EFM32
... the world's most energy friendly microcontrollers

Digital Signal Processing with the EFM32

AN0051 - Application Note

Introduction

This application note shows how to perform Digital Signal Processing (DSP) on the EFM32 using the DSP library provided with CMSIS in Simplicity Studio. The differences between the ARM Cortex-M3 and Cortex-M4F cores are considered.

This application note includes:

- This PDF document
- Source files (zip)
  - Example C-code
  - Multiple IDE projects

ARM ZERO TINY GECKO LEOPARD GIANT WONDER

SILICON LABS

1 Introduction

Digital Signal Processing (DSP) is the act of mathematically processing signals using a digital computer. The signals may be samples from an ADC, data from an MP3 file, pixels in a picture or some other source of digital data. These samples go through one or multiple steps of processing before the result is stored or output in some way.

The processing often involve, but is not limited to, filtering, transformations between time and frequency domains, matrix calculations and compression.

Real-time DSP applications often use microprocessors that are specially designed with DSP in mind. But microcontrollers in the EFM32 family has a good amount of processing power, and many DSP applications can run directly on the microcontroller. This reduces both cost and the energy consumption of the application.

In cases where DSP processing is performed, but the processing performance of a signal processing core is not strictly needed, it can still give a huge energy saving benefit because of the reduced processing time a DSP CPU offers. This is demonstrated in the included *lightsensefft* example.

2 DSP Concepts

This document uses several DSP specific concepts to describes the features of the Cortex-M4F enabled EFM32 devices. This section will outline and describe some of these concepts.

2.1 Number Formats

Most of the DSP functions in a software DSP Library comes in both single precision floating point and fixed point variants. The difference between floating and fixed point numbers is outlined below.

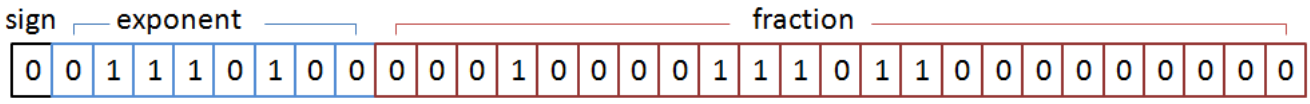
2.1.1 Floating Point

In computing, floating point numbers is a method of representing an approximation to real numbers in a way that can support a wide range of values. The numbers are, in general, represented approximately to a fixed number of significant digits (the mantissa) and scaled using an exponent.

Since the floating point unit in the Cortex-M4F is only single precision, the rest of this document will only discuss the single precision floating point type.

The IEEE 754 standard specifies the floating point formats and operations. A single precision floating point number is stored as shown in Figure 2.1 (p. 3)

Figure 2.1. IEEE 754 Single Precision floating point format



The first bit is the sign bit, then there are eight bits for the exponent, and the remaining 23 bits are used for the fractional part.

Unless you have an Cortex-M4F which includes an FPU, all floating point calculations will be performed in software. The compiler will insert soft-float routines to perform the floating point calculations using regular integer instructions. Floating point calculations using the FPU, and also fixed-point calculations, will be significantly faster than these soft-float calculations.

Floating point acceleration in hardware will help reduce code development time and cost in almost every application where the data processed has 16 bits or more dynamic range. Floating point gives the programmer freedom in the choice of algorithm to use. It also frees the designer of the tedious task of figuring out if the result of a calculation will be within reasonable size compared to the variable size, also known as overflow or underflow.

When processing unknown sensor input, making sure that the result of every calculation stays within the integer variable size and at the same time carry enough information (not underflowing) can be difficult. With floating point variables, large/small result values will only reduce the accuracy, not cause an error condition such as overflow or underflow.

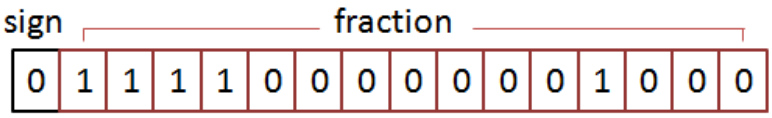
2.1.2 Fixed Point

Fixed point formats are ways of representing rational numbers using integer types. One does this by deciding a fixed position in the integer where the decimal mark should be, or rather, how many bits should represent the integer part of the number, and how many bits should represent the fractional part. The format is then called `Qn.m` where `n` is the number of integer bits, and `m` is the number of fractional bits.

