Introducing the EFM32 Wonder Gecko:
An Energy Friendly MCU with Signal Processing Capabilities

With the release of the EFM32 Wonder Gecko family, Energy Micro expands its MCU portfolio to combat energy consumption in more applications. The new EFM32 family members include the ARM Cortex-M4 core with a DSP (Digital Signal Processing) instruction set and dedicated Floating Point Unit (FPU). The EFM32 Wonder Gecko includes an extensive set of autonomous low power peripherals and extremely efficient sleep modes. This white paper will explain how these features work and how they help developers build low power systems.

Intelligent Embedded Systems

Embedded devices are becoming more and more sophisticated. A trend in today's market is to incorporate more sensors and do more processing in small nodes which are connected wirelessly to other parts of the system. Names like Internet of Things (IoT) or Machine to Machine communication (M2M) are used frequently for the next generation of devices. The idea is to add more and more 'intelligence' to devices around us, and let these intelligent devices communicate, not only with humans, but also among themselves.

Creating smarter nodes has many benefits. One of them is increased reliability. Nodes can operate independently, without relying on a 'master' or base station to control them. If other parts of the system go down or communication is blocked for some time, nodes can keep operating independently.

On the flipside, adding more functionality or better performance to a node does not come without its drawbacks. Perhaps the most prevalent is increased power consumption. Adding more sensors will not only add power consumption for the sensors themselves, but also require additional processing which causes the MCU to draw more power.

However, processing locally *does* make sense from an energy perspective. When the alternative is to transmit the raw data to a base station over a wireless link and perform the computation there, more energy would typically be consumed during transmitting. Therefore, by processing locally and only transmitting the result, the overall energy consumption goes down.

A central challenge is therefore how to do data processing as efficiently (in terms of energy) as possible. As a consequence, an efficient CPU is very important. At the same time it is important to consider the entire system, including data acquisition and even the time spent idle, to make sure the overall energy consumption is as low as possible.

MCU Power Consumption

When considering power consumption of an MCU, it is typically divided in two concepts: dynamic and static. Dynamic power consumption relates to the power that is consumed by transistors switching on and off, and this quantity is roughly proportional to the clock frequency. Static power relates to the leakage currents that flow when transistors are NOT switching and is more or less a constant quantity.

When an MCU is in active mode, where the CPU is active and running code, the power consumption is much higher than in a sleep mode, typically several orders of magnitude. In active mode the power consumption is dominated by dynamic power and a high clock frequency. In sleep mode on the other hand, the CPU is off and the MCU is typically clocked at a much lower clock frequency (if clocked at all). The sleep mode power consumption is therefore much more influenced by static power.

A typical MCU spends most of its time in sleep mode to save power, and only wakes up the CPU when required. Referring to Figure 1, there are three important parameters that influence the total energy consumption of an MCU:

- Active mode power consumption
- Sleep mode power consumption
- Time spent in active mode

It is clear from this figure that while it is important to reduce the active and sleep mode power consumption, reducing processing time will also have a significant impact.

