New RTOS embOS-Ultra

Real-Time Operating System – Saving Energy Without a System Tick

28. März 2022, 6:00 Uhr | Frank Riemenschneider

(Bild: everythingpossible/stock.adobe.com)

Almost 40 years after the introduction of the first RTOS for embedded systems, Segger Microcontroller presents embOS-Ultra, a completely new approach in scheduling. The advantages compared to conventional RTOS include higher precision and more energy-efficient operation - effectively by default.

In the embedded world, where demand for »hard« real-time applications is the rule rather than the exception, the use of a real-time operating system (RTOS) has become more and more common. The reason for this is simple: Studies in software engineering in the 80s showed that the highest increase in productivity in software development is achieved through reuse.

Before discussing the specific innovations and advantages of embOS-Ultra compared to all conventional RTOSes known so far, the use of an RTOS in the embedded world shall be discussed in general for a better understanding.

Even in a comparatively simple application like the control of a washing machine, temperatures and pressures have to be measured and motors and pumps have to be controlled. One implementation without RTOS is the so-called super-loop architecture – also known as »bare-metal programming«. Here, there is no operating system, and the structure is quite simple: In the main() function, one sets up all variables, drivers, libraries, etc., and then executes one or more periodic tasks in a while loop (Figure 1).

External events must be programmed at the interrupt level of the processor, which communicates with functions from the main loop via global variables. There are always dependencies between data, time, functions and priorities. A large problem occurs, if program parts from the main loop are extended, then the time behavior of the whole application changes. It is also difficult to process a program in a priority-oriented way with this approach.

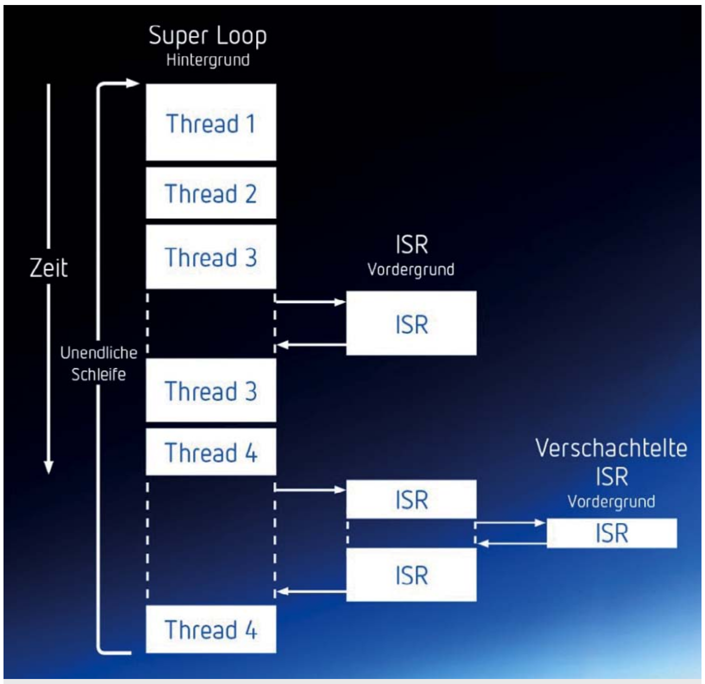

Figure 1. In the super-loop architecture, there is no operating system. In the main() function, one or more periodic tasks are executed in a while loop.
© Segger Microcontroller

Disturbances in one function means all others are also affected, making the system very error-prone. Further disadvantages are the loss of computing time by polling and missing multitasking, since an intervention into the program sequence is possible only by interrupts.

Of course, there are other challenges such as the manual management of resources, e.g. timers, interfaces, memory and computing time, which all have to be shared.

Advantages of Using an RTOS

The use of an RTOS eliminates the described disadvantages, since the operating system takes care of all these tasks, such as resource management. The application is divided up by the programmer into so-called threads, whereby a high degree of modularity is achieved. Tasks that are to run with different priorities are implemented as separate threads (Figure 2).