Taking Q1.15 as an example, has one bit for the integer part, and 15 bits for the fractional part. These numbers are signed, and the first bit therefore represents the sign of the number. With 15 fractional bits, dividing the signed integer by 2^{15} gives the Q1.15 represented number.

Figure 2.2 (p. 4) shows the Q1.15 fixed point representation of a number approximately equal to 0.937744

Figure 2.2. Q1.15 fixed point representation



For signed 16-bit integers, *0x8000* = *-32768* is the lowest representable number, this means the lowest representable number of Q1.15 is

Lowest number representable with Q1.15

$$-32768 / 2^{15} = -1 \quad (2.1)$$

0x7fff = *32767* is the highest representable number for 16-bit integers, so the highest representable number of Q1.15 is about

Highest number representable with Q1.15

$$32767 / 2^{15} = 0.99996948 \quad (2.2)$$

Care must be taken when performing operations on fixed point numbers. The two rational numbers *a* and *b*, are represented in fixed point and stored as the signed integers

Two numbers in fixed point representation

$$A = a * 2^{15} \text{ and } B = b * 2^{15} \quad (2.3)$$

Addition and subtraction can then be done simply by adding or subtracting the integers *A* and *B*, although over-/underflow might occur as with regular integer arithmetic. Multiplication and division does not work the same way. Simply multiplying A with B gives

Multiplying fixed point numbers

$$A * B = a * b * 2^{15} * 2^{15} \quad (2.4)$$

and the result needs to be shifted 15 bits to the right in order to obtain the fixed point representation $C = a * b * 2^{15}$.

Dividing A by B gives

Dividing fixed point numbers

$$A / B = a / b \quad (2.5)$$

and the result needs to be shifted 15 bits to the left in order to obtain the fixed point representation $C = (a / b) * 2^{15}$.

The CMSIS DSP Library uses the fixed points formats Q1.7, Q1.15 and Q1.31 represented in the code with type names q7, q15 and q31.

2.2 DSP Instructions

When implementing DSP algorithms one often ends up with a set of typical instruction patterns that are executed in the inner most loops of the algorithm. To make these number crunching patterns run more

efficiently and in less cycles, DSP processors include a few families of special instructions, some of these are outlined below.

2.2.1 Multiply Accumulate Instructions

These instructions perform a multiplication and adds the result to an existing register. They are very useful in filter algorithms like FIR and IIR filters, but also in other applications such as FFT (Fast Fourier Transform).

2.2.2 Saturating Instructions

These instructions limit the range of the resulting value instead of letting it under- or overflow. If we for example add two 16-bit numbers that would normally overflow, with saturating instructions, the result will hit the maximal value 32767 without overflowing. This is very useful in audio and image application where one wants to avoid distortions due to over- and underflowing registers.

2.2.3 SIMD, Single Instruction Multiple Data

This is a set of instructions that can be performed on multiple data input in parallel. One can for example perform more than one addition during the same cycle. The Cortex-M4 supports instructions performing four 8-bit operations or two 16-bit operations at once.

3 The CMSIS DSP Library

CMSIS, the Cortex Microcontroller Software Interface Standard by ARM, is a hardware abstraction layer for Cortex-M processors, and it includes a DSP library written purely in C.

3.1 Where To Find The CMSIS DSP Library

Simplicity Studio comes with the CMSIS Library installed. It is located in the CMSIS folder under the Simplicity Studio work folder.

Documentation for CMSIS, including the DSP library is found under API documentation in Simplicity Studio. It includes thorough explanations of all the included functions and data structures.

Inside the *CMSIS* directory there is also a *DSP_Lib* directory which contains the DSP library. Included is both the C source for all the library functions, and examples on how to use some of them.

3.2 CMSIS DSP Library Contents

A full description of all the included functions can be found in the CMSIS DSP Library documentation as described above. Here is an overview of what it includes:

- *BasicMathFunctions* contains the most elementary vector operations, like addition, subtraction, scaling, and dot products.
- *ComplexMathFunctions* contains functions for dealing with vectors of complex numbers.
- *ControllerFunctions* contains some transforms and algorithms popular in industrial control systems, including a Proportional Integral Derivative (PID) controller, Clarke and Park transforms.
- *FastMathFunctions* contains fast approximating functions to the regular trigonometric functions sine, cosine, and also the square root.
- *FilteringFunctions* contains a collection of different filtering functions, including FIR, IIR, biquad and LMS filters. There are also functions for calculating the correlation and convolution of two sample sequences.

