Feature article

Reference: EM018

Energy debugging – the next step in MCU software optimisation

Knowing where your application is consuming resources is a crucial step in minimising energy usage

Øyvind Janbu, Chief Technology Officer, Energy Micro

Almost all applications running on microcontrollers today need to be far more aware of how they consume the precious resources available to them, in terms of memory, clock cycles and perhaps most importantly, energy. And while engineers may instinctively know how much power or energy an application should consume, it is only through closer examination that instincts are borne out. Typically, the examination is in the form of a simple current measurement averaged over a given time and then extrapolated to deduce a total lifetime expectancy for, say, a single battery cell.

With much greater emphasis now on configurable and programmable solutions, most typically in the form of a microcontroller, a larger proportion of the energy consumed can be directly attributed to the activity of the processing core and its many peripherals, which is the main reason why the semiconductor industry is now seeing a significant increase in the availability of 'ultra low-power' microcontroller solutions.

Traditionally, 8-bit or 16-bit devices have been used in the most energy sensitive applications because their cores are small, have relatively few gates and produce low levels of leakage current. Applications today however demand far more processing capability than the 8-or 16-bit cores can muster.

It has generally been assumed that the current drawn by a 32-bit core in its power down mode must simply be too great for energy sensitive applications. This today is a misconception. By harnessing the full range of low power design techniques now available, 32-bit cores can be implemented that offer low power modes as good as or better than 8-bit alternatives.

Energy Micro's EFM32 Gecko microcontroller is a new breed of energy friendly devices single-mindedly developed to minimise the product of current and time (i.e. the real energy) over all phases of MCU operation. Simply illustrated in

