows that the conditional codes that can be appended to instructions.

Table 5-1 Condition code suffixes and related flags

Suffix	Flags	Meaning
EQ	Z set	Equal
NE	Z clear	Not equal
CS or HS	C set	Higher or same (unsigned >=)
CC or LO	C clear	Lower (unsigned <)
MI	N set	Negative
PL	N clear	Positive or zero
VS	V set	Overflow
VC	V clear	No overflow
HI	C set and Z clear	Higher (unsigned >)
LS	C clear or Z set	Lower or same (unsigned <=)
GE	N and V the same	Signed >=
LT	N and V differ	Signed <
GT	Z clear, N and V the same	Signed >
LE	Z set, N and V differ	Signed <=
AL	Any	Always. This suffix is normally omitted.

Figure 4-1. Condition Code Suffixes and Related Flags

The main reason for creating the *.asm* versions of the trigonometric functions was to remove branches inserted by the compiler and replace with conditional instructions instead. This had the effect of reducing the max cycles due to incorrect branch predictions. This reduction was enabled by replacing branch instructions with conditional operations. The delta between the max and the average could not be completely removed as the algorithm contains some divide instructions in the range reduction code which are conditionally implemented depending on the input values.

Note

The TI Arm Clang compiler performs these assembly optimizations automatically when compiler optimization is enabled, therefore these assembly versions have been replaced with their C-equivalent starting with MCU+ SDK v8.5.

4.2 Floating-Point Single-Precision Instructions

Time-expensive instructions were avoided during implementation that would cause a longer execution time of the trigonometric functions (such as square root and deviation). Figure 4-2 shows floating-point single-precision data processing instructions cycle timing behavior of the FPU.

Example instruction	Cycles	Early Reg	Result latency
VMLA.F32 <Sd>, <Sn>, <Sm> ^a	1 ^b	<Sn>, <Sm>	5 ^c
VADD.F32 <Sd>, <Sn>, <Sm> ^d	1	<Sn>, <Sm>	2
VDIV.F32 <Sd>, <Sn>, <Sm>	2	<Sn>, <Sm>	16
VSQRT.F32 <Sd>, <Sm>	2	<Sm>	16
VMOV.F32 <Sd>, #<imm>	1	-	1
VMOV.F32 <Sd>, <Sm> ^e	1	-	1
VCMP.F32 <Sd>, <Sm> ^f	1	<Sd>, <Sm>	-
VCMP.F32 <Sd>, #0.0 ^f	1	<Sd>	-
VCVT.F32.U32 <Sd>, <Sm> ^g	1	<Sm>	2
VCVT.F32.U32 <Sd>, <Sd>, #<fbits> ^h	1	<Sd>	2
VCVTR.U32.F32 <Sd>, <Sm> ⁱ	1	<Sm>	2
VCVT.U32.F32 <Sd>, <Sd>, #<fbits> ^j	1	<Sd>	2
VCVT.F64.F32 <Dd>, <Sn>	3	<Sm>	5

Figure 4-2. Floating-Point Single-Precision Instructions

4.3 Memory Placement

The purpose of the Tightly-Coupled Memory (TCM) is to provide low-latency memory that the processor can use without the unpredictability that is a feature of caches. The main use of TCM is to store performance critical data and code. Interrupt handlers, data for real-time tasks and OS control structures are a common example. There are two external TCMs that can be configured to only store instructions, only data, or a mixture of the two (TCMA and TCMB). In our library, the polynomial coefficients were placed in TCMB, and the functions themselves in TCMA.

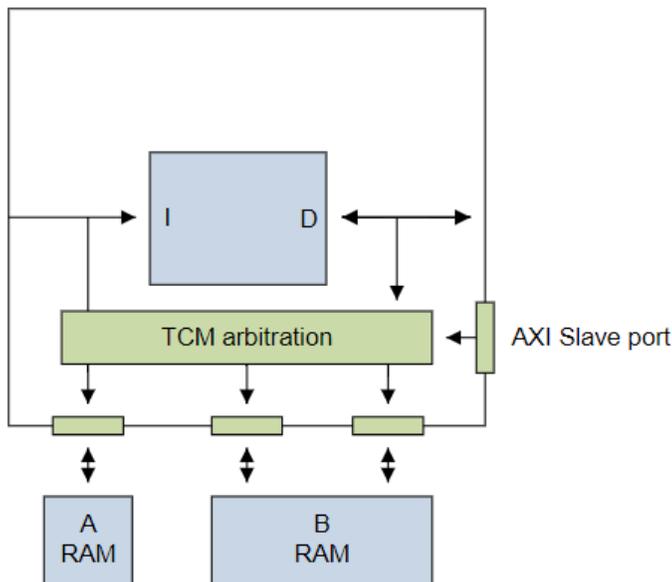


Figure 4-3. Memory Placement

Enabling a TCM to include both instructions and data provides more flexibility from a system perspective, but might limit performance compared with optimizing a TCM to solely store instructions or data. The TCMB is accessible via two ports. This indicates that the TCM has been implemented as two separate banks of RAM so that the two banks can be accessed simultaneously.

4.4 Compiler

The compiler optimization to be **-O3** was configured so that the compiler minimizes some attributes of the executable program. To understand what optimizations are performed on this level, see the *Optimization Options* section in the [TI Arm Clang Compiler Tools User's Guide](#).

In order to place different parts in different memory sections, use the attribute syntax (which is described in the [Attribute Syntax](#) section of the [TI Arm Clang Compiler Tools User's Guide](#)).

Here is an example from the implementation of placing coefficients in TCMB:

```
float __attribute__((aligned(8), section(".testInData"))) ti_r5fmath_cosCoef[5] =
{ 0.999999953466670136306412430924463351f,
  -0.49999905347076729097546897993796764f,
  0.0416635846931078386653947196040757567f,
  -0.00138537043082318983893723662479142648f,
  0.0000231539316590538762175742441588523467 };
```

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision * (July 2022) to Revision A (August 2022)	Page
• Updated Section 1	2
• Updated Section 2	2
• Updated Section 3.4	8
• Updated Section 4.1	9

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to [TI's Terms of Sale](#) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2022, Texas Instruments Incorporated