  These filters work on a block of samples at a time. A state object is used to keep track of the filter state between blocks.
- *MatrixFunctions* contains functions for dealing with matrices. A matrix is initialized from a data array and operations like addition, subtraction, scaling, matrix multiplication etc. can be performed by calling these functions. The library also contains a matrix inversion function based on the Gauss-Jordan algorithm.
- *StatisticsFunctions* contains functions for calculating descriptive statistics on input vector, including maximum, minimum, mean, sum of squares root mean square, standard deviation, variance etc.
- *SupportFunctions* contains some basic functions for dealing with arrays of data, and converting between the different numerical types.
- *TransformFunctions* contains the most common transforms. Included is functions performing the complex fast Fourier transform, in both radix-2 and radix-4 variants. Also included is a type-IV discrete cosine transform, useful in audio and image compression applications. There is also a Fast Fourier Transform for real valued arrays.

Note

Remember to read the documentation of the DSP Library functions you are using, they might not work exactly as you expect. Especially when using the fixed point functions, there are some traps. Some fixed point functions output values in a different format than the input, and some functions expect you to scale your input in order to avoid overflow.

Most of the filtering and transform functions require the user to initialize a state object before doing the actual processing. When using the matrix functions, matrix objects will need to be initialized from data arrays.

3.3 How To Use The CMSIS DSP Library

To use the CMSIS DSP Library, include *arm_math.h* which holds the function declarations.

arm_math.h expects one of the symbols

- ARM_MATH_CM0
- ARM_MATH_CM3
- ARM_MATH_CM4

to be defined. Depending on what core used, define one of them in the preprocessor/compiler section of the IDE project.

The next step is to import the library into the project so that the linker can find the actual function definitions. This can be done in one of two ways:

Either include a pre-compiled library file found under the *Lib* subdirectory in the *CMSIS* folder. Choose the folder corresponding to the IDE and the file corresponding to the processor core. M0l means little endian Cortex-M0, and M4lf means little endian Cortex-M4F (with FPU). All EFM32 CPUs are little endian.

Note

There is no IAR folder, but IAR provides it's own library which is found in the installation directory of IAR under `Embedded Workbench N.M\arm\CMSIS\Lib\IAR`

Or import the .c-files needed from *CMSIS\DSP_Lib\Source* and compile them together with the rest of the application. Note that one might need to import more than just the files used directly in the application, as the library functions sometimes needs other library functions and tables. This approach is shown in the attached software examples.

Note

The performance might differ between the pre-compiled libraries and compiling them yourself, so if the DSP functions in your application are performance critical, you should try the different options, and also get to know the optimization options of your compiler and linker.

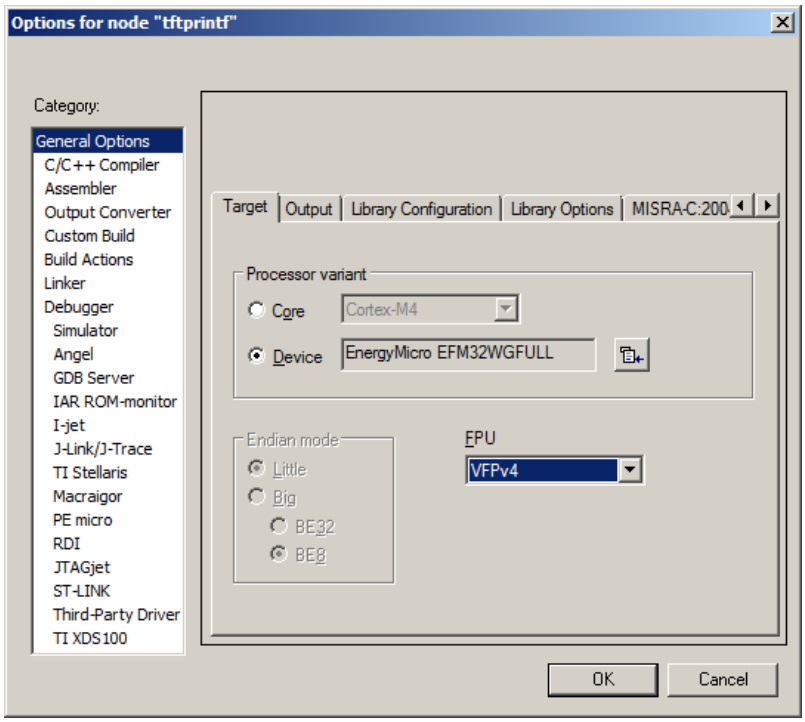
3.3.1 Enabling the FPU on the Cortex-M4F

3.3.1.1 IAR

Before using the FPU of the Cortex-M4F core, it needs to be enabled, this is done in the `SystemInit()` function, located in the `system_efm32wg.c` file.