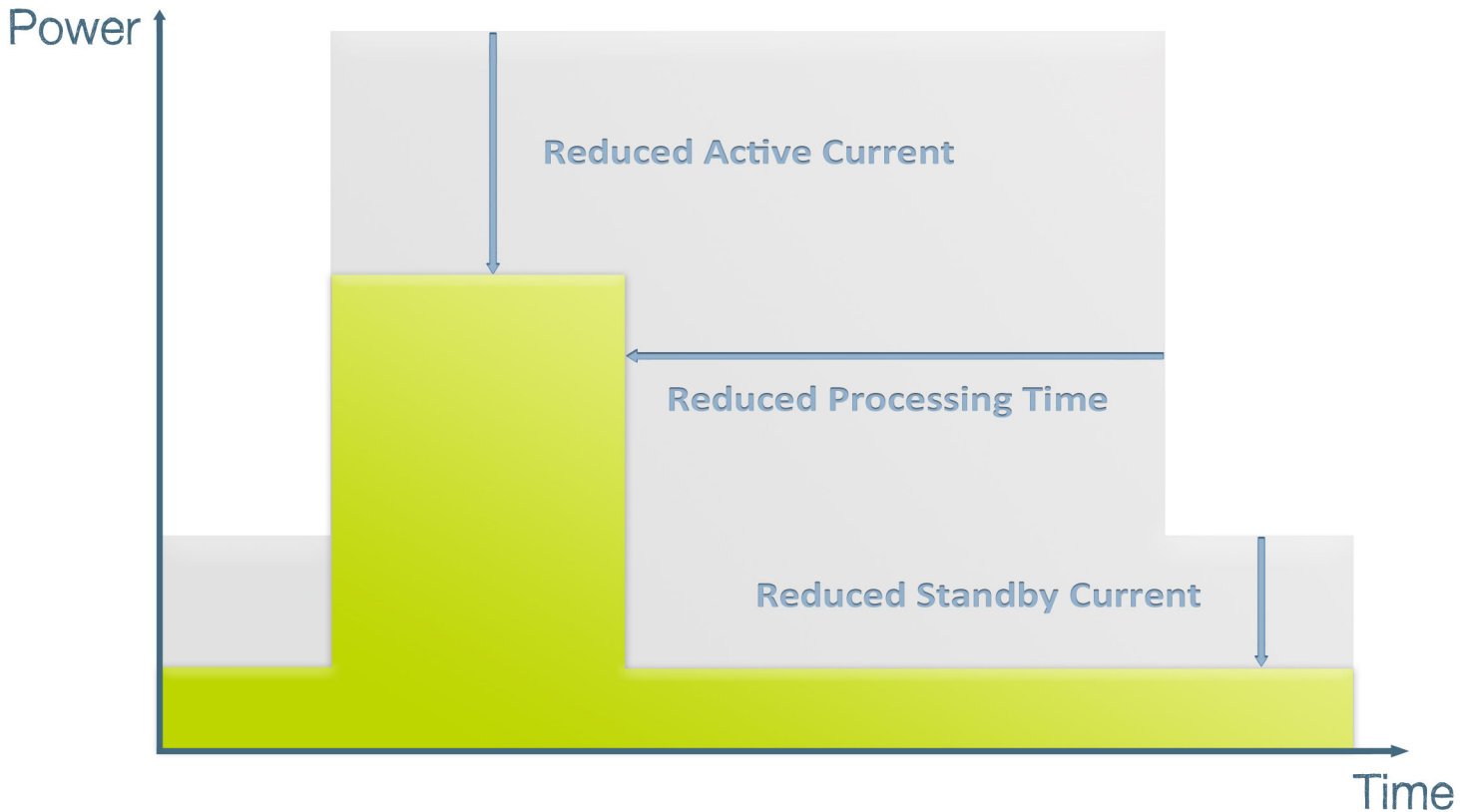

Power
Reduced Active Current
Reduced Processing Time
Reduced Standby Current
Time

Figure 1, compared to smaller processor cores, such an ARM Cortex™-M3 based device will finish a task quicker, enabling it to spend more time in low power modes, thereby lowering the average power consumption even further.

Illustrated in Figure 2, the Gecko has proved capable of consuming a quarter of the energy required by alternative 8-bit, 16-bit or 32-bit solutions. This performance is achieved in no small measure through a combination of a low energy peripheral set, a peripheral reflex system (enabling peripherals to function autonomously of the core), five distinct and graded low energy modes and very fast wake-up times.

In achieving the lowest possible energy consumption in a target application, what simply cannot be overlooked however is the crucial role application source code has to play. Source code needs to be engineered to make best use of low energy peripherals and best use of low energy modes if battery cell lifetime is to be maximised to the very fullest extent.

As source code bases grow in size of course it becomes increasingly difficult to identify, for example, while-loops that should be replaced with an interrupt service routine: a simple code oversight that could cause the processor to remain fully active while waiting for an external event instead of going into an energy saving sleep mode.