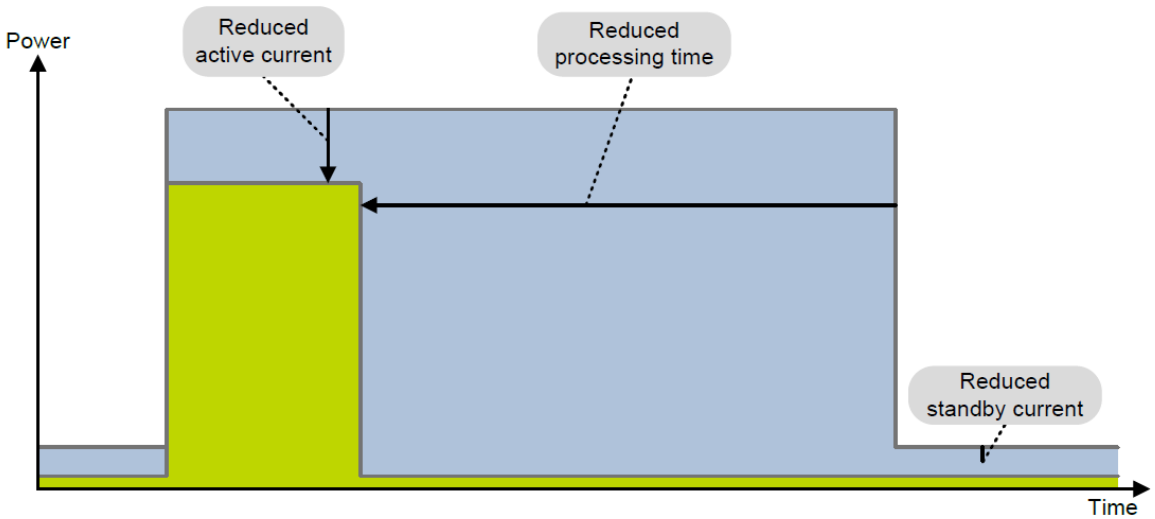

Figure 1: MCU power consumption over time. The total energy consumption is the area below the curves. There are three principal ways to reduce energy consumption: reducing active current, reducing processing time and reducing standby current. The blue area represents the saved energy.

Introducing the Wonder Gecko

This chapter will introduce some of the core features of the EFM32 Wonder Gecko that help save energy.

Cortex-M4 Core with FPU

The EFM32 Wonder Gecko uses the ARM Cortex-M4 core with FPU[1]. With the extended DSP instruction set and integrated hardware Floating Point Unit the Cortex-M4 can reduce the run-time of many common algorithms. The next chapter will go into more detail about these features and how the Cortex-M4 core can help reduce processing time.

Autonomous Peripherals

For data acquisition, the EFM32 Wonder Gecko is equipped with an efficient 12-bit ADC capable of capturing 1 Msample/s, while consuming only 350 µA. The ADC has full DMA support and can be triggered via PRS, the Peripheral Reflex System, that allows peripherals on the EFM32 to interact with each other without waking up the CPU. The CPU can therefore be asleep during measurements and only wake up when data is available in memory and ready for processing.

Many other useful interfaces, both analog and digital are included in the MCU. The EFM32 Wonder Gecko contains a DAC, several timers, analog comparators, pulse counters and operational amplifiers. Hardware support for serial and parallel interfaces are available. All peripherals are designed to have the lowest power consumption possible and the flexible clock gating system allows every module to be turned completely off when not needed. Each peripheral can be configured to run on its optimal frequency by configuring several independent prescalers. A block diagram of the available modules included on the EFM32 Wonder Gecko is shown in Figure 2.

LESENSE

The Low Energy Sensor Interface (LESENSE) is an important peripheral. With LESENSE up to 16 sensors can be monitored autonomously in Deep Sleep mode with the CPU off. LESENSE is capable of duty-cycling the sensors to make sure they are only active when a measurement is being performed. Resistive, inductive and capacitive sensors can all be used with LESENSE. The peripheral has an internal state machine that can be used to only wake up the CPU when a certain sequence of measurements occur, for instance a finger sliding across several touch pads. The power of LESENSE is that sensors can be monitored continuously without CPU intervention. A different microcontroller would have to periodically wake up the CPU to perform the measurements, which would require much more energy.

[1] The FPU is an optional part of the ARM Cortex-M4 IP. Thus it is up to each chip manufacturer if they want to include the FPU.

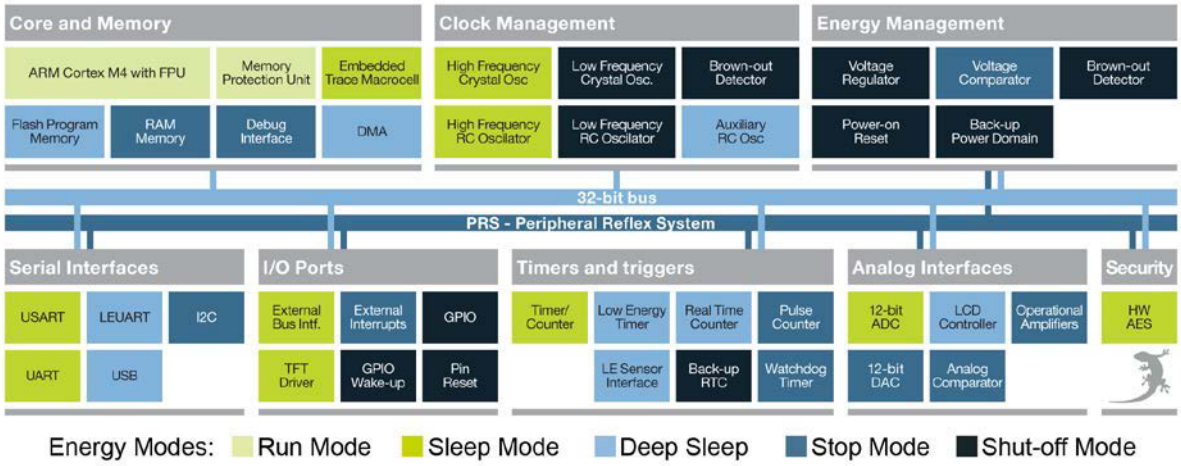

Figure 2: EFM32 Wonder Gecko Block Diagram. The peripherals are color coded to indicate the lowest Energy Mode (sleep mode) where they are able to operate.

