Technical notes on using Analog Devices products and development tools
Visit our Web resources http://www.analog.com/ee-notes and http://www.analog.com/processors or e-mail processor.support@analog.com or processor.tools.support@analog.com for technical support.

Using the ADSP-CM41x MATH Unit for Clarke and Park Transforms

Contributed by Punarva Katte *Rev 1 – January 3, 2017*

Introduction

The ADSP-CM41x family of mixed-signal control processors provides an on-chip MATH accelerator unit, which can be used to offload most of the common transcendental functions such as e^x, sin(x), cos(x). atan2(y/x), etc., from the Cortex-M4F core. The accelerator is tightly coupled to the Cortex-M4F core and is within the core clock domain. The unit is operated with a simple and flexible store-load mechanism by storing operands to registers and reading results from other registers.

Clarke and Park transformations are matrices of transformation to convert the current/voltage system of any ac-machine from one base to another. The Clarke transform converts a three-phase system into a two-phase system in a stationary frame. The Park transform converts a two-phase system from a stationary frame to a rotating frame. By changing the reference frame, it is possible to considerably simplify the complexity of the mathematical machine model. These techniques are invaluable tools in the digital control of ac-machines.

The purpose of this EE-note is to introduce users to the MATH accelerator unit and techniques that may be used to reduce the computation time of mathematical calculations, such as the Clarke and Park transforms.

MATH Accelerator Unit

The MATH unit provides accelerated functions such as reciprocal, square root, trigonometric functions, exponential functions, and their inverses. It also provides accelerated functions to convert between rectangular and polar coordinates. Most operations by this tightly-coupled accelerator complete in a deterministic number of core clock cycles, faster than the Cortex-M4F core could accomplish the same task. Table 1 provides the cycles taken to execute each of the functions using the MATH unit. The two columns, *Full Domain* and *Normal Domain*, correspond to different ranges of input to the function. Refer to the ***ADSP-CM41x Hardware Reference Manual***[1] for more details.

The MATH unit provides an easy-to-use function calculator for general programming operations. Its operands, results, and functions adhere to the *IEEE 754-2008 Single-Precision Floating-Point Arithmetic Standard*[2]. In general, results returned are accurate to within a standard relative error of 23.5 bits compared to the infinitely precise mathematical result.

Function	Full Domain	Normal Domain
adi_recipf(x) or 1/*x*	9 cycles	9 cycles
adi_sqrtf(x) or √x	9 cycles	9 cycles
adi_expf(x) or e^x	9 cycles	9 cycles
adi_exp2f(x) or 2^x	8 cycles	8 cycles
adi_log2f(x)	10 cycles	10 cycles
adi_lnf(x)	11 cycles	11 cycles
adi_sinf(x)	x = [−∞, +∞]: 14 cycles	x = (-8,+8): 9 cycles
adi_cosf(x)	x = [−∞, +∞]: 14 cycles	x = (-8,+8): 9 cycles
adi_tanf(x)	x = [−∞, +∞]: 20 cycles	x = (-8,+8): 13 cycles
adi_asinf(x)	|x| ≤0.5: 11 cycles 0.5< |x| ≤0.75: 12 cycles 0.75< |x| ≤1: 15 cycles	
adi_acosf(x)	|x| ≤0.5: 12 cycles 0.5< |x| ≤0.75: 13 cycles 0.75< |x| ≤1: 14 cycles (and 15 cycles for negative x)	
adi_atanf(x)	|x| ≤0.00325: 8 cycles |x| >0.00325: 20 cycles	
adi_atan2f (x,y)	x: [−∞, +∞] y: [−∞, +∞]: 22 cycles	
adi_hypotf(x,y) or $\sqrt{(x^2+y^2)}$	|x| ≤0.00325: 10 cycles |x| >0.00325: 22 cycles	
adi_rtopf(x,y)	x: [−∞, +∞] y: [−∞, +∞]: 33 cycles	
adi_ptorf(r,a)	r:[0,+∞] a:(−∞, +∞): 20 cycles	r:[0,+∞] a:(−8, +8): 15 cycles

Table 1. Core Clock Cycles taken by MATH Unit Operations

Note:

1. adi_tanf requires an additional 8 cycles if the input operand (after normalizing by 2π) falls in the range $\left(\frac{\pi}{4},\frac{3\pi}{4}\right)$ *or* $\left(-\frac{3\pi}{4},-\frac{\pi}{4}\right)$
2. Cycle counts do not include latencies associated with loading and storing from/to the MATH accelerator registers. With code optimization, it is possible to achieve a MMR latency of:
    a. 4-5 cycles, for single-operand functions.
    b. 6 cycles, for double-operand, single-output functions like adi_atan2f and adi_hypotf.
    c. 7 cycles, for double-operand, double-output functions like adi_rtopf and adi_ptorf.