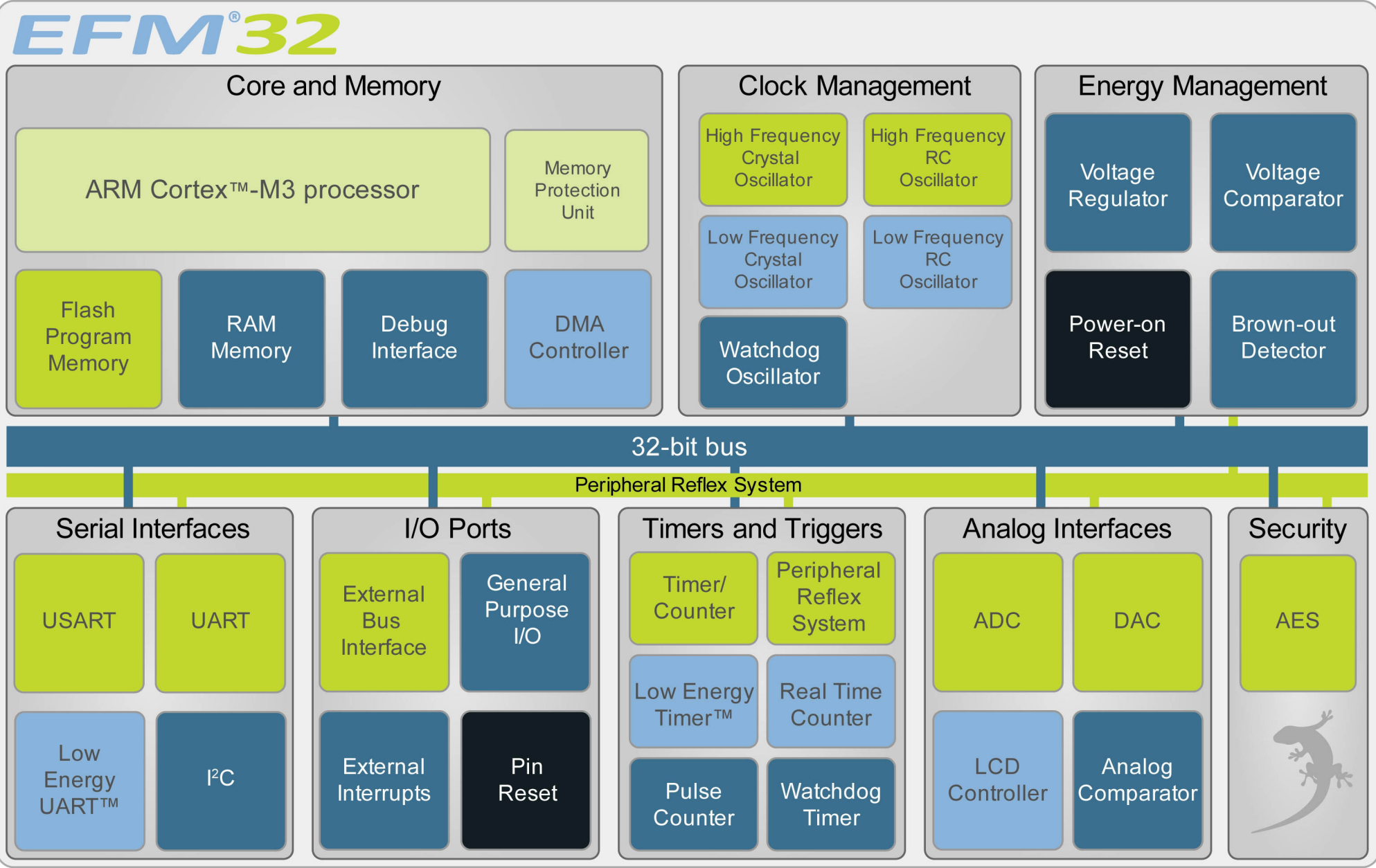
This kind of pseudo-random event can easily be missed when examining the code or testing under ideal conditions, and is difficult to capture during soak test. Similarly, identifying sections of code that consume a disproportionate amount of energy may be impossible to identify from a pure code listing alone, even for the most experienced engineer.

While a multimeter reading or oscilloscope trace may produce an average level of energy consumption over a given time it cannot identify the current consumed by specific events. Likewise, a logic analyser can display when and how many times a particular routine is executed but it cannot correlate that with power surges.

Through the use of innovative technology, Energy Micro has overcome these limitations by developing a solution that provides not only the level of instantaneous energy being used but also correlates that information with the actual code being executed at the time.

The energyAware Profiler is an ‘energy debugging’ tool for the PC that makes use of the dedicated Advanced Energy Monitoring (AEM) system present on existing EFM32 Gecko development kits. While the AEM is capable of displaying the application’s real-time current consumption on the development kits’ on-board LCD display, as shown in Figure 3, the real power of energy debugging is realised when the Profiler software is used.

The software runs under Windows and interfaces with the development kit via its USB interface. Basic data transferred from the development kit allows the PC to display a real-time energy profile of the application code running on the target MCU.