Energy Modes

Efficient sleep modes ensure minimum power consumption while the MCU is idle. The EF32 Wonder Gecko includes four optimized sleep modes, also referred to as Energy Modes. The MCU can quickly switch between the various Energy Modes depending on the functionality needed at any point in time. Within each Energy Mode the functionality, and power consumption, can be further customized by turning on and off peripherals. Figure 2 illustrates which peripherals are available in each Energy Mode by color-coding them according to the lowest mode they can be enabled.

In Deep Sleep mode, with full RAM and CPU data retention, power supervision and RTC running, the EFM32 Wonder Gecko only consumes 1 µA of current.

The wakeup time is also kept short. It only takes 2 µs from when an interrupt is received until the core is started. Not only does this ensure a fast response time, but it also directly reduces the energy consumption by wasting as little energy as possible during the wakeup phase.

The EFM32 Wonder Gecko has been designed to minimize leakage currents. Leakage currents are responsible for much of the current consumption in sleep modes. As more and more parts of the MCU are turned off, leakage currents contribute a larger and larger portion of the total current consumption. Leakage currents also increase with temperature, which can cause a significant penalty on battery life time on industrial applications which must operate in rough and hot environments. The Wonder Gecko is designed to have low leakage currents and thus low sleep mode current consumption over the entire operating temperature range.

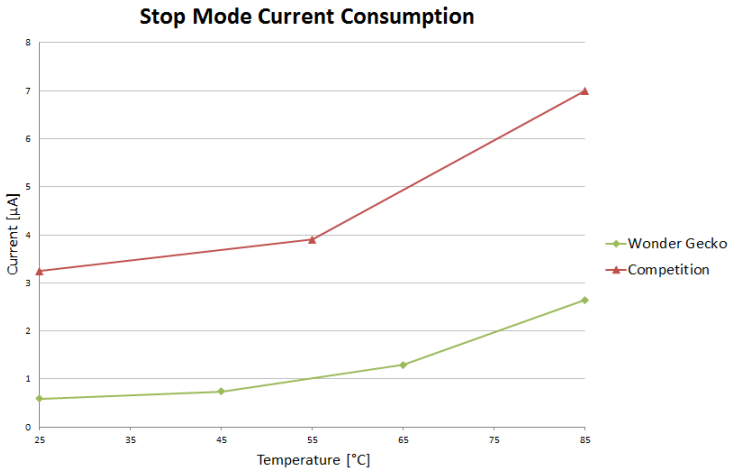

Figure 3: Stop Mode current consumption over temperature (RAM retained, all oscillators off). The EFM32 Wonder Gecko is designed to achieve low sleep mode current consumption over the entire temperature range.

Learn More

To learn more about the technology behind the EFM32 and why it is the world's most energy friendly microcontroller, the reader is encouraged to see the white paper "WP0001 EFM32 Introduction" (available in Simplicity Studio and at energymicro.com). The rest of this paper will focus on the DSP extensions of the Cortex-M4 core and how this can reduce energy consumption by reducing processing time.

DSP Features of the Cortex-M4

The Cortex-M4 core introduces several features that make it particularly suited for handling DSP-type tasks. The instruction set of the Cortex-M4 is a superset of the Cortex-M3 instruction set (see Figure 4) with focus on improving the signal processing capabilities. The integrated hardware FPU speeds up computation when working with floating point numbers.

The DSP extension can be divided into four concepts:

- Single-cycle Multiply Accumulate (MAC) instructions
- Single Instruction Multiple Data (SIMD) instructions
- Saturating instructions
- Floating Point Unit

This chapter will introduce each of these and explain how they can help reduce processing time.

