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.

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
Hardware Reduces Energy Consumption

To deliver true energy-friendly products, microcontroller makers have to consider several factors, and with a finite amount of charge available from a battery cell, it is how the MCU uses energy—power over time—that makes the difference. Examining embedded applications shows that many systems spend up to 99% of their time waiting for a pin change, a timer to match a compare value or to receive data. Looking at these wait states where the CPU could be sleeping, it becomes evident that minimizing the power and time is as important during sleep as in active periods.

Before we look at the sleep modes, consider the period when the CPU is awake and processing data. To achieve higher performance, processors can either work hard or work smart. A 32-bit CPU can finish tasks or crunch more data in less time, and the system can return to a low-energy sleep mode sooner. By taking a task-oriented approach and choosing the appropriate RTOS, the application can handle task control, communication and synchronization in such a way that the CPU overhead is minimized. This successfully reduces the active state.

The EFM32 MCU is based on the Cortex-M3 architecture and has been designed to help designers significantly reduce the active-mode power consumption. In benchmark tests, EFM32 at 32 MHz on a real-world 3V supply will run proper application code from flash memory at 180 µA/MHz.

With fast code execution and defined sleep modes, the low energy advantage is lost if the time it takes for the MCU to wake up from deep sleep and re-enter the active mode is too long. Energy is wasted. For MCU applications with real-time demands, the wake-up time must be kept to a minimum to enable the application to respond to an event within a set period of time. With latency demands shorter than the wake-up time of many existing MCUs, the device is often inhibited from going into deep sleep at all. In response, EFM32 uses a combination of design techniques to reduce the wake-up time from deep sleep to 2 µs, ensuring as little energy as possible is used before the CPU is enabled.

In the opposite situation, when the CPU is sleeping, the peripherals affect the energy efficiency. By applying the right low power design techniques, 32-bit processors can easily deliver sub-µA standby modes. At only 900 nA, the EFM32 microcontrollers incorporate real-time clock, power-on reset, brown-out detection and baseline functionality to make sleep modes useful.

If the microcontroller can leave peripherals functioning autonomously for extended periods, the CPU can either solve other high-level tasks or fall asleep, saving energy either way. The EFM32 peripherals are designed to look after themselves and communicate directly via the Peripheral Reflex System. The MCU incorporates several autonomous peripherals:

- an 8-channel, 12-bit ADC using 350 µA at full resolution and 1 Msamples/sec conversion rate;
- a 4x40 segment LCD controller using just 550 nA and providing boost, contrast, animation and blink functions; and
- a special low-energy UART, a full UART with 32 kHz clock, consuming only 150nA at a data transmission speed of 9600 baud.

Optimize Software Too

Having a super ultra low power microcontroller will not by itself guarantee us-
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 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 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.

**FIGURE 3**

Function contribution to energy consumption.

**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).
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.

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

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 µA levels.

If we change the Profiler to logarithmic scale, we see the current dropping to 1 µA in Deep Sleep Mode and 80 µA 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 Energy Micro, 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 saving for various Low-power modes of operation.

<table>
<thead>
<tr>
<th></th>
<th>EM0 Run Mode</th>
<th>EM1 Sleep Mode</th>
<th>EM2 Deep Sleep Mode</th>
<th>EM3 Stop Mode</th>
<th>EM4 Shutoff Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current consumption</td>
<td>180 µA/Mhz</td>
<td>45 µA/Mhz</td>
<td>0.9 µA</td>
<td>0.6 µA</td>
<td>20 nA</td>
</tr>
<tr>
<td>Wake-up time</td>
<td>0</td>
<td>0</td>
<td>2 µs</td>
<td>2µs</td>
<td>160 µs</td>
</tr>
<tr>
<td>Wake-up events</td>
<td>Any</td>
<td>Any</td>
<td>32 kHz peripherals</td>
<td>Asynic IRQ, I2C slave, Analog Comparators, Voltage Comparator</td>
<td>Reset</td>
</tr>
<tr>
<td>CPU</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
</tr>
<tr>
<td>High frequency peripherals</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
</tr>
<tr>
<td>Low frequency peripherals</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
</tr>
<tr>
<td>Full CPU and SRAM retention</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
</tr>
<tr>
<td>Power-on Reset/Brown-out Detector</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
</tr>
</tbody>
</table>

### TABLE 1

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).

### References

[www.expresslogic.com].

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