EFM32
Core and Memory
ARM Cortex™-M3 processor
Memory Protection Unit
Flash Program Memory
RAM Memory
Debug Interface
DMA Controller
Clock Management
High Frequency Crystal Oscillator
High Frequency RC Oscillator
Low Frequency Crystal Oscillator
Low Frequency RC Oscillator
Watchdog Oscillator
Energy Management
Voltage Regulator
Voltage Comparator
Power-on Reset
Brown-out Detector
32-bit bus
Peripheral Reflex System
Serial Interfaces
USART
UART
Low Energy UART™
I²C
I/O Ports
External Bus Interface
General Purpose I/O
External Interrupts
Pin Reset
Timers and Triggers
Timer/ Counter
Peripheral Reflex System
Low Energy Timer™
Real Time Counter
Pulse Counter
Watchdog Timer
Analog Interfaces
ADC
DAC
LCD Controller
Analog Comparator
Security
AES

A default configuration displays energy levels over time, allowing an engineer to identify specific areas of concern where the energy used is perhaps higher than expected. Extrapolating this over time can also give a more accurate indication of life expectancy for a battery-powered application, than estimates based on best- and worse-case figures in a datasheet.

When used with the energyAware Profiler, the AEM system uses an ARM-based serial interface to collect additional information from the application. The data passed to it is decoded by the AEM system's hardware which is then passed on to the PC. The non-intrusive nature of this activity means that the energy profile of the target isn't altered by it in any way.

The additional data includes important debug information, including the Program Counter, which allows the energyAware Profiler to identify the actual source code being executed at a given moment in time as shown on the energy graph. This instantly gives the engineer a pointer to any part of the program that causes high energy consumption, allowing the code to be optimised to lower the overall energy usage. This is illustrated in Figure 4.

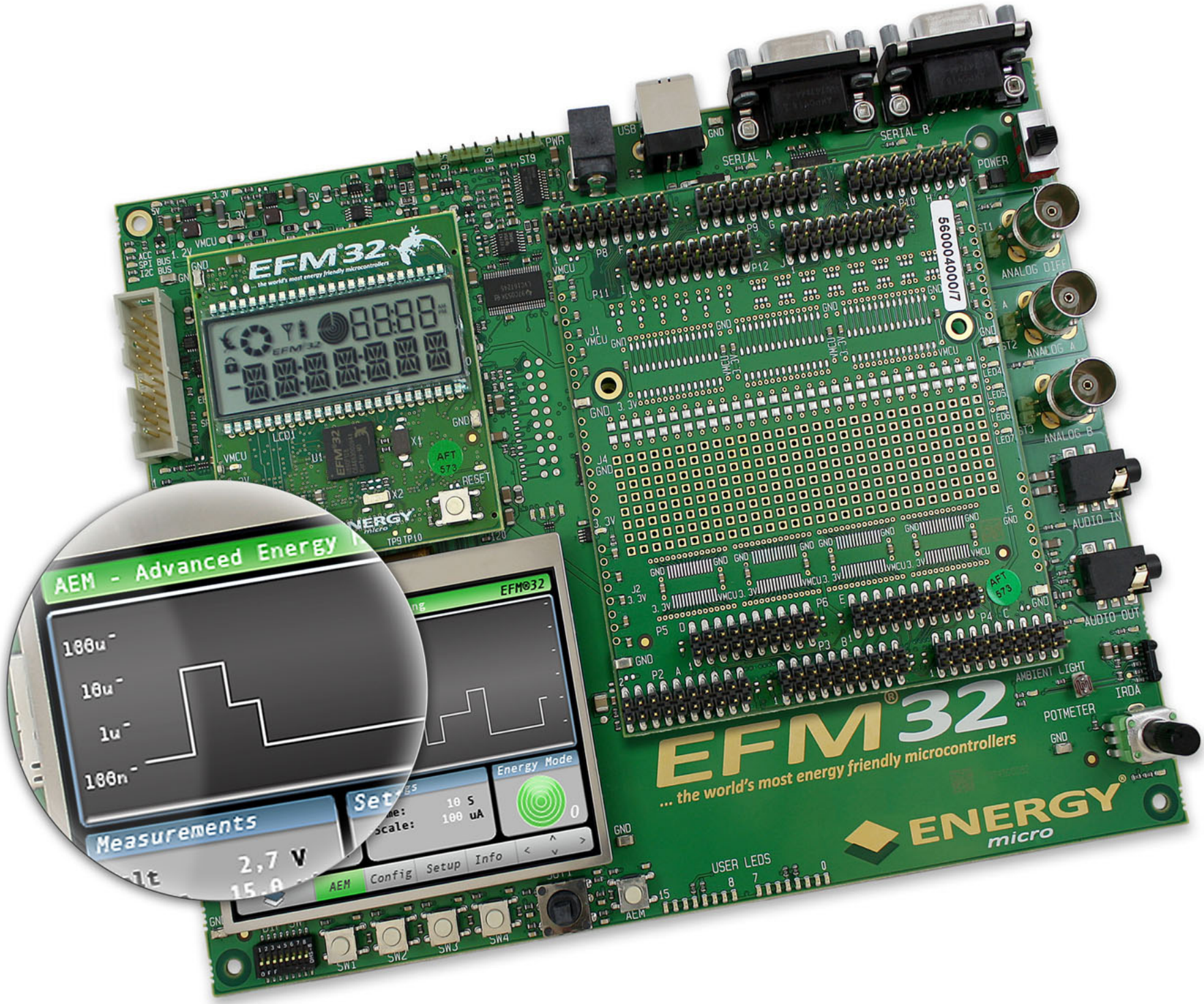
The graph represents energy usage by the width and height of the trace; magnitude against time. It follows therefore that isolated peaks – which would be easily identified using an oscilloscope to monitor the power supply's current – may not actually warrant further investigation, while long periods of relatively little activity could in fact represent idle loops that could be easily replaced with an interrupt-driven event, which would allow the device to enter an energy-saving sleep mode in the interim.