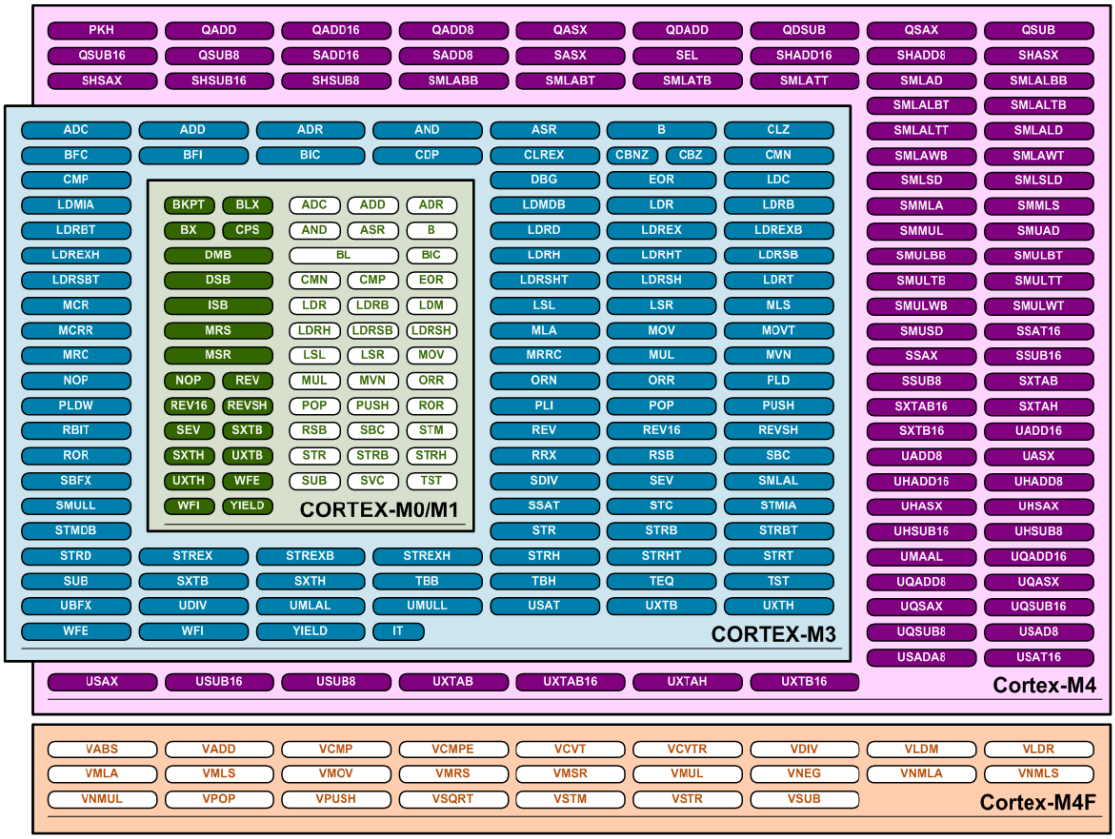

Figure 4: Instruction set overview of the Cortex-M family. The Cortex-M4 instruction set is a superset of the instruction set found in the Cortex-M3.

Single-Cycle Multiply Accumulate (MAC)

Multiply accumulate instructions are instructions in the form of

$$S = S + A \times B$$

These instructions multiply two registers and adds the result to a third register, called the accumulator. MACs are significant because of their central role in many common DSP algorithms. DSP algorithms will often execute these instructions many times inside their innermost loop. Take for instance the Finite Impulse Response (FIR) Filter. The equation for a normal FIR filter is given by

$$y_n = \sum_{i=0}^{N} c_i x_{n-i}$$

Where x is the input, y is the output and c represents the filter coefficients. The filter computation is simply a series of multiply accumulate operations! Consequently, being able to execute these instructions as fast as possible is beneficial to the computation time for the algorithm.

In many traditional microcontrollers there is no hardware support for MACs and the operation has to be decomposed into several instructions. On most of the modern MCUs that include these instructions, they take several cycles to complete. With the Cortex-M4, integer (and fixed-point) MACs are all executed in a *single CPU cycle*.

Single Instruction Multiple Data (SIMD)

SIMD instructions reduce computation time by performing multiple operations in parallel. The parallelization is realized by packing multiple values into the same registers and using the ALU to perform arithmetic on the packed values simultaneously. Since the general purpose registers in the Cortex-M4 core are all 32 bit wide, they can potentially hold two 16-bit values or four 8-bit values. Consider first a normal case, adding together two 32-bit integers.

```
ADD R0, R1, R2
```

This instruction will add the (32-bit) values in R1 and R2 and store the result in R0. The instruction takes one clock cycle to finish. Next, consider the following SIMD instruction.

```
SADD16 R0, R1, R2
```

This instruction will interpret both R1 and R2 as having two values. Bits [15:0] contain the low value and bits [31:16] contain the high value. The low values in R1 and R2 will be added and stored in the lower bits of R0. Similarly, the result of adding the high values will be stored in the higher bits of R0. This is shown schematically in Figure 5.

