Elektronik

Neues Echtzeitbetriebssystem embOS-Ultra

RTOS – Energiesparen ohne System-Tick

(Bild: everythingpossible/stock.adobe.com)

Fast 40 Jahre hat es nach der Vorstellung des ersten Real Time Operating Systems für Embedded-Systeme gedauert, bis Segger Microcontroller mit embOS-Ultra einen völlig neuen Ansatz für das Scheduling vorgestellt hat. Zu den Vorteilen gegenüber herkömmlichen RTOS zählt neben höherer Präzision der energieeffizientere Betrieb – quasi per Default.

Von Frank Riemenschneider

In der Embedded-Welt, in der die Forderung nach »harter« Echtzeit eher die Regel als die Ausnahme ist, hat sich der Einsatz eines Echtzeitbetriebssystems (RTOS) mehr und mehr durchgesetzt. Der Grund hierfür ist einfach: Untersuchungen im Software Engineering haben bereits Ende der 80er-Jahre ergeben, dass die höchste Produktivitätssteigerung in der Softwareentwicklung durch Wiederverwendung erreicht werden kann.

Schon bei einer vergleichsweise einfachen Anwendung wie der Steuerung einer Waschmaschine müssen Temperaturen und Drücke gemessen und Motoren und Pumpen angesteuert werden. Eine Umsetzungsmöglichkeit ohne RTOS ist die sogenannte Super-Loop-Programmierung – auch bekannt als »Bare-Metal-Programmierung«. Hierbei wird kein Betriebssystem eingesetzt und die Programmstruktur ist recht einfach: In der main()-Funktion

werden alle Variablen, Treiber, Bibliotheken usw. eingerichtet und anschließend eine oder mehrere periodische Aufgaben in einer while-Schleife ausgeführt (**Bild 1**).

Externe Ereignisse müssen auf der Interrupt-Ebene des Prozessors programmiert werden, die mit Funktionen aus der Hauptschleife über globale Variablen kommuniziert. Es gibt immer Abhängigkeiten zwischen Daten, Zeit, Funktionen und Prioritäten. Ein großes Problem tritt auf, wenn Programmteile aus der Hauptschleife erweitert werden, dann ändert sich nämlich das Zeitverhalten der ganzen Applikation. Auch ein Programm prioritätsorientiert abzuarbeiten, ist mit diesem Ansatz schwer.

Kommt es zu Störungen in einer Funktion sind alle anderen Funktionen mitbetroffen, das System ist daher sehr fehleranfällig. Weitere Nachteile sind Rechenzeitverlust durch Polling und fehlendes Multitasking, da ein Eingriff in den Programmablauf nur durch Interrupts möglich ist.

Natürlich gibt es noch weitere Herausforderungen wie die händische Verwaltung von Ressourcen z. B. von Timern, Schnittstellen, Speicher und Rechenzeit, die ja allesamt geteilt werden müssen.