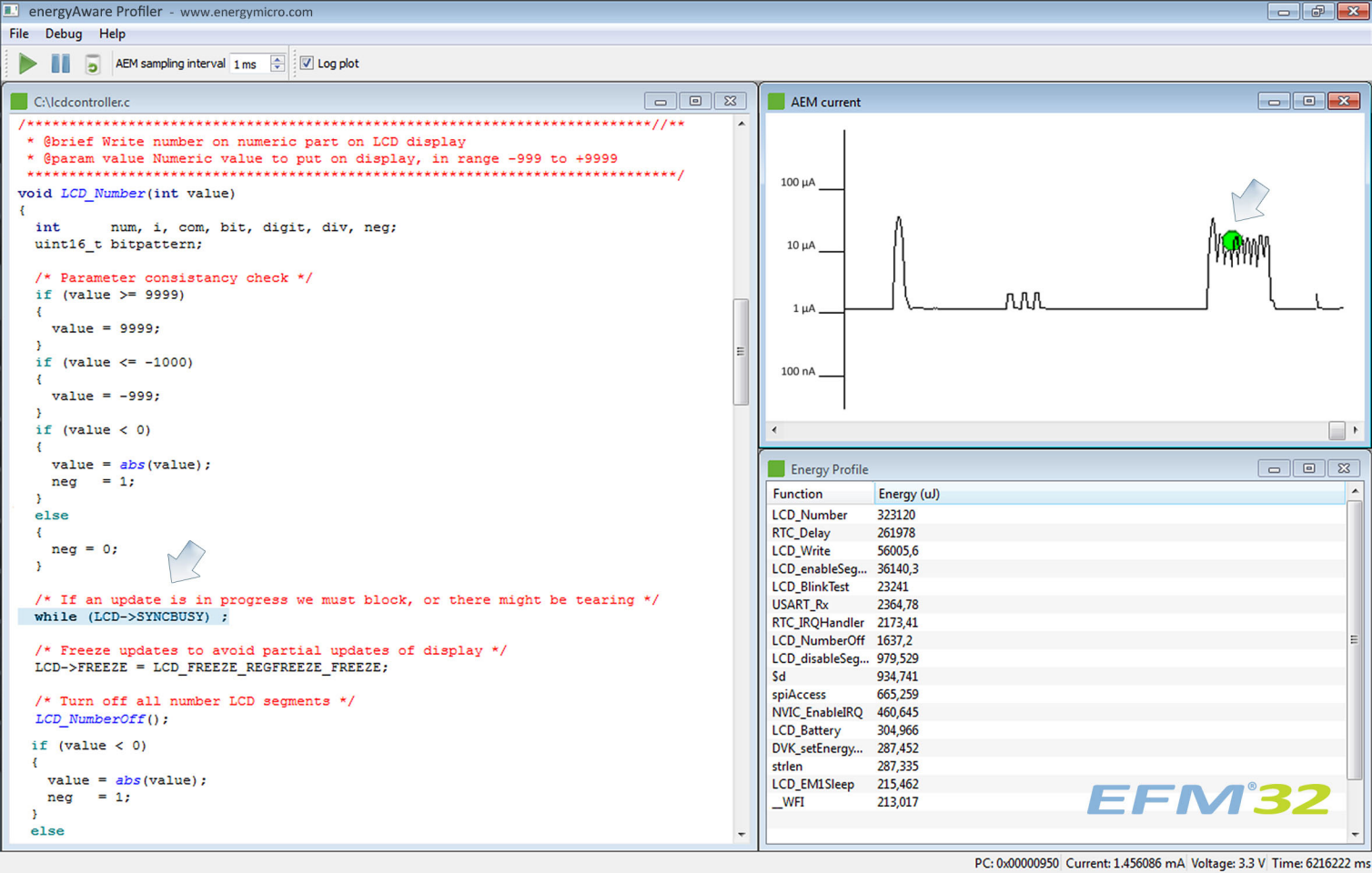
By seeing this information graphically represented and instantly correlated to the source code, the engineer is quickly able to identify, dismiss and prioritise specific routines within a program that could represent unnecessary energy usage. This can easily translate into an order of magnitude lower energy consumed and, therefore, a much more efficient application.