Each thread has a specific state at any point in time.
The three basic states are:

- Running: the thread is executing. On a single-core microcontroller, of course, only one thread can be in the Running state at any given time, since there is only one CPU available.
- Ready: The tasks in the Ready state are ready to run and are waiting for the CPU to allocate them.
- Waiting: The tasks are not ready to run at the moment. Only once certain events have occurred, e.g. data is available or a certain time has been reached, do the threads become Ready.

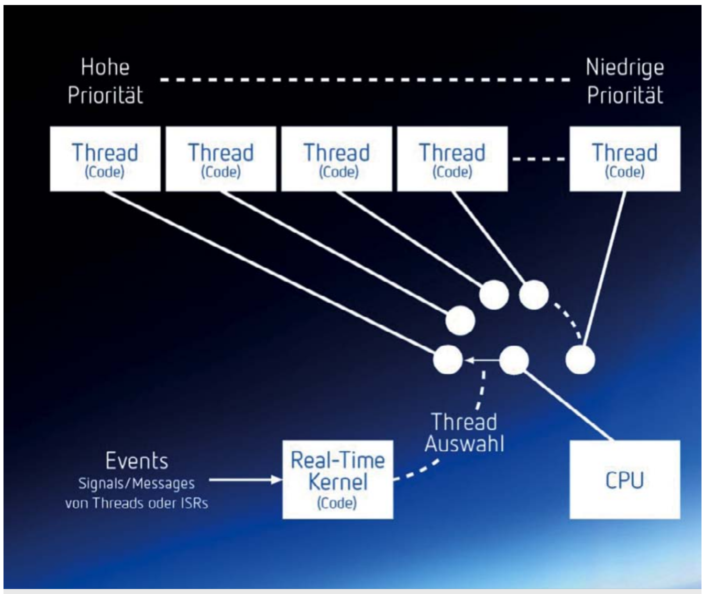

Figure 2. In an RTOS, the application is divided into threads. Tasks that are to run with different priority are implemented as separate threads.

The central switching point of the RTOS is the scheduler. It determines which thread gets the CPU allocated. All threads must be executed before their deadline to avoid problems. The deadline is the maximum permissible time, which may pass from the arrival of an event to the completion of the associated function, without causing issues.

Threads can be executed virtually concurrently, even if the embedded system on which they run has only a single CPU. This is done in one of two ways: time-slicing or priority-driven scheduling. Most RTOSes, such as Segger's embOS, allow both types of scheduling to be combined.

Time-Slice Driven and Priority Driven Scheduling

In a time-slice system, also known as a »round-robin« scheduling system, all threads that are ready to execute are executed for a specified time, the time slice. Then, the next thread is executed for a specified amount of time – usually the same amount. In this way, all threads share the CPU equally. A time slice is usually one ms or a multiple of one ms.

The disadvantage of the time-slice approach is that the execution of a given task can take several time slices, i.e., a few ms or even dozens of ms. This is not critical for things like the user interface (UI). The user will not notice a delay of, say, 10 ms in response.

However, for a network stack that can operate in a separate thread, this can be critical. If the network stack takes that much time to respond, it may not be able to complete its task.

The solution to this is priority-driven scheduling, where more time-critical tasks have a higher priority than less time-critical tasks. In this example, the UI task would have a lower priority than the network task. The network task can wait for an event in this way. The event can be triggered by an ethernet interrupt when a data packet is received. This allows the RTOS to immediately activate the network task, which can then immediately process and respond to the incoming packet.

Another advantage of using an RTOS that should not be underestimated is hardware abstraction: Especially now in the chip crisis, for example, the importance of portability of applications from one hardware platform to another has become apparent. Whereas with in-house developments you are locked into a specific controller regardless of the higher development and maintenance effort, Segger's embOS, for example, supports almost 1,000 different microcontrollers, making it very easy to switch without having to change the application. How this looks in practice has been described by a well-known industrial company in electronics [1].

