ELEKTRONIK PRAXIS

Interview

"Cycle resolution timing brings enormous benefits to embedded applications"

22.12.2021 | Von Sebastian Gerstl

Embedded software expert SEGGER wants to shake up the RTOS market with a revolutionary new approach. The new RTOS embOS-Ultra promises higher resolution and greater precision as well as energy savings right out of the box. In an exclusive interview with ELEKTRONIKPRAXIS, Lead Software Architect Martin Croell explains the technical details.

Martin Croell in his office in Monheim am Rhein. As Software Lead Architect, he was responsible for the development of the first RTOS with cycle resolution timing.
(Bild: Segger)

ELEKTRONIKPRAXIS: Mr. Croell, an RTOS (Real Time Operating System) is not a new invention, and hardly anyone thought that there was much potential for major improvement. How did you come up with the idea for embOS-Ultra in 2021, almost 40 years after the first RTOS?
Martin Croell: One of our corporate goals is to be and remain climate neutral [1]. This also means making our own products - such as our market-leading debug probes J-Link, J-Trace and our flashers, to name a few - even more energy efficient, although they already consume less energy in total than the fan alone in some competing products. Toward this goal, we also looked at the RTOS, among other things, and with embOS-Ultra we achieved the desired result of saving even more energy.

EP: You use an RTOS in your debuggers and flashers?
Croell: Correct, we have not only been selling our traditional RTOS embOS to customers for many years, but

we also use it in our J-Links and Flashers, just like other components of our all-in-one embedded OS emPower OS. Thus, we are the first beneficiary of any improvement in the emPower OS and provide our customers with products that have already been proven in practice by ourselves.

EP: Please explain to our readers first how embOS works and then how it is different from embOS-Ultra.
Croell: Gladly. embOS works - like all RTOSes since the 80s - with a system tick. This is the basic unit of time. Each time specification is given in multiples of ticks, and users can individually configure the distance between these ticks, typically to 1 millisecond, for example. This means that a hardware timer is programmed to generate an interrupt 1000 times per second, indicating the expiration of one millisecond at a time. Although 1 millisecond already sounds high-resolution and fast, we still saw the need for improvement given the requirements of modern embedded systems.

EP: What do you mean?

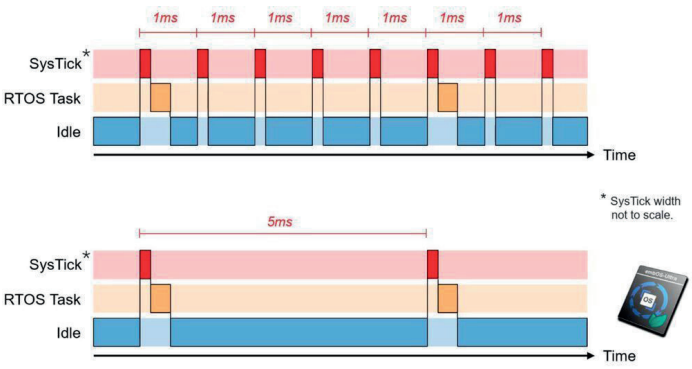

Figure 1: This graphic shows the difference between embOS-Ultra (bottom) and the conventional RTOS embOS (top): With embOS-Ultra, the CPU stays in energy-saving mode significantly longer and is woken up less often by interrupts (red). The result is more computing power for the application and less energy consumption.
(Bild: Segger)

Croell: Our new cycle-based scheduling changes the fundamental unit of time of the system, dramatically increasing the resolution of the scheduling. Instead of relying on the traditional system tick, embOS-Ultra uses clock cycles internally for all operations. Time-based operations such as for task delays or software timers, which previously could only be specified in multiples of ticks, can now be specified in clock cycles. In addition, embOS-Ultra users can now use system-independent, high-resolution units such as microseconds or even nanoseconds for time-based operations in one and the same application instead of just system ticks.

EP: In your opinion, does this option expand the market for RTOSes?
Croell: Absolutely. Finer temporal granularity means that the timing requirements of an application can be met with maximum accuracy.

As a result, RTOS scheduling can be used in many situations where it could not be used before because of the coarse time resolution. For example, there are often situations where delays of between 5 and 100 microseconds are required but cannot be provided by conventional tick-based systems, or only with significant disadvantages. In these cases, customers have often had to incur considerable additional development costs because an RTOS could not meet these requirements.

EP: What advantages does an RTOS offer in general, besides saving development effort?

Croell: (smiling) Well, especially now in the chip crisis, the importance of portability of applications from one hardware platform to another has become apparent. With in-house developments, you are locked to a specific controller, regardless of the higher development and maintenance costs. If that device becomes unavailable, you have a problem. Our embOS supports almost 1,000 different microcontrollers, so you can switch very easily without having to change your application.

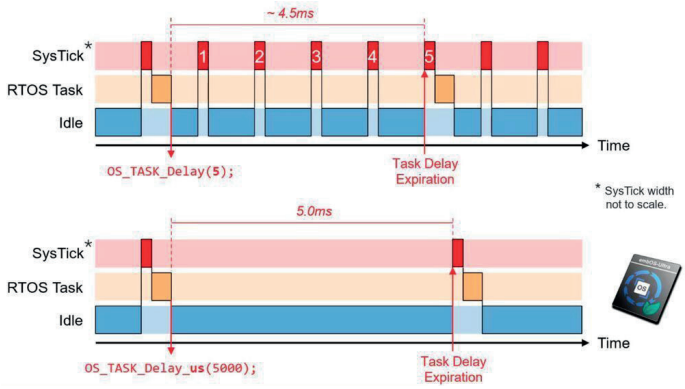

Figure 2: An example for the higher precision of embOS-Ultra. A delay of 5 ms can in reality only last 4.5 ms with a conventional RTOS, because a programmed delay cannot end between two system tick interrupts, but only with the next system tick interrupt, which then triggers
(Bild: Segger)

EP: But what does embOS-Ultra bring to applications that already use a traditional RTOS today?
Croell: Two things: Precision and energy consumption. First, temporal processes become more precise. In most RTOSes, for example, a Delay(10) will interrupt the operation of the corresponding thread for a time between 9 and 10 milliseconds - depending on how far away the next tick of the system is. Why is this? A programmed delay cannot end between two system tick interrupts, but only at the next system tick interrupt, which triggers the scheduler. Therefore, tasks that have to be suspended for a period shorter than a system tick can only do this by actively polling the hardware timer until the desired period has elapsed.

