Optimize MCU Code for Low Energy Operation

Rasmus Christian Larsen, Energy Micro, www.energymicro.com

Once you've selected the right low power microcontroller, how important is your application code in achieving energy saving and efficiency goals? And how do you go about optimizing it for ultra low power operation? Answering such questions has always been difficult, since correlating current consumption with software code has traditionally been a somewhat inexact and cumbersome task.

Software has never really been identified as an 'energy drain', but energy is being consumed on every clock cycle, so it really deserves closer attention, particularly in today's battery-driven, energy sensitive applications. We need to go much further than just maximizing the amount of time a microcontroller spends in a sleep mode and limiting code size to reduce memory usage.

Figure 1. The EFM32 Gecko microcontroller is made for energy sensitive applications.

Consider a simple while loop which would have been better replaced with an interrupt service routine, it's the kind of basic oversight that causes a processor to stay active instead of entering a sleep mode. Identifying such energy bugs in field or burn-in tests is just too late and costly. We need for embedded system development to become a 3 phase cycle; of hardware debugging, software functionality debugging and software energy debugging. Having full control of the hardware surrounding the microcontroller and the overall software and peripheral usage inside are crucial factors in reducing a system's total energy consumption and maximizing battery lifetime.

Traditionally, measurements of a hardware setup made by an oscilloscope or multimeter and entered into spreadsheets can be extrapolated to provide a

reasonable estimation of an application's battery life expectancy. The approach though presents no real correlation between current consumption and code. Equally, while a logic analyzer can be employed as more of code view tool, it again lacks the correlation. Use all the equipment together of course and the picture becomes a little clearer but setup time, accuracy and its repeatability are not insignificant issues.

Three things are actually needed to produce energy optimized MCU designs quickly and economically: the right microcontroller, development kits that can reliably measure a prototype's current consumption and supporting software tools that can clearly show the correlation between the current being consumed and the code that's running. With all these elements in place, no special equipment set-up is required, and accurate measurements can be obtained in a way that significantly reduces design cycle time.

To address the challenge of ultra low power embedded system design, the microcontroller should be characterized by very low active power consumption, ultra low standby current and very fast wake-up and task execution times while providing a selection of low energy peripherals functions that can operate autonomously of the processor core enabling it to stay asleep for longer. One such example is the ARM Cortex-M3-based EFM Gecko microcontroller from Energy Micro, which offers a range of low energy operating modes is offered, from the 20nA EM4 'shut off' mode through to the 160µA/MHz 'run mode'.

The Right Hardware Tools

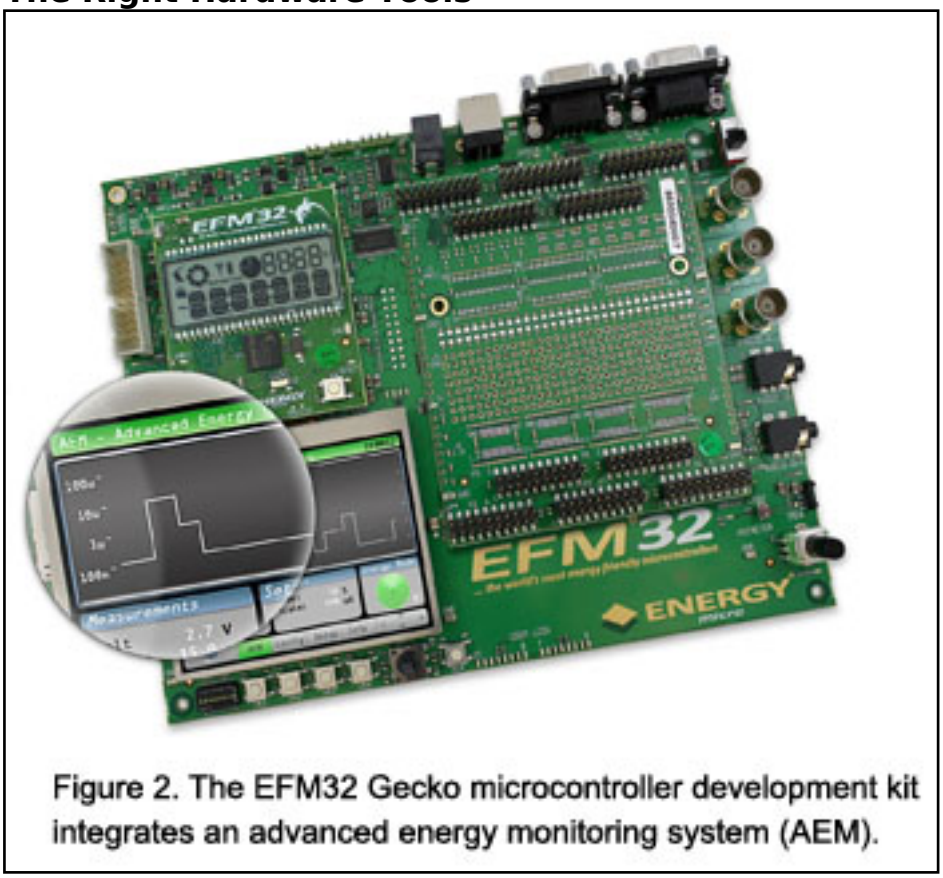

Figure 2. The EFM32 Gecko microcontroller development kit integrates an advanced energy monitoring system (AEM).

An energy monitoring system on the development kit can enable developers to keep track of their application's current consumption in real time using an on-board

LCD. The Advanced Energy Monitoring (AEM) system on the EFM32 development kit samples current flowing through the MCU power rail and immediately displays it along with voltage and time data. Energy consumption is simply the area below the current trace.