The RTOS Revolution: Cycle-Resolution-Scheduling of embOS-Ultra

The scheduler of all RTOSes, from the 1980s to today, works with a system tick, which is the basic time unit. So every time specification is given in multiples of ticks, where users can configure the distance between these ticks individually, usually for example to 1 millisecond. This then means that a hardware timer is programmed to generate an interrupt 1000 times per second, indicating the expiration of one millisecond each time. Although 1 millisecond already sounds high-resolution and fast, there was still a need for improvement given the requirements of modern embedded systems.

embOS-Ultra's revolutionary clock-cycle-based scheduling changes the fundamental time unit of the system, dramatically increasing the resolution of the scheduling. Instead of relying on the traditional system tick, embOS-Ultra internally uses clock cycles 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 as a result. In addition, users of embOS-Ultra 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.

Advantages of embOS-Ultra in Practical Use

The cycle resolution scheduling of embOS-Ultra brings two decisive advantages to the developer: Higher precision and lower energy consumption. First, temporal sequences become more accurate.

In most RTOSes, for example, a Delay(5) will interrupt the operation of the corresponding thread for a time between 4 and 5 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 with the next system tick interrupt, which then triggers the scheduler (Figure 3, above).

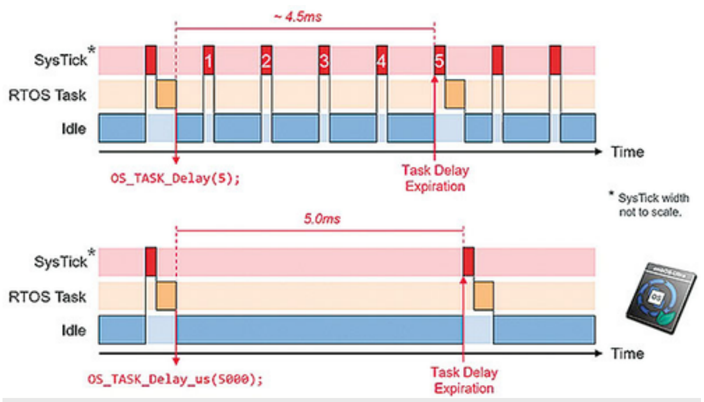

Figure 3. An example for the higher precision of embOS-Ultra. A delay of 5 ms can in reality only last 4.5 ms with a conventional RTOS, because a programmed delay cannot end between two system tick interrupts, but only with the next system tick interrupt, which then triggers the scheduler (top). With embOS-Ultra, on the other hand, the programmed delay is kept exactly (bottom).
© Segger Microcontroller

Therefore, tasks that are to be interrupted for a period shorter than a system tick can only achieve this by actively waiting via polling of the hardware timer until the desired period has elapsed. In embOS-Ultra with cycle-based resolution, however, a Delay_ms(5) results in exactly 5 ms interruption time (Figure 3, below).

Energy Saving by Default

And then there is the energy saving effect. Semiconductor manufacturers spend a lot of effort year after year on the development of even more energy saving microcontrollers. With embOS-Ultra you get energy savings quasi out-of-the-box.

Even if there is only one thread executing for several consecutive system ticks, the system tick interrupt will still occur periodically, thus »wasting« computing time (Figure 4, above). In addition, the status of the CPU, i.e., register contents and flags, must also be saved to the stack beforehand and then restored, which also consumes computing time.

In the same way, for example, a CPU that is in energy-saving mode when no threads are being executed must be switched back to active mode each time in order to execute the interrupt service routine. This also costs computing time and thus energy, which can be saved with embOS-Ultra. With embOS-Ultra, the CPU simply stays in energy-saving mode longer and is only woken up when there is something to do again (Figure 4, below).

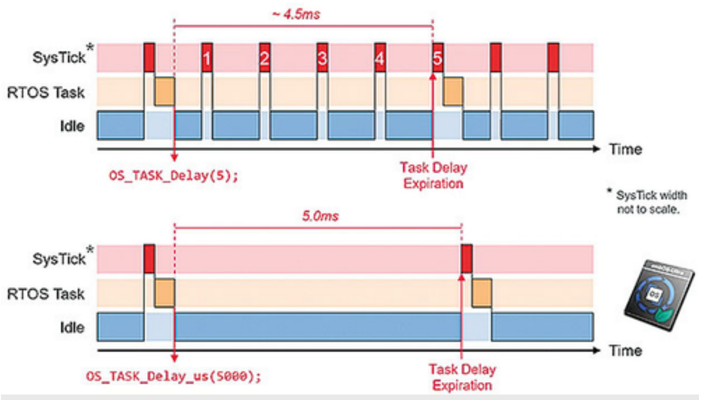

Figure 4. With embOS-Ultra (bottom), the CPU stays in energy-saving mode significantly longer and is woken up less often by interrupts (red). The result is more computing power for the application and less energy consumption.
© Segger Microcontroller