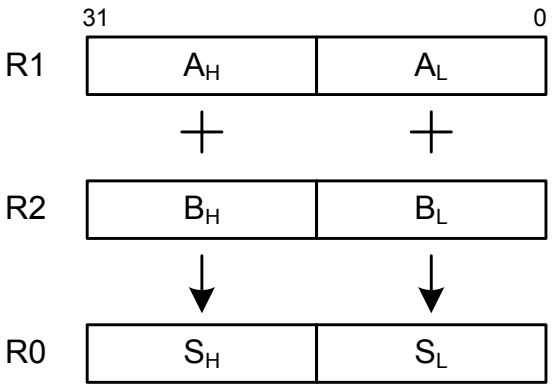

Figure 5: The SADD16 SIMD instruction can add two 16-bit variables in parallel

The SIMD instruction also takes one clock cycle to execute, but in this example we have performed two additions instead of one. The requirements for this optimization are that all values fit within 16 bits, and that the two additions are *independent* – meaning that the result of the first addition is not used in the second. If the algorithm can get away with using 8-bit values, it is possible to perform four additions in parallel.

The Cortex-M4 includes many such SIMD instructions that allow large computations to be done with one instruction. The instructions are all executed in a *single cycle*. A few SIMD instructions are also MACs, which lead to even more powerful expressions. As an example, the following expression can be done in one clock cycle. Here, A, B, C and D are variables, and S is the accumulator. The subscript denotes the bit width of each variable.

$$S_{64} = S_{64} + A_{16} \times B_{16} + C_{16} \times D_{16}$$

Saturating Instructions

Saturating instructions allows arithmetic operations to saturate instead of overflowing. As an example, consider adding two large numbers. If the sum ends up requiring more bits than available in the result register, we say that an overflow has occurred. With normal instructions the result register will 'wrap around' starting on zero after reaching the maximum value. This behavior requires software developers to insert overflow checks in their code in conditions where overflows can occur.

When using the saturating instructions, the result is instead clamped to the maximum value when it normally would overflow. Including these instructions can both speed up and simplify the algorithm by removing the need for overflow software checks.

In Figure 6, the difference is illustrated with two waveforms being generated using saturating and overflowing (normal) instructions. Generating the waveform in Figure 6a is possible with normal instructions but it would require additional overflow checks, which take up computation time.

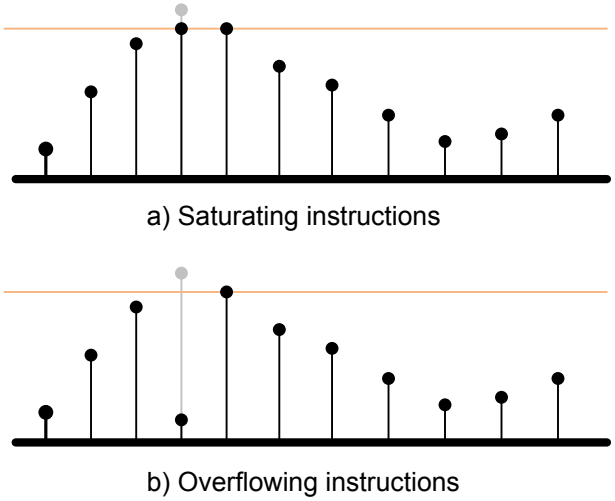

Figure 6: Waveform generation with saturating and overflowing instructions. With normal instructions, values that are too large to be stored will cause overflow. When using saturating instructions the values are automatically clamped to the maximum value.

Floating Point Unit (FPU)

Floating point numbers share a similar characteristic with the scientific notation. The numbers are stored using a fixed number of bits for the significant digits (also called the mantissa) which are scaled by an exponent. The format is standardized by IEEE 754, and Figure 7 illustrates the single precision (32-bit) format. 1 bit is reserved for the sign, 8 bits for the exponent and 23 bits are used to store the mantissa.

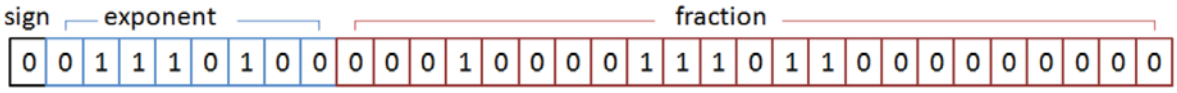

Figure 7: IEEE 754 Single Precision Floating Point Format

This format gives floating point numbers a large dynamic range. The IEEE 754 Single Precision format can store values between 1e-38 and 1e38. This is very convenient because the same number format can be used to store extremely large and extremely small numbers.