The dynamic range of the onboard AEM system is from a minimum of 100 nA to a maximum of 50 mA and so is able to detect changes in current consumption in even the most energy sensitive applications. Note that it measures not only the current drawn by the MCU but also the current drawn by other prototype system components, as long as they are supplied from the power rail monitored by the AEM.

The Software Tools

Importantly, an MCU should have the ability to be set to output program counter samples through the core's standard serial wire output (SWO) pin. Using the EFM32, this information can be sent along with the development kit's AEM current, voltage and timing data over USB to a Windows based software tool called the energyAware Profiler. Uploaded with the MCU's code object file (*.out) (the Profiler software then has all the information it needs to directly correlate energy consumption with code and thereby implement 'energy debugging'.

There are three main windows to the software tool: a code listing, a function listing and a current graph. By clicking on any point on the current graph, the corresponding portion of code giving rise to that current consumption is highlighted in the code listing. The function listing details the energy that each function consumes and the percentage of total system energy consumption for which it accounts.

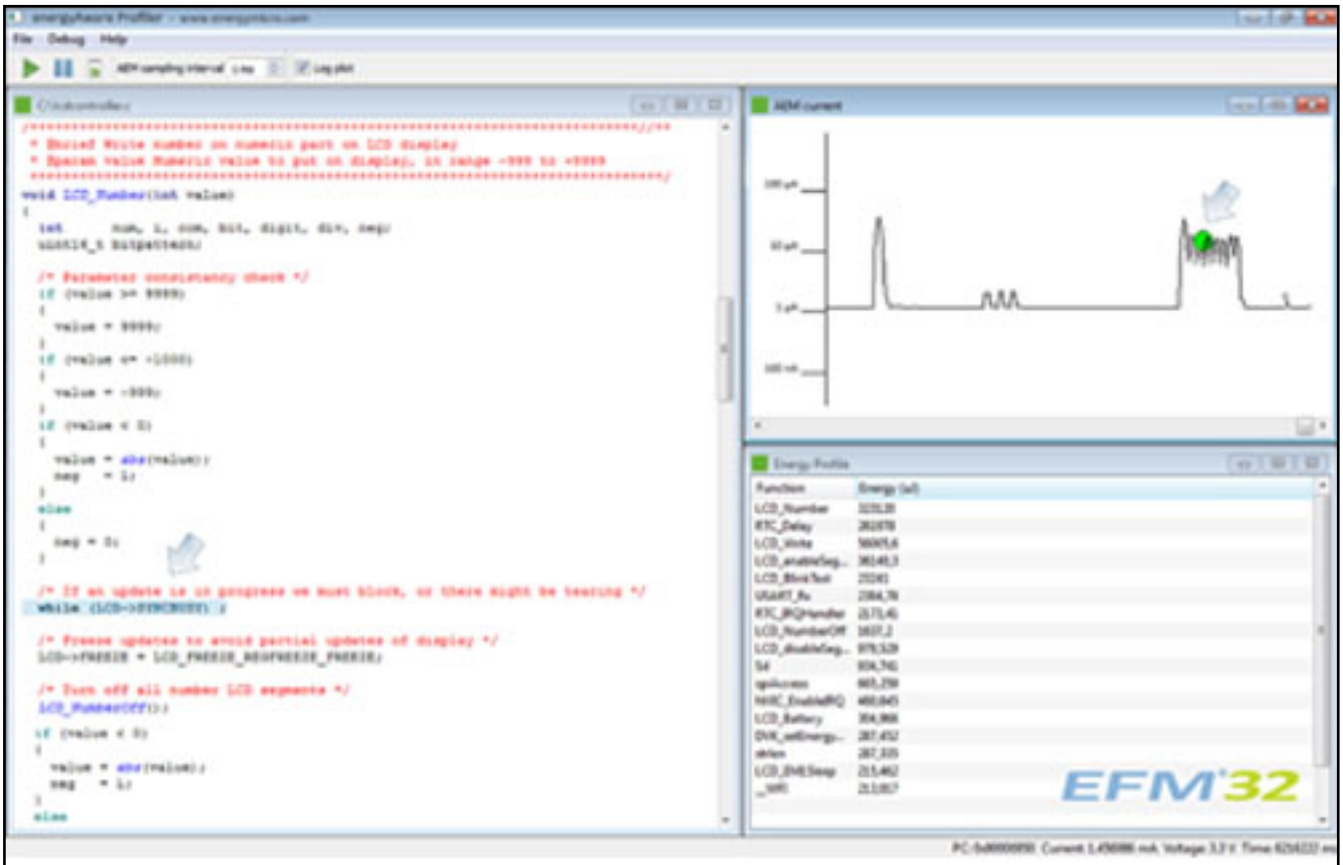

Figure 3. This profiler tool shows the correlation between current consumption, C code and the energy used by a particular function. Clicking on a current peak reveals the code associated with it.

As well as showing the code associated with the current graph at a certain point in time, the code listing will also display a further data line revealing which function is running, the last PC sample, current, voltage and at what point in time the code was executed. Average current can also be calculated by highlighting a portion of the current graph. It is also possible to enable IRQ (interrupt request) pinpointing that shows a listing of different IRQs in different colors for easier identification.

Until now, the energy footprint of a product was not really known until the very end of the design cycle, by which time optimization of the code for energy efficiency was really impractical. As result, an oversized battery was often required, meaning an increase in product size, cost and complexity. A move to energy debugging tools enables designers to identify and remove energy bugs at the early stages of a design process without penalizing development time.