embOS-Ultra Internal

Many applications today use the system tick to count the number of interrupts since system startup, either to display them in a web interface or to use them in short loops with timeouts. The simple »Tick Count«, which gives the number of timer ticks since system startup, has disappeared in embOS-Ultra. In clock cycle-based scheduling, the timer interrupt is still used, but it is not periodic. Instead, it is a single-shot timer that is programmed to generate a timer interrupt exactly when it is needed.

However, in embOS-Ultra there is a way to replace the traditional Tick Count. Namely, to replicate the tick count, one can query the cycle-based time and divide it by the clock frequency, e.g., for a 400 MHz system, calculate OS_TIME_GetCycles() / 400000. To simplify things even more, there is even an API function for this that returns this value, conveniently named OS_TIME_GetTime_ms().

Most simple RTOSes simply use a hardware timer configured to generate periodic interrupts, typically just once per millisecond. embOS-Ultra basically uses two hardware timers. One timer is used for long-term stability. This timer runs in continuous mode and does not generate 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 to the next configurable limit, even one hardware timer is sufficient for embOS-Ultra operation. However, most embedded systems have more than enough hardware timers available, so even using two timers is usually not a problem.

Unlike conventional RTOSes, embOS-Ultra calculates the number of clock cycles with 64-bit values instead of 32-bit values, which are still sufficient for counting system ticks. The resulting performance loss is minimal and not significant on modern 32-bit CPUs in practice.

The fears of 64-bit values overflowing after a certain time are also unfounded: Even on an extremely fast CPU with 1 GHz, an overflow occurs after 264 clock cycles, which corresponds to about 585 years.

Easy Migration from a Conventional RTOS

In embOS-Ultra, the existing API has been left unchanged compared to embOS. Existing functions therefore behave the same in the new clock-cycle-based embOS-Ultra as they do in traditional embOS. This means that API functions such as OS_TASK_Delay() still result in the same millisecond-based timing, ensuring that the timing of an application migrating from embOS to embOS-Ultra does not change.

However, to take advantage of the new functionality, functions such as OS_TASK_Delay_ms(), OS_TASK_Delay_us(), OS_TASK_Delay_Cycles() have been added to the API to provide much more accurate 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.

Energy and Resource Efficiency Are the Drivers

One of Segger's corporate goals is to be and remain carbon neutral. This also means making its own products – the market-leading debug probes J-Link, J-Trace and Flasher are worth mentioning here – even more energy-efficient, although they already consume less energy overall than some fans alone in competitor products. To achieve this goal, among other things, the RTOS was also targeted and with embOS-Ultra the desired result of saving even more energy were achieved.

For many years Segger has not only sold embOS to customers, but also used it as well as other components of the all-in-one embedded OS emPower OS in the J-Links and Flashers. Thus Segger is the first beneficiary of any improvement in the emPower OS and provides its customers with products that have already been proven in practice at Segger itself.

embOS-Ultra is available for many CPU and compiler combinations, including of course ARM Cortex-A/R/M and RISC-V.

An embOS-Ultra port also includes a variety of board support packages for different devices and evaluation boards. This allows users to put embOS-Ultra directly into operation without any additional effort.

Last but not least, embOS-Ultra has been optimized for minimal memory consumption in RAM and ROM as well as for high speed and versatility. Throughout the development process of embOS-Ultra the limited resources of microcontrollers were always kept in mind. The internal structure has been optimized in a variety of applications with different customers to meet industry requirements. Due to the high level of modularity, only the functions that are needed are included in an application, keeping the ROM size very small. A few files are included as source code to ensure that customers do not lose flexibility and can fully customize the system to their needs.