

Managing Energy Savings in Real Time

The use of power-aware MCU features along with power-aware software and development tools can lead to significant optimization of low power consumption.

Raman Sharma, Energy Micro and John Carbone, Express Logic

Picking the right microcontroller and development environment for applications that need to support many years or even decades of operation off a battery is not easy. A real-time operating system can take away issues with asynchronous task handling and improve power management. And the latest development tools make it easy to perform real-time debugging of application code to expose energy bugs. Still, the biggest challenge remains in finding the correct microcontroller in a steadily increasing “ultra low power microcontroller” market.

The ARM Cortex-M3 architecture released a wave of new silicon vendors. Although a dramatic step up from 8- and 16-bit performance, the new 32-bit CPU immediately found itself surrounded by legacy peripherals incapable of meeting energy conscious requirements. For a host of battery-powered applications and energy-sensitive products in sectors such as metering, medical devices, wireless communication and security equipment, more advances in MCU design and development tools are needed to meet energy efficiency and processing power demands.

With a fresh approach to energy-friendly design, Energy Micro has equipped the EFM32 microcontrollers, an ARM Cortex-M3-based design, with energy-efficient peripherals smart enough for the low-power 32-bit core architecture. The supporting software and hardware tools introduce the capability to perform real-time energy debugging, and embedded designers have a device capable of consuming a quarter of the energy needed by incumbent 8-, 16- and 32-bit MCUs. Thanks to these changes, product designers are now able to significantly reduce the cost and size of the battery powering their product.

With modern microcontrollers becoming ever more power-conscious and incorporating various power-saving modes of operation, ARM’s Cortex-M3, widely adopted for microcontroller designs, employs some of the most advanced power-saving technology seen among 32-bit processors. Energy Micro’s Cortex-M3-based EMF32 adds low-power peripherals and offers a range of power-saving modes for various operational situations. The industry as a whole is focusing on power conservation, and the EMF32 architecture can serve as an example of industry trends in this direction.

ers the lowest possible energy usage. Writing embedded code requires as much care as engineering the hardware. Software is not usually seen as an energy drain, but every clock cycle consumes energy, and minimizing this becomes fundamental to reducing overall system consumption.

Optimizing software begins by choosing the right tools. Tools able to identify and remove energy drains at an early stage of prototype development can significantly reduce the overall energy consumption of the end product.

A good compiler starts energy-efficient design by generating smaller code that uses less memory. Since such memory is usually high speed, expensive and consumes more power, reducing the demands on memory yields multiple gains. A good compiler also produces code that runs more quickly, speeding entry into power-saving mode sooner than code taking longer to perform its functions.

An efficient operating system is the next pivotal factor. An efficient RTOS performs requested services more quickly, taking the program into power-saving mode sooner. A good RTOS achieves real-time responsiveness and operation with a less powerful processor, which normally results in lower power demands.

While these capabilities are fundamental to good compiler and RTOS technologies, some RTOSs go the second mile, providing additional capabilities for managing power resources. A properly equipped RTOS informs the application

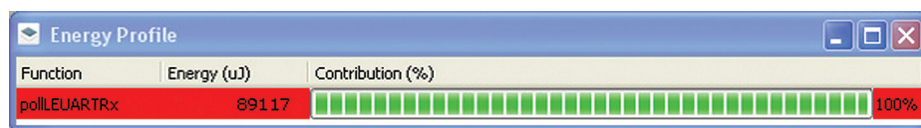


FIGURE 3

Function contribution to energy consumption.

how long before the next scheduled event. The application then knows how long it may be idle, enabling it to enter the deepest low-power state practical, given the wake-up time required to emerge from power-down mode, and the maximum duration of potential power-saving time. This ensures that, if the period of sleep or power-down is too brief, the overhead needed to reawaken does not waste all the gains of the low-power mode or even increase power consumption.

RTOS technology, such as Express Logic's ThreadX, lets developers select appropriate power modes with this "time budget" in mind, avoiding a costly "thrashing" in and out of low-power mode. Table 1 offers the wake-up time for each mode.