Conversely, the maximum value of a (signed) 32-bit fixed-point number, which uses 16 fraction bits, is only 32768. This limited dynamic range of fixed-point numbers can often lead to overflows, which forces developers to add extra checks for overflow. In addition, all code which uses fixed point numbers must implicitly keep track of how many bits are used for the fraction. Developing algorithms with fixed-point instructions therefore takes longer time than working with floating point values. The algorithms themselves can also take longer to execute because of the extra overflow checks.

The use of floating point numbers is therefore often desirable. However, using floating point numbers without a Floating Point Unit is extremely slow. Each operation must be decoded into

several instructions, which use integer operations to calculate the floating-point numbers. The compiler will do this automatically, but the performance is far from optimal.

Therefore, having an integrated Floating Point Unit is a great asset for developers. The EFM32 Wonder Gecko includes a single-precision FPU compatible with the IEEE 754 standard. With this unit, 32-bit floating point operations are executed in hardware. The integrated FPU enables developers to take advantage of the great dynamic range of floating point numbers and, with the speed of the FPU, create efficient algorithms. Table 1 lists a few important floating-point instructions and their cycle counts.

Table 1: Selected floating point instructions

Operation	Cycle count
Addition	1
Subtraction	1
Multiplication	1
Division	14
Load N floats	1 + N
Store N floats	1 + N
Multiply Accumulate	3
Square root	14

DSP Performance

To see the actual benefit of the DSP extensions, this chapter will compare the performance between Cortex-M3 and Cortex-M4 cores when running some common algorithms.

Floating Point Performance

The largest performance boost is obtained when working with floating point numbers. Figure 8 shows the computation time when running two common algorithms with floating point: a Fast Fourier Transform (FFT) and a Finite Impulse Response (FIR) Filter. In this graph, the running time for both processors is plotted (normalized with respect to the Cortex-M3 run-time). The FIR filter runs 16.6 times faster on the Cortex-M4 while the FFT is 11.4 times faster than on the Cortex-M3.

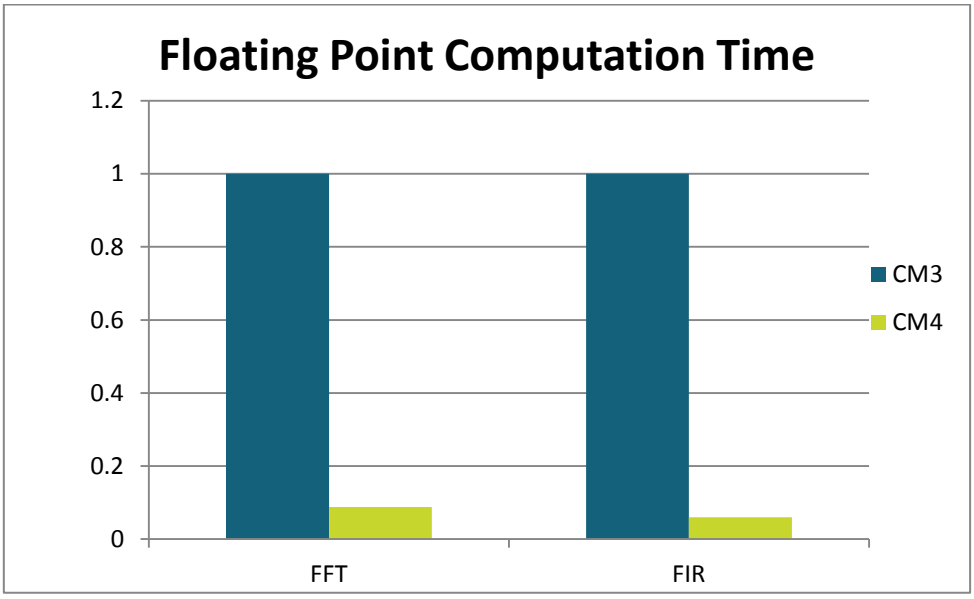

Figure 8: Floating point computation time comparison. Lower is better. FFT is run on 1024 samples, FIR filter is run on 320 samples and 30 filter coefficients. Tests are compiled with IAR EWARM 6.50 and speed optimizations enabled.

Fixed Point Performance

One can also expect a performance gain when using fixed-point values, due to the new DSP instructions. Figure 9 shows a benchmark of the same operations, but this time using fixed-point arithmetic (Q1.15 format). The FFT is here 2.2 times faster and the FIR is 3.8 times faster on the Cortex-M4. Here the performance increase is due to SIMD instructions and faster MACs present on the Cortex-M4.