The MATH unit is operated using stores and loads for operands and results. All hardware synchronization is handled automatically. Refer to the *Math Programming Model* in the ***ADSP-CM41x Hardware Reference Manual*** for a guide to optimal usage of the MATH functions using C or assembly.

FPMark™

Similar to Coremark, Dhrystone, Whetstone, etc., FPMark[3] is an EEMBC benchmark used to evaluate embedded processors or processing units. Unlike other benchmarks, it is used to evaluate the floating-point computation capability of a processor. The FPMark suite consists of algorithms like Fast Fourier Transform, Horner's Method, Linear Algebra, ArcTan, Neural Net, Black Scholes, Enhanced Livermore Loops, LU Decomposition, and Ray Tracer.

Both the Cortex-M4F core and the ADSP-CM41x MATH unit can perform floating-point operations. Hence, FPMark is a good benchmark to compare the performance between the two. Table 2 shows the performance improvement (in %) when using the MATH unit as compared to the Cortex-M4F core, for each algorithm.

FPMark Algorithm	Performance Improvement
ArcTan	None
Black Scholes	24.68%
Horner's Method	None
Linear Algebra	None
Enhanced Livermore Loops	15.44%
Neural Net	22.08%

Table 1. Performance Comparison Between Cortex-M4F Core and ADSP-CM41x MATH Unit

Clarke and Park Transforms

A significant breakthrough in the analysis of three-phase ac machines was the development of the *Reference Frame Theory*[4]. These techniques are invaluable for analysis, simulation and digital control (like current, torque and flux) of AC machines. Over the years, many different reference frames have been proposed for the analysis of ac machines, with the most commonly used being the *Stationary Reference Frame* and the *Rotor Reference Frame*.

Clarke Transform (Three-Phase to Two-Phase)

Three-phase ac machines are conventionally modeled using phase variable notation, though it is possible to transform the system to an equivalent two-phase representation. The transformation from three-phase to two-phase quantities is written in matrix form as Equation 1:

$$\begin{bmatrix} i_{s\alpha}(t) \\ i_{s\beta}(t) \end{bmatrix} = \frac{2}{3}\begin{bmatrix} 1 & \cos(\gamma) & \cos(2\gamma) \\ 0 & \sin(\gamma) & \sin(2\gamma) \end{bmatrix}\begin{bmatrix} i_{sA}(t) \\ i_{sB}(t) \\ i_{sC}(t) \end{bmatrix}$$

- γ is the separation between axes of the three-phase machine, which is conventionally $\frac{2\pi}{3}$.
- i_{sA}, i_{sB} and i_{sC} are three-phase stator currents.
- $i_{s\alpha}$ and $i_{s\beta}$ are two-phase stator currents.

Note that the transformation is equally valid for the voltages and flux linkages.

Substituting $\gamma = \frac{2\pi}{3}$, this becomes Equation 2:

$$\left.\begin{aligned} i_{s\alpha} &= \frac{1}{3}(2i_{sA} - i_{sB} - i_{sC}) \\ i_{s\beta} &= \frac{1}{\sqrt{3}}(i_{sB} - i_{sC}) \end{aligned}\right\}$$

In a balanced three-phase ac-machine, $i_{sA} + i_{sB} + i_{sC} = 0$, which simplifies to Equation 3:

$$\left.\begin{aligned} i_{s\alpha} &= i_{sA} \\ i_{s\beta} &= \frac{1}{\sqrt{3}}(i_{sA} + 2i_{sB}) \end{aligned}\right\}$$

Park/Inverse Park Transform (Vector Rotation)

In the analysis of electrical machines, it is generally necessary to adopt a common reference frame for both the rotor and the stator. For this reason, a second transformation, known as a *vector rotation*, is formulated that rotates space vector quantities through a known angle. The transformation can be written in matrix form as Equation 4:

$$\begin{bmatrix} i_{sd} \\ i_{sq} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} i_{s\alpha} \\ i_{s\beta} \end{bmatrix}$$

- θ is the angle of rotor from stator.
- i_{sd} and i_{sq} are direct and quadrature axis components of the current space vector, respectively.