It is commonplace for engineering teams to refer to datasheets when establishing the nominal power for a device or application. However as intimated earlier, energy and power are simply not the same; many low power devices use more energy because they are active over a long period of time. For this reason, the time axis for power management cannot be ignored but is rarely accessible in a reliable way.

Clearly in this scenario the software's profile is crucial. Unfortunately most software engineers aren't aware of the extent to which code can be overtly 'energy efficient'. This is not a criticism, it is an observation; software isn't seen as inherently resource hungry beyond the conventional terms of clock cycles and memory. However, today every clock cycle used is power spent and minimising that is a key challenge for engineers developing ultra-low energy applications.

Furthermore while minimising clock cycles directly relates to less energy used, optimising exactly when the clock cycles are used also gives a better overall


EFM32
... the world's most energy friendly microcontrollers
ENERGY micro
AEM - Advanced Energy
Measurements
2,7 V
Energy Mode
AEM
Config
Setup
Info
USER LEDS
SERIAL A
SERIAL B
USB
POWER
ANALOG A
ANALOG B
AUDIO IN
AUDIO OUT
IRDA
AMBIENT LIGHT
POTMETER
RESET


energyAware Profiler - www.energymicro.com
File
Debug
Help
AEM sampling interval
1 ms
Log plot
C:\lcdcontroller.c
/***//**
* @brief Write number on numeric part on LCD display
* @param value Numeric value to put on display, in range -999 to +9999
***/
void LCD_Number(int value)
{
int num, i, com, bit, digit, div, neg;
uint16_t bitpattern;
/* Parameter consistancy check */
if (value >= 9999)
{
value = 9999;
}
if (value <= -1000)
{
value = -999;
}
if (value < 0)
{
value = abs(value);
neg = 1;
}
else
{
neg = 0;
}
/* If an update is in progress we must block, or there might be tearing */
while (LCD->SYNCBUSY) ;
/* Freeze updates to avoid partial updates of display */
LCD->FREEZE = LCD_FREEZE_REGFREEZE_FREEZE;
/* Turn off all number LCD segments */
LCD_NumberOff();
if (value < 0)
{
value = abs(value);
neg = 1;
}
else
AEM current
100 µA
10 µA
1 µA
100 nA
Energy Profile
Function
Energy (uJ)
LCD_Number 323120
RTC_Delay 261978
LCD_Write 56005,6
LCD_enableSeg... 36140,3
LCD_BlinkTest 23241
USART_Rx 2364,78
RTC_IRQHandler 2173,41
LCD_NumberOff 1637,2
LCD_disableSeg... 979,529
$d 934,741
spiAccess 665,259
NVIC_EnableIRQ 460,645
LCD_Battery 304,966
DVK_setEnergy... 287,452
strlen 287,335
LCD_EM1Sleep 215,462
__WFI 213,017
EFM32
PC: 0x00000950
Current: 1.456086 mA
Voltage: 3.3 V
Time: 6216222 ms