The compiler and linker also needs to know that an FPU is available. In IAR this is done by selecting VFPv4 in the FPU box of the General Options of the project like shown in Figure 3.1 (p. 8)

Figure 3.1. FPU Enabled in IAR Project options



3.3.1.2 GCC

For GCC there are specific compiler and linker flags associated with the floating point unit. These flags are identifiable by comparing the soft float- and hard float-Makefiles for GCC projects included with this application note. The FPU related flags are:

- *-D__FPU_PRESENT=1* Tells the compiler that a floating point unit is present, use FPU instructions allowed.
- *-mfloat-abi=name* Tells compiler and linker how floating point parameters are passed to functions.
- *-mfpu=fpv4-sp-d16* Set the type/version of FPU.

If using pre-compiled libraries with GCC, make sure that the compiler and linker Embedded Application Binary Interface (EABI) settings (*-mfloat-abi=name*) for floating point matches what was used for the pre-compiled library. This ensures that floating point parameters are passed correctly between the functions in the library and application.

-mfloat-abi=name Specifies which floating-point ABI to use. Permissible values are: ‘soft’, ‘softfp’ and ‘hard’.

More information about the floating point EABI flag (*-mfloat-abi*) can be found here: **http://gcc.gnu.org/onlinedocs/gcc/ARM-Options.html**

4 DSP On The Cortex-M Family

For any low power application that includes some form of digital signal processing, the DSP-enabled Cortex-M4F cpu core is ideally suited as it will significantly lower the energy consumption of the application.

Digital signal processing can be done with both fixed point and floating point number types. While the ARM Cortex-M3 does not include hardware support for floating point types, the Cortex-M4 in the EFM32 devices includes a hardware Floating Point Unit (FPU) which accelerates floating point operations considerably. When this FPU is present, the core is called Cortex-M4F. Cortex-M4 also comes with a whole set of single cycle integer DSP instructions. These are instructions that are often used in DSP applications.

4.1 Single Precision Floating Point Unit

The EFM32 Wonder Gecko family of microcontrollers is based on the Cortex-M4F which contains an FPU. The FPU is single precision, which means floating point operations on 32-bit sized operands will be performed in hardware. Floating point operations on this FPU is significantly faster than the software routines that perform floating point operations on cores without an FPU.

The FPU is IEEE 754 compliant, so the operations, rounding, exceptions etc. can be expected to perform as specified in this standard.

The single precision floating point type *float32_t* is defined in *arm_math.h* and works just like regular floating point types.

Read more about the IEEE 754 standard at

http://grouper.ieee.org/group/754/

Table 4.1 (p. 9) shows the most central floating point instructions and their cycle counts.

Table 4.1. Some of the floating point instructions of the Cortex-M4F

Instruction	Description	Cycles
VADD.F32	Addition	1
VSUB.F32	Subtraction	1
VMUL.F32	Multiply	1
VDIV.F32	Division	14
VCVT.F32	Conversion to/from integer/fixed-point	1
VLDM.32	Load N floats	1+N
VMOV	Move float to float-register	1
VFMA.F32	Fused multiply accumulate	3
VSQRT.F32	Square root	14
VSTM.32	Store N float registers	1+N

4.2 Cortex-M4 DSP Instruction Set

The Cortex-M4 integer DSP instructions can increase the performance of integer and fixed point DSP applications.

The instruction set includes typical DSP instructions such as: Multiply Accumulate Instructions (MAC), Saturating Instructions (SAT) and Single Instruction Multiple Data (SIMD) instructions. All of these instructions are single cycle on the Cortex-M4 core.