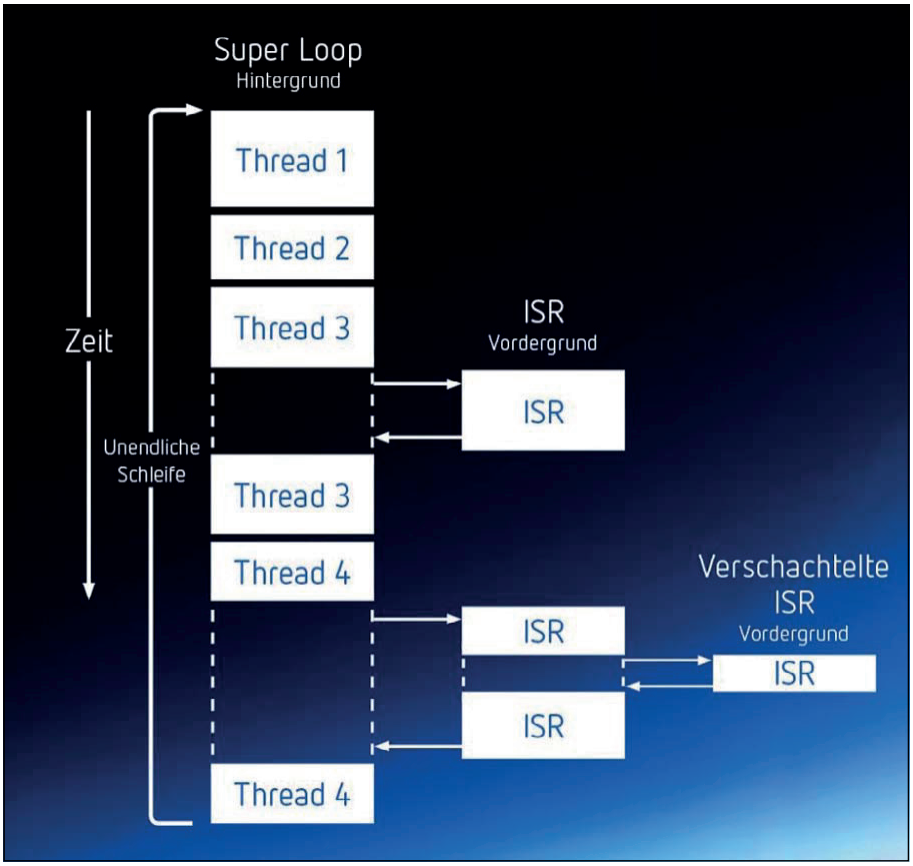

Bild 1. Bei der Super-Loop-Programmierung gibt es kein Betriebssystem. In der main()-Funktion werden eine oder mehrere periodische Aufgaben in einer while-Schleife ausgeführt. (Bild: Segger Microcontroller)

Vorteile der Nutzung eines RTOS

Durch den Einsatz eines RTOS werden die beschriebenen Nachteile beseitigt, da sich das Betriebssystem um all diese Aufgaben wie z. B. das Ressourcenmanagement kümmert. Die Anwendung wird vom Programmierer in sogenannte Threads aufgeteilt, wodurch ein hohes Maß an Modularität erreicht wird. Aufgaben, die mit unterschiedlicher Priorität ablaufen sollen, werden als getrennte Threads realisiert (**Bild 2**). Jeder Thread hat zu jedem Zeitpunkt einen bestimmten Zustand. Die drei grundlegenden Zustände sind:

- ➔ Running: Der Thread wird ausgeführt. Auf einem Single-Core-Mikrocontroller kann zu einem bestimmten Zeitpunkt natürlich nur ein Thread im Running-Zustand sein, da es ja nur eine CPU gibt.
- ➔ Ready: Die Tasks im Ready-Zustand sind laufbereit und warten auf die Zuteilung der CPU.
- ➔ Waiting: Die Tasks sind momentan nicht laufbereit. Erst wenn bestimmte Ereignisse eingetreten sind, wie z. B. Daten verfügbar sind oder ein bestimmter Zeitpunkt erreicht wurde, werden die Threads Ready.

Die zentrale Schaltstelle des RTOS ist der Scheduler. Er bestimmt, welcher Thread wann die CPU zugeteilt bekommt. Alle Threads sollen oder müssen vor Ablauf ihrer Frist ausgeführt werden. Die Frist ist dabei die maximal zulässige Zeit, die vom Eintreffen eines Ereignisses bis zur Beendigung der dazugehörenden Funktion vergehen darf, ohne dass es zu Problemen – welcher Art auch immer – kommt.

Threads können quasi gleichzeitig ausgeführt werden, auch wenn das Embedded-System, auf dem sie laufen, nur eine einzige CPU hat. Dies geschieht auf eine von zwei Arten: Zeitscheibenverfahren oder prioritätsgesteuertes Scheduling. Die meisten RTOS, wie beispielsweise Seggers embOS, erlauben es, beide Arten des Schedulings zu kombinieren.

Zeitscheiben- und prioritätsgesteuertes Scheduling

In einem Zeitscheibensystem, auch bekannt als »Round-Robin«-Scheduling-System, werden alle Threads, die zur Ausführung bereit sind, für eine bestimmte Zeit (die Zeitscheibe) ausgeführt. Danach wird der nächste Thread für eine bestimmte Zeitspanne – in der Regel die gleiche Zeitspanne – ausgeführt. Auf diese Weise teilen sich alle Threads die CPU gleichmäßig. Eine Zeitscheibe dauert in der Regel 1 ms oder ein Vielfaches davon.