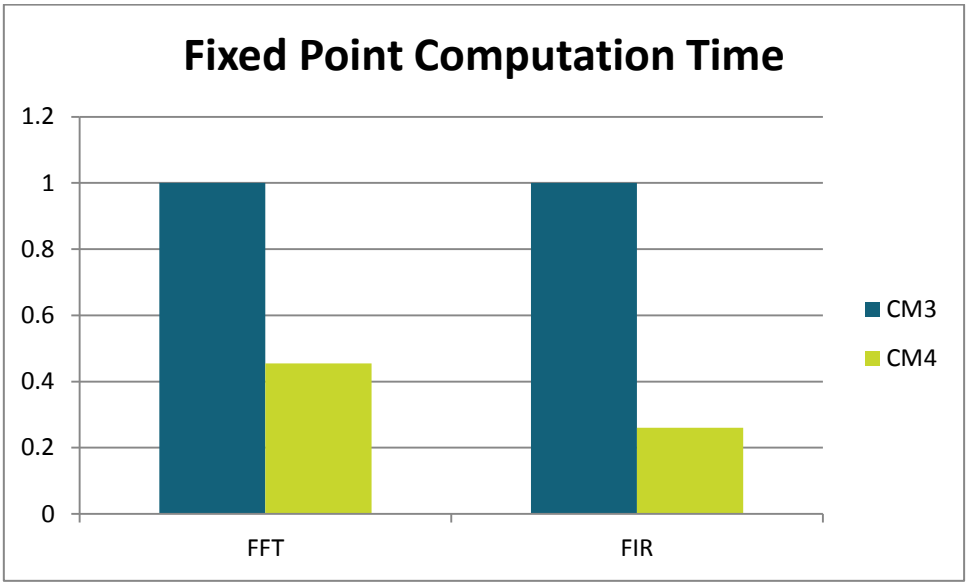

Figure 9: Fixed point (Q1.15) computation time comparison. Lower is better. FFT is run on 1024 samples, FIR filter is run on 320 samples and 30 filter coefficients. Tests are compiled with IAR EWARM 6.50 and speed optimizations enabled.

Matrix multiplication is another operation where fast MACs significantly reduce the computation time. For instance, matrix multiplication is important in monitoring applications that use multiple sensors to get a more accurate reading than using the sensors individually. A good example is combining GPS, accelerometer and gyro measurements to improve location accuracy. A common algorithm to correlate the sensor readings is the Kalman Filter, which relies heavily on matrix multiplications.

Figure 10 compares the computation time for calculating ten 10x10 matrix multiplications. The floating point algorithm runs 8.3 times faster and the fixed point algorithm improvement is also significant, about 1.8 times faster.

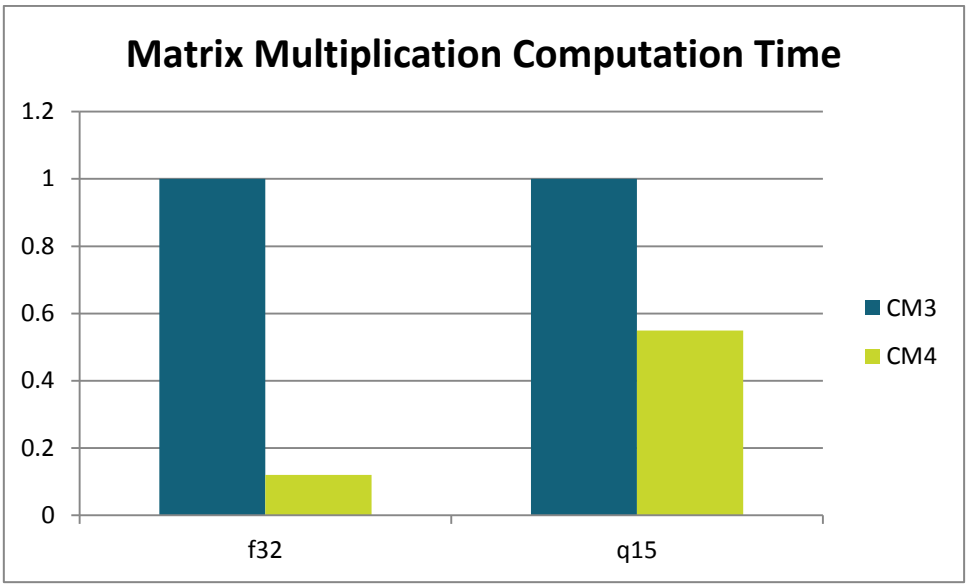

Figure 10: Matrix Multiplication computation comparison. Lower is better. Ten 10x10 matrix multiplications are performed with both floating point and fixed point (Q1.15) algorithms. Tests are compiled with IAR EWARM 6.50 and speed optimizations enabled.