For more information about the M4 DSP instructions and the instruction sets of the different cores, see ARM's infocenter at

http://infocenter.arm.com

4.3 Benchmarks

4.3.1 FFT Comparison

Figure 4.1. Comparison of floating and fixed point FFT on the Cortex-M3 and Cortex-M4F

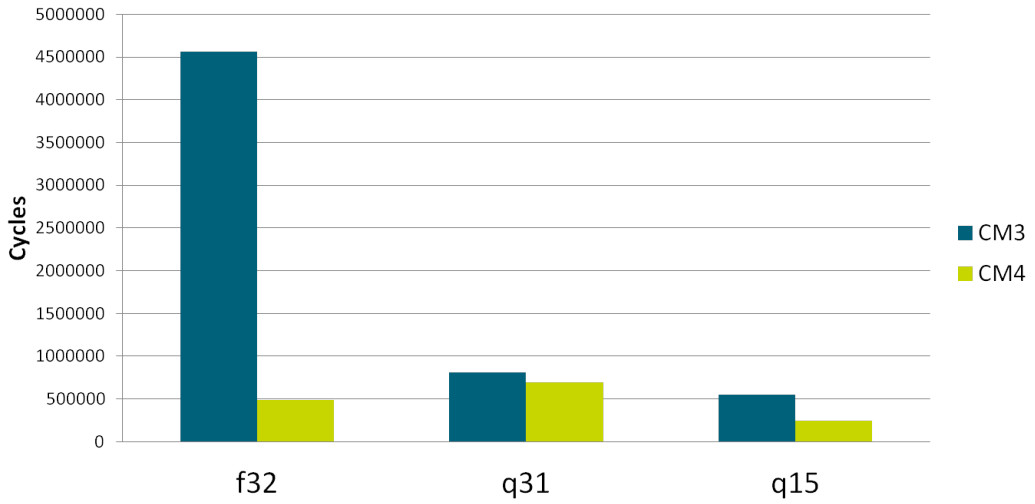


Figure 4.1 (p. 10) shows the cycle count of the complex FFT algorithm provided in the CMSIS DSP Library. All three versions of the algorithm is run on both the Cortex-M3 and the Cortex-M4F. The biggest increase is seen in floating point performance due to the change from soft-float to hard-float. The Q1.31 fixed point type algorithm does not perform much better, but the Q1.15 algorithm does perform more than twice as fast due to 16-bit SIMD. As mentioned previously, SIMD on the Cortex-M4 includes instructions for operating on two sets of 16-bit data in parallel.

4.3.2 Matrix Calculations

Figure 4.2. Comparison of cycle count for ten 10*10 matrix multiplications on the Cortex-M3 and Cortex-M4F

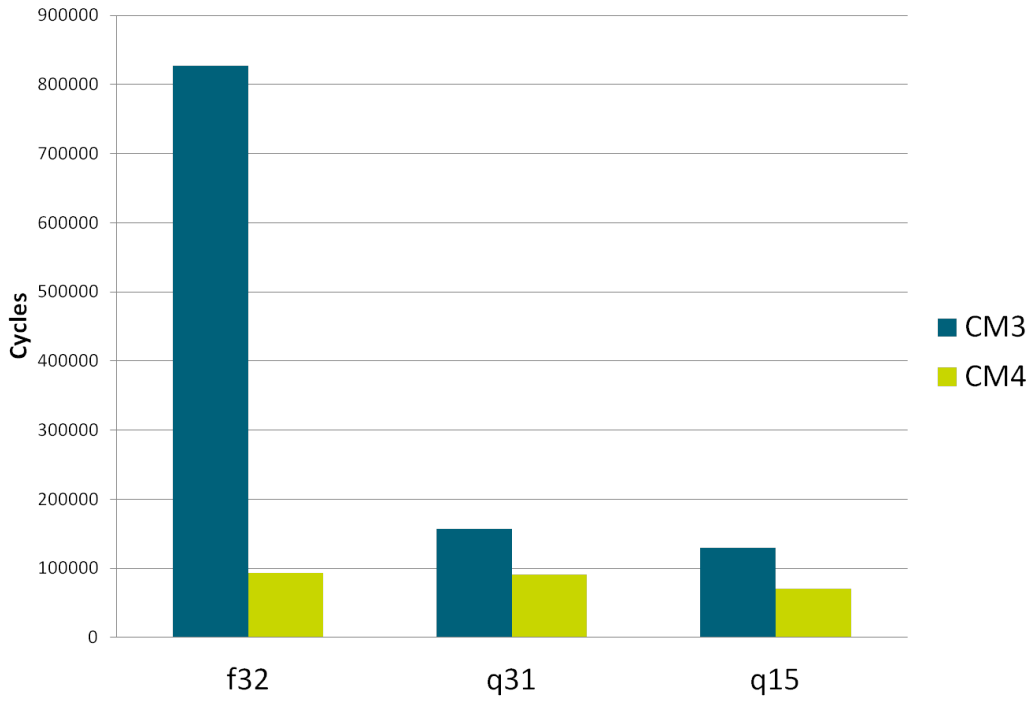


Figure 4.2 (p. 11) shows the cycle count of 10 matrix multiplications of 10 by 10 matrices with either floating point, 32 bit or 16 bit fixed point values. Matrix multiplications are typically used in sensor filtering applications, for instance statistical combination of sensor data, also known as Kalman filtering.