Der Nachteil des Zeitscheibenverfahrens besteht darin, dass die Ausführung einer bestimmten Aufgabe mehrere Zeitscheiben, d. h. einige Millisekunden oder sogar Dutzende von Millisekunden, in Anspruch nehmen kann. Dies mag zwar für einige Dinge wie

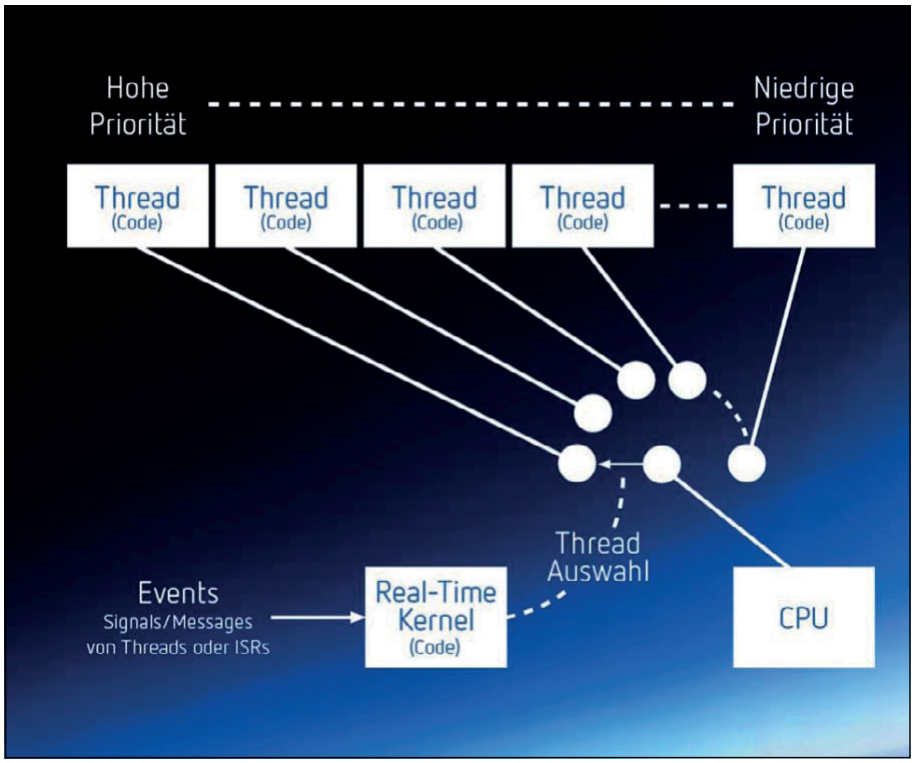

Bild 2. Bei einem RTOS wird das Programm in Threads aufgeteilt. Aufgaben, die mit unterschiedlicher Priorität ablaufen sollen, werden als getrennte Threads realisiert. (Bild: Segger Microcontroller)

die Benutzeroberfläche (UI) unkritisch sein, der Nutzer wird eine Verzögerung von z. B. 10 ms bei der Antwort nicht bemerken. Für einen Netzwerk-Stack, der in einem separaten Thread arbeiten kann, kann sich eine solche Verzögerung jedoch kritisch auswirken. Wenn der Netzwerkstapel so viel Zeit für die Reaktion benötigt, kann er seine Aufgabe möglicherweise nicht erfüllen.

Die Lösung hierfür ist das prioritätsgesteuerte Scheduling, bei dem zeitkritischere Tasks in der Regel eine höhere Priorität haben als weniger zeitkritische Tasks. In diesem Beispiel hätte der UI-Task eine niedrigere Priorität als der Netzwerk-Task. Der Netzwerk-Task kann auf diese Weise auf ein Ereignis warten. Das Ereignis kann durch einen Interrupt in der Interrupt Service Routine (ISR) ausgelöst werden, wenn ein Datenpaket empfangen wurde. Dies ermöglicht es dem RTOS, den Netzwerk-Task sofort zu aktivieren, der dann das eingehende Paket sofort verarbeiten und darauf reagieren kann.

Ein weiterer nicht zu unterschätzender Vorteil beim Einsatz eines RTOS ist die Hardwareabstraktion: Gerade jetzt in der Chipkrise ist die Bedeutung der Portierbarkeit von Programmcode von einer Hardwareplattform auf eine andere deutlich sichtbar geworden. Bei Eigenentwicklungen ist man – unabhängig vom höheren Entwicklungs- und Wartungsaufwand – auf einen bestimmten Mikrocontroller festgelegt, Betriebssysteme wie z. B. Seggers embOS unterstützten dagegen fast 1.000 unterschiedliche Mikrocontroller, sodass man sehr einfach den Mikrocontroller wechseln kann, ohne den Programm-Code verändern zu müssen [1].

RTOS-Revolution Cycle Resolution Scheduling

Der Scheduler aller bisher bekannten RTOS seit den 80er-Jahren arbeitet mit einem System-Tick, der die grundlegende Zeiteinheit darstellt. Jede Zeitangabe wird also in Vielfachen von Ticks angegeben, wobei Benutzer den Abstand zwischen diesen Ticks individuell konfigurieren können, üblicherweise zum Beispiel auf 1 ms. Das bedeutet dann, dass eine Timer-Schaltung so programmiert wird, dass sie 1000-mal pro Sekunde einen Interrupt auslöst und damit jeweils das Ablaufen einer Millisekunde anzeigt. Obwohl 1 ms sehr kurz zu sein scheint und eine hohe zeitliche Auflösung verspricht, stellen moderne Embedded-Systeme darüber hinausgehende Anforderungen. Es besteht also die Notwendigkeit für Verbesserungen.