This transformation is referred to as the *Inverse Park Transformation.*

Similarly, the transformation used to rotate from rotor frame to stator frame is the *Park Transformation*, and the matrix form of this transform is shown in Equation 5:

$$\begin{bmatrix} i_{s\alpha} \\ i_{s\beta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} i_{sd} \\ i_{sq} \end{bmatrix}$$

The elimination of position dependency from the machine electrical variables is the main advantage of vector rotation.

Transform Implementation on ADSP-CM41x Devices

Equation 2 and Equation 3 (Clarke Transform), Equation 4 (Inverse Park Transform), and Equation 5 (Park Transform) are implemented on the ADSP-CM41x device using simple C code as a part of one of two implementations:

- the math.h library, which runs on the Cortex-M4F core
- the ADSP-CM41x MATH Unit Accelerator

Both are run with code in SRAM or flash memory, and the number of core cycles required can be measured. Table 3 shows the performance of each function averaged over 128 samples of input data, using both the Cortex-M4F core and the ADSP-CM41x MATH unit, with code in SRAM or flash memory.

Motor Control Kernels		Clarke Transform		Clarke Transform (Balanced)		Park Transform		Inverse Park Transform	
		SRAM	Flash	SRAM	Flash	SRAM	Flash	SRAM	Flash
Cortex-M4F using math.h (Core cycles / 128)		21.078	21.141	13.055	13.102	200.547	239.156	201.547	239.380
MATH Accelerator	Core cycles / 128	21.078	21.141	13.055	13.102	35.055	35.078	35.055	35.078
	ns	87.826	88.086	54.395	54.590	146.061	146.160	146.061	146.160

Table 2. Performance of Motor Control Functions

The results in Table 3 were obtained using a core clock of 240 MHz and a system clock of 96 MHz with random values chosen for the input currents and theta.

Along with measuring the performance, the code also compares the accuracy of output currents with the standard implementation (with same set of inputs).

Techniques for Optimal Use of the MATH Unit

This section describes some optimization techniques for getting the best performance out of the MATH unit.

adi_math.h

When developing a C/C++-based project using the MATH unit functions, include the *adi_math.h* header file that is included in the ADSP-CM419F EZ-KIT® Board Support Package. This file defines the MATH unit operations as inline functions, and including it will directly replace the corresponding *math.h* library functions so that the code need not be changed when moving across platforms. An example is shown in Listing 1.

```
#define sinf(x)     adi_sinf(x)

inline
float32_t adi_sinf (float32_t x ) {pADI_MATH0->SINF = x; return pADI_MATH0->RES1;}
```

Listing 1. sinf() implementation in adi_math.h

With the appropriate compiler optimizations, the inline functions are disassembled to the most optimal assembly instructions, as described in the *Math Assembler Programming Model* in the ***hardware reference manual***.

Compiler Optimization

The IAR tools provide four levels of compiler optimization – none, low, medium, and high - and seven different optimization techniques – common subexpression elimination, loop unrolling, function inlining, code motion, type-based alias analysis, static clustering, and instruction scheduling. For efficient use of the *adi_math.h* library, choose the high (with speed) optimization level and enable at least the common subexpression elimination and function inlining optimization techniques.

Code Interleaving

The MATH unit does not require the Cortex-M4F core to remain idle while it performs calculations. To optimize code for performance, it is useful to move unrelated code between the operand-store and the result-load operations. In this manner, the effective time of a function call is eliminated (except for the MMR read/write latency). Refer to the *Math Programming Model* in the ***hardware reference manual*** for a guide for interleaving code with MATH unit operations.

References

[1] *ADSP-CM41x Mixed-Signal Control Processor with ARM Cortex-M4/M0 and 16-bit ADCs Hardware Reference Manual.* Rev 0.2, May 2016. Analog Devices, Inc.

[2] *IEEE Standard for Floating-Point Arithmetic (IEEE Std 754-2008).* August 29, 2008. IEEE Computer Society (Sponsored by the Microprocessor Standards Committee).

[3] *EEMBC® FPMark™: The Embedded Industry's First Standardized Floating-Point Benchmark Suite.* August, 2013. The Embedded Microprocessor Benchmark Consortium.

[4] *ADSP-21990: Reference Frame Conversions (AN21990-11).* January 2002. Analog Devices, Inc.

Document History

Revision	Description
Rev 1 – January 3rd, 2017 by Punarva Katte	Initial Release