energy profile. It is clear, therefore, that a functionally correct program isn't necessarily optimised for energy efficiency by design.

The energyAware Profiler technology developed for the EFM32 Gecko MCU has a dynamic range from 100nA to 50mA, and fine-tuning of the application should be carried out at the same time as the functional debugging to maximise the results from the development time.

Energy debugging and software profiling is becoming more crucial in ultra-low power applications and technologies. While the EFM32 Gecko MCU is an inherently low power technology, maintaining lower energy usage is intrinsically linked to the application's performance over time. This makes it transient and subject to application-specific conditions and as such it is difficult to simulate. While a datasheet may give an engineer a good idea of the amount of power a device uses under given conditions, it isn't until the application is actually running that the datasheet figures are really tested.

With experience and enough time, engineers would be able to develop code that is functionally correct and energy-optimised, but with the help of energyAware Profiler that time and expertise is reduced significantly, making it accessible to all engineering teams whatever their time pressures. The use of low power technology, coupled with an energy profiling solution, means engineering teams are now better equipped to tackle the challenge of designing truly low energy solutions.

Figure captions:

Figure 1: An energy friendly MCU core saves power in several distinct areas over a complete wake/operate/return-to-sleep cycle. The blue area indicates the energy saved by a more capable 32-bit core completing a task in fewer cycles than an 8-bit core would require and consuming less current in both active and sleep modes.

Figure 2: Chip architecture for Energy Micro's ARM® Cortex™-M3 based EFM32 Gecko microcontroller.

Figure 3: Development kit for the energy friendly EFM32 Gecko microcontroller complete with Advanced Energy Monitoring.

Figure 4: energyAware Profiler provides three simultaneous views; a graph of real-time application current consumption, an object code listing and an energy profile of individual application functions.

About the author:

Øyvind Janbu co-founded Energy Micro in 2007 and has been the company's CTO since its inception. He has more than 12 years semiconductor industry experience, covering research, development and management. Øyvind previously held key positions with Chipcon, working on digital and analog design, embedded microcontroller design and ZigBee software. He was a

contributing editor for the IEEE 802.15.4-2006 wireless PAN standard and holds an MSEE from NTNU in Trondheim, Norway.

For further information and reader enquiries:

Øyvind Borgan, Energy Micro AS, PO Box 4633, Nydalen, N-405 Oslo, Norway

Tel: +47 23 00 98 00 o.borgan@energymicro.com
Fax: +47 23 00 98 01 www.energymicro.com

For further information or to discuss feature article opportunities:

Rob Davies, Publitek Limited, 18 Brock Street, Bath, BA1 2LW, United Kingdom

Tel: +44 (0)1225 470 000 rob.davies@publitek.com
Fax: +44 (0)1225 470 047 www.publitek.com

About Energy Micro

Energy Micro develops, markets and sells the world's most energy friendly microcontrollers, based on the industry leading ARM® Cortex™-M3 32-bit architecture. The company was founded in 2007 by experienced semiconductor professionals with previous expertise from Chipcon, Texas Instruments, Atmel and Nordic Semiconductor. More information on Energy Micro is available at www.energymicro.com