Das neue, revolutionäre Taktzyklusbasierte Scheduling von embOS-Ultra (Cycle Resolution Scheduling) verändert die grundlegende Zeiteinheit des Embedded-Systems, wodurch die Auflösung des Schedulings drastisch erhöht wird. Anstatt sich auf den traditionellen System-Tick zu verlassen, verwendet embOS-Ultra intern für alle Operationen Taktzyklen. Zeitbasierte Operationen etwa für Task-Verzögerungen oder Software-Timer, die bisher nur in Vielfachen von Ticks definiert werden konnten, können dadurch nun in Taktzyklen angegeben werden. Zusätzlich können Entwickler mit embOS-Ultra für zeitbasierte Operationen statt lediglich System-Ticks nun auch systemunabhängige Zeiteinheiten, wie Mikrosekunden oder sogar Nanosekunden, in ein und derselben Applikation verwenden.

Vorteile von embOS-Ultra im praktischen Einsatz

Das Cycle Resolution Scheduling von embOS-Ultra bringt dem Entwickler zwei entscheidende Vorteile: Höhere Präzision und geringere Energieaufnahme. Zeitliche Abläufe lassen sich nun genauer definieren. In den meisten RTOS wird z. B. ein Delay(5) den Betrieb des entsprechenden Threads für eine Zeit zwischen 4 ms und 5 ms unterbrechen – je nachdem, wie weit der nächste Tick des Systems zeitlich entfernt ist. Warum ist das so? Eine programmierte Verzögerung kann nicht zwischen zwei System-Tick-Interrupts enden, sondern erst mit dem nächsten System-Tick-Interrupt, der dann den Scheduler auslöst (**Bild 3**, oben). Daher können Aufgaben, die für einen Zeitraum unterbrochen werden sollen, der kürzer als ein System Tick ist, dies nur erreichen, indem sie aktiv mittels Polling des Hardware Timers warten, bis der gewünschte Zeitraum verstrichen ist. In embOS-Ultra mit zyklusbasierter Auflösung führt ein Delay_ms(5)

hingegen zu einer Unterbrechungszeit von genau 5 ms (Bild 3, unten).

Energiesparen per Default

Halbleiterhersteller betreiben Jahr für Jahr hohen Aufwand, um noch energiesparendere Mikrocontroller zu entwickeln. Mit embOS-Ultra bekommen Entwickler von Embedded-Systemen Energieeinsparungen quasi »out of the box«.
Selbst wenn es nur einen Thread gibt, der für mehrere aufeinander folgende System Ticks ausgeführt wird, wird die System-Tick-Unterbrechung immer noch periodisch auftreten und dadurch Rechenzeit »verschwendet« (Bild 4, oben). Dazu müssen zuvor auch noch der Status der CPU, also Registerinhalte und Flags, auf dem Stack gerettet und anschließend wiederhergestellt werden, was ebenfalls Rechenzeit beansprucht. Genauso muss zum Beispiel eine CPU, die sich im Energiesparmodus befindet, wenn keine Threads ausgeführt werden, jedes Mal wieder in den aktiven Modus überführt werden, um die Interrupt-Service-Routine auszuführen. Auch das kostet Rechenzeit und somit Energie, die mit embOS-Ultra eingespart werden kann. Mit embOSUltra bleibt die CPU einfach länger im Energiesparmodus und wird erst dann aufgeweckt, wenn es tatsächlich wieder etwas zu tun gibt (Bild 4, unten).

embOS-Ultra intern

Viele Anwendungen verwenden heute den System Tick, um die Anzahl der Interrupts seit dem Systemstart zu zählen, sei es, um sie in einer Webschnittstelle anzuzeigen oder in kurzen Schleifen mit Timeouts zu verwenden. Der einfache »Tick Count«, der die Anzahl der Timer-Ticks seit dem Start des Systems angibt, ist in embOS-Ultra verschwunden. Beim taktzyklusbasierten Scheduling wird der Timer-Interrupt zwar immer noch verwendet, aber er ist nicht periodisch. Stattdessen handelt es sich um einen Single-Shot-Timer, der so programmiert wird, dass er genau dann einen