Light Sense Example

To see the effect on power consumption, consider a simple application that measures the frequency of a strobing light source.[2] The application periodically measures the light intensity from a light sensor, calculates the Fourier transform and finds the peak value in the frequency domain. Figure 11 shows the comparison between running the algorithm on Cortex-M3 versus Cortex-M4. The algorithm executes much faster on the Cortex-M4 allowing it to spend more of its time in sleep. The average current consumption in this example was reduced from 550 µA to 170 µA when moving from Cortex-M3 to Cortex-M4.

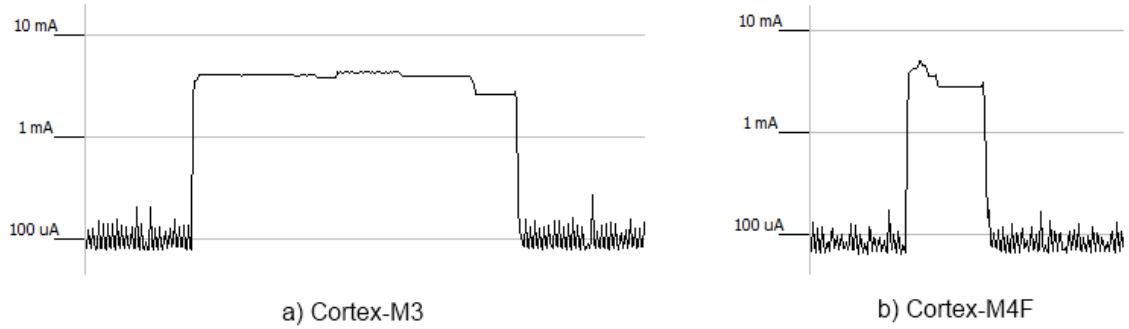

Figure 11: Current consumption during one 512-point FFT calculation. Images are taken from the energyAware Profiler.

[2] The actual application can be found in application note AN0051 (available through Simplicity Studio and energymicro.com) and uses the light sensors on STK3700/STK3800. Light intensity is sampled at 1024 Hz and the 512-point FFT is calculated twice per second. The MCU is in Deep Sleep between measurements.

Ease of Use

An important philosophy behind the EFM32 is that it should be an easy to use microcontroller. With Simplicity Studio, everything a developer needs, be it documentation, code examples, application notes etc. is only one click away.

The large portfolio of devices ensures that there is always a part that fits the application. The ARM core ensures software portability and the various EFM32 families are pin-compatible, making it painless to migrate to another device. The large ecosystem of tools and software for the ARM Cortex-M cores enables developers to choose what suits them best in any situation.

Writing code becomes even simpler for the Wonder Gecko with the integrated FPU. No longer is it necessary to use fixed-point algorithms; or constant range checking. With the Wonder Gecko the algorithms can be written directly with floating point values, taking advantage of the great dynamic range and precision.

The ARM CMSIS DSP library provides developers with all the basic building blocks needed when developing a signal processing application. The CMSIS DSP library contains basic vector and matrix operations, complex arithmetic and various filters and transforms, such as FIR, IIR and FFT. The implementations are fully portable between Cortex devices, and are highly optimized. The Cortex-M4 implementation naturally takes advantage of the enhanced instruction set and FPU to boost the performance of the algorithms.

Summary

The EFM32 Wonder Gecko provides both low energy consumption and fast signal processing in the same package. Taking advantage of the Cortex-M4 with integrated FPU, single-cycle MAC and SIMD instructions as well as saturating arithmetic, the Wonder Gecko achieves faster processing and longer sleep times. Designed for minimum leakage, low sleep mode consumption is guaranteed, even at high temperatures. High integration of peripherals designed for low power consumption offloads the CPU and reduces power consumption both in active and sleep modes.

More Information

Energy Micro provides both code examples and reference designs to help customers quickly get started with their applications. For more information about how to save energy with the EFM32 MCUs see the following application notes and white papers (all available through Simplicity Studio or the Energy Micro website):

- AN0027 Energy Optimization
- AN0051 Digital Signal Processing with the EFM32
- WP0001 EFM32 Introduction
- WP0002 Energy Debugging