As you can see, the power-saving modes that consume the least power also require the longest time for the system to wake up. It would not make sense, for example, to enter EM4 if a scheduled event were to occur within 100 microseconds, since 160 microseconds would be required to reawaken. Modes EM2 or EM3 offer a better combination of saving and wake-up time and can be employed beneficially, even with just a few microseconds of time available.

It's also necessary to consider the power expended in start-up. This, too, might affect the decision of which mode to enter, coupled with the time it can expect to remain in that mode.

Additional Efficiency Resources

Energy-friendly embedded systems development can be seen as a three-stage cycle: hardware debugging, software functionality debugging and software energy debugging.

For hardware debugging, the most common way to track how much energy a system draws is by sampling the current over a certain period followed by averaging and extrapolation to longer time periods. This kind of measurement can be done using a multimeter or oscilloscope, but it is not possible to relate the results to code routines. On the other hand, a logic analyzer can be used to keep track of the routines but cannot relate that to the energy consumption.

Given the criticality of software efficiency, software energy monitoring tools are now available. In addition to using the RTOS to determine how long a period of power-reduction can last to direct the

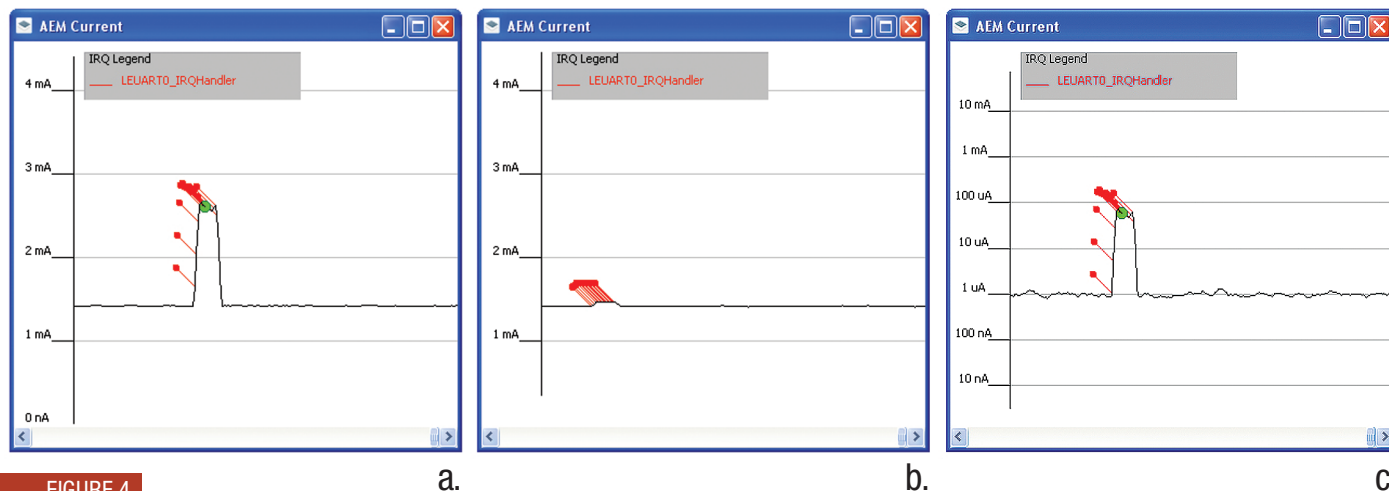


FIGURE 4

LEUART RX Interrupt with LEUART TX polling (a), EFM32 in Sleep Mode between received bytes (b), and EFM32 in Deep Sleep Mode (c).