For more information about the Wonder Gecko benefits, please see the EFM32 Wonder Gecko White paper.

http://www.energymicro.com/downloads/white-papers

5 Software Examples

Included with this application note are several examples on how to use the CMSIS DSP Library.

5.1 Frequency Estimation

The frequency estimation example is made for the DK3750/DK3850. It samples audio from the audio in port of the DK, and uses FFT and *sinc* interpolation to display the loudest frequency with a precision of about 0.1Hz on the TFT.

The interpolation method used can be found in Appendix A of the following article by M. Aung, W. J. Hurd, C. M. Buu and J. B. Berner:

http://tmo.jpl.nasa.gov/progress_report/42-118/118I.pdf

5.2 FIR Filter Demonstration

This example implements a low pass FIR filter with 29 taps to filter an array with input values. The example comes directly from the CMSIS DSP Library and shows how easy it is to compile and try these examples.

5.3 Matrix Multiplication Example

This example uses the CMSIS DSP Library to perform matrix multiplication. The same two matrices are multiplied using floating point, Q15 fixed point and Q31 fixed point format. The number of cycles used is sent out over SWO with printf. The eACommander software includes an easy to use GUI terminal to capture SWO printf data.

5.4 FFT with Light Sensor Data

This example uses the CMSIS DSP Library to perform an FFT of ADC sampled light sensor data on the starter kits. The EM2 + ADC routine presented in the ADC application note is used to sample the light sensor in an energy efficient way at 1 kHz sample rate.

Note

In order to connect the light sensor to the correct ADC channel for this example, the user must connect the light sensor pin (PC6) with one of the ADC pins (PD0-PD7). The default channel used in the software example is ADC channel 5, which is pin PD5.

Every 0.5 seconds a 512 point FFT is executed together with a frequency estimation algorithm to figure out the frequency with the most energy in the spectrum (except DC). In indoor fluorescent lighting this will typically be 50/60 Hz. 100/120 Hz is displayed because the light has a peak in intensity both at the top and bottom of the sine wave. Both the detected frequency and CPU cycle count is displayed on the STK's LCD display.

This example demonstrates that even though the cortex M3 has more than enough power to do the DSP calculations, the cortex M4F will make the battery last 3-4 times as long. Typical average current consumption for M3 is around 500-600 uA, while the average for Wonder Gecko should be around 170 uA.

6 Revision History

6.1 Revision 1.04

2014-05-07

Updated example code to CMSIS 3.20.5

Changed source code license to Silicon Labs license

Added project files for Simplicity IDE

Removed makefiles for Sourcery CodeBench Lite

6.2 Revision 1.03

2013-10-14

New cover layout

6.3 Revision 1.02

2013-05-08

Added software projects for ARM-GCC and Atollic TrueStudio.

Reorganized document.

Added new software examples for Wonder Gecko kits.

Added matrix comparison between Cortex M4F and M3.

Added FPU enable and ABI description for GCC.

6.4 Revision 1.01

2012-11-12

Adapted software projects to new kit-driver and bsp structure.

6.5 Revision 1.00

2012-09-18

Initial revision.

A Disclaimer and Trademarks

A.1 Disclaimer

Silicon Laboratories intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Laboratories products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Laboratories reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Laboratories shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products must not be used within any Life Support System without the specific written consent of Silicon Laboratories. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Laboratories products are generally not intended for military applications. Silicon Laboratories products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

A.2 Trademark Information

B Contact Information

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701

Please visit the Silicon Labs Technical Support web page:
http://www.silabs.com/support/pages/contacttechnicalsupport.aspx
and register to submit a technical support request.

Table of Contents

List of Figures

List of Tables

List of Equations

silabs.com


ZERO
ARM Cortex-M0+
TINY
ARM Cortex-M3
GECKO
ARM Cortex-M3
LEOPARD
ARM Cortex-M3
GIANT
ARM Cortex-M3
WONDER
ARM Cortex-M4