In embOS-Ultra with cycle resolution timing, on the other hand, a Delay_ms(10) results in exactly 10 ms of interruption time.

And then, of course, there is the power saving effect. Just think about the effort semiconductor manufacturers spend year after year to develop even more energy saving ULP microcontrollers. With embOS-Ultra you get energy savings quasi out-of-the-box.

EP: How exactly does this work?
Croell: Even if there is just one thread executing for several consecutive system ticks, the system tick interrupt occurs periodically and "wastes" computation time. In addition, the CPU state, i.e. register contents and flags, must be saved on the stack beforehand and then restored which also consumes computation time. To give another example: A CPU in power saving mode when no threads are running needs to be switched back to active mode in order to execute the interrupt service routine – also requiring computation time and thus energy that can be saved with embOS-Ultra.

EP: In what way does this affect cycle-based scheduling?
Croell (smiling): I was hoping you wouldn't ask. But seriously: As the system is able to handle different time

units (i.e. convert them into system cycles) instead of just computing in ticks, the code size is slightly increased by a few hundred bytes – which is insignificant for most systems. We, of course, calculate with 64-bit values, but 32-bit values are still sufficient for counting system ticks.

However, the resulting performance loss is minimal and insignificant on modern 32-bit CPUs in practice.

EP: Doesn't this still put you at risk of overflowing the 64-bit values with the number of cycles since system startup? Will the system then crash then?
Croell: Not at all. Let's do some math and assume a fast system with a clock frequency of 1 GHz. Here an overflow occurs after 2 ^ 64 cycles, which corresponds to about 585 years. This is unlikely to be a problem in 99.999% of cases.

But even if this were the case and we assume the system would have to run for 1,000 years, there are solutions, e.g. by using a slower clock. If you only assume a 100 MHz clock instead of 1 GHz, you already reach 5,850 years and still have a timing resolution of 10 nanoseconds.

EP: You have previously mentioned the counting of system ticks. Many developers focus on this value in their applications because they want to know how many interrupts have occurred since system startup. If embOS-Ultra doesn't have a "tick count", will those applications stop working?
Croell: True, the simple "tick count" was abolished in embOS-Ultra. Cycle-based scheduling still uses the timer interrupt, but it is no longer periodic. Instead, it is a single-shot hardware timer that is programmed to generate a timer interrupt exactly when it is needed.

But as you say: In some cases it proved useful to query the number of system ticks – whether to display them in a web interface or to use them in short loops with timeouts. If you need such a periodic interrupt every millisecond, using embOS-Ultra can replace the traditional tick count.

To replicate the tick count, you can query the cycle-based time and divide it by the cycle frequency. Taking a 400 MHz system, this means calculating OS_TIME_GetCycles() / 400000. Making things easier, there is even an API function for this that returns this value, conveniently named OS_TIME_GetTime_ms().

EP: Which timer concept does embOS-Ultra use? Everything you describe sounds relatively complex.
Croell: Most RTOSes simply use a hardware timer that gets configured to generate periodic interrupts, typically one per millisecond.

embOS-Ultra basically uses two hardware timers. One timer is used for long-term stability. This timer runs in continuous mode without generating interrupts. The other timer operates in single-shot mode, i.e. it counts down from a configured value to 0 or from 0 to a configured counter limit and then generates a single interrupt. If a timer is used that first counts from 0 to a configured limit, and then continues from there on to the next configurable limit, even one hardware timer is sufficient for using embOS-Ultra.

However, in most embedded systems there are more than enough hardware timers available, so that even the use of two timers typically is not a problem.

EP: The Arm architecture is the quasi-standard in the embedded world today and is supported as a matter of course. Can embOS-Ultra also be used for other architectures?
Croell: embOS-Ultra is available for many CPU and compiler combinations, including Arm Cortex-A/R/M or RISC-V. One embOS-Ultra port also includes a variety of board support packages for different devices and evaluation boards that allow users to directly put embOS-Ultra into operation without any additional effort.

EP: Assuming someone is already using a traditional RTOS and now wants to migrate to embOS-Ultra – how much effort is required?
Croell: In embOS-Ultra, the existing API remains unchanged compared to embOS. With the new cycle-based embOS-Ultra, existing functions therefore behave in the same way as with the traditional embOS. This means that API functions like OS_TASK_Delay() still result in the same millisecond-based timing, ensuring that the timing of an application migrated from embOS to embOS-Ultra does not change.

To take advantage of the new functionality, however, the API has been extended with functions such as OS_TASK_Delay_ms(), OS_TASK_Delay_us(), OS_TASK_Delay_Cycles() that provide much more precise timing. The same was done for the software timers provided by the RTOS.

The result is the best of both worlds: more accurate timing for modified and/or enhanced applications, while maintaining 100% compatibility for applications that are not to be modified.

Mr. Croell, thank you very much for the interview