EFM32 running real application from Flash memory with 3V power supply	EM0 Run Mode	EM1 Sleep Mode	EM2 Deep Sleep Mode	EM3 Stop Mode	EM4 Shutoff Mode
Current consumption	180 μ A/MHz	45 μ A/MHz	0.9 μ A	0.6 μ A	20 nA
Wake-up time	0	0	2 μ s	2 μ s	160 μ s
Wake-up events	Any	Any	32 kHz peripherals	Async IRQ, I2C slave, Analog Comparators, Voltage Comparator	Reset
CPU	On				
High frequency peripherals	On	On			
Low frequency peripherals	On	On	On		
Full CPU and SRAM retention	On	On	On	On	
Power-on Reset/Brown-out Detector	On	On	On	On	On

TABLE 1

Energy saving for various Low-power modes of operation.

choice of low-power mode, monitoring tools, such as Advanced Energy Monitoring (AEM), continuously measure current consumed. This information is correlated with the program counter sampling and allows real-life use cases to be debugged for low power operation.

Energy debugging software tools, such as the energyAware Profiler, enable the user to identify the source code being executed at a given moment in time as shown by an energy graph (Figure 1). The engineer can instantly pinpoint any part of the program causing high-energy consumption, ensuring code optimization and energy savings can be managed more closely.

An Example Application for Reducing Energy Consumption

Using a LEUART module, let's take a look at how the different views in the energyAware Profiler work together with the autonomous peripherals on the EFM32 to reduce energy consumption and increase battery life in the widest range of applications. In this example, the LEUART module enables UART communication up to a 9600 baud rate while keeping energy consumption to a minimum.

A common way of getting data from the reception buffer is to poll it until you get valid data and then read the buffer. By doing this, the processor must be in a run mode, which results in relatively high current consumption.

The profile for such a loop, shown in Figure 2a, is a constant current consumption of 3.33 mA. By clicking on the graph, the function causing the drain is highlighted.

```
void pollLEUARTRx(void)
{
    while ( !( LEUART0 -> STATUS &
LEUART_STATUS_RXDATAV ) );
}
```

The highlighted code line is the polling loop, which checks if the buffer has received any valid data. The Profiler also shows each function and how much each contributes to total energy consumption. In this case, the only function in the code is *pollLEUARTRx()*, which accounts for all the energy consumption (Figure 3).

A common workaround to avoid polling the RX buffer is to enable RX interrupts and set the MCU in sleep mode. If we enable the LEUART RX interrupt and put the EFM32 in Sleep Mode (shut off the CPU), it is easy to achieve significant energy savings.

As we shut off the processor, the current drops to 1.40 mA (Figure 2b). Now, when the LEUART receives a frame of bytes, it wakes up and transmits the data back through the TX buffer.

When the interrupt is triggered, the current spikes to around 2.5 mA, displayed by the Profiler (Figure 4a). By clicking on the graph, it is possible to detect another common mistake when using UART communication.

```
void pollLEUARTTx(void)
{
    while ( !( LEUART0 -> STATUS &
LEUART_STATUS_TXC ) );
}
```

After sending the data, a while loop waits for the transmission to finish. The loop can be replaced by an interrupt that wakes up the processor once the transmission is finished, and the current consumption will again be reduced (Figure 4b).

Now the processor goes into Sleep Mode between each frame byte. It is not necessary to poll the buffer to know when the transmission is finished. Replacing the loop with an interrupt routine is a much more elegant and energy-friendly solution, shown with the different profiles of the two approaches.

Because the LEUART module on the EFM32 MCUs is functional in a Deep Sleep Mode, the low-frequency oscillators are available and clocking the LEUART. If we repeat the above example putting the EFM32 in Deep Sleep Mode, the energy consumption drops to μ A levels.

If we change the Profiler to logarithmic scale, we see the current dropping to 1 μ A in Deep Sleep Mode and 80 μ A when receiving the frame (Figure 4c).

The improved energy savings from the traditional approach to this configuration is a factor of over 1000. Clearly, energy efficient has come a long way, now offering energy-efficient architecture such as the EFM32, compilers and RTOSs tuned to maximize energy-efficient opportunities, along with profilers that help developers pinpoint where their program is wasting energy using traditional methods. Combining these technologies ensures that you can reduce power consumption, extending your device's battery life for significantly longer infield time. ■

Energy Micro.
Oslo, Norway.
+47 23 00 98 00.
[www.energymicro.com].

Express Logic
San Diego, CA.
(858) 613-6640.
[www.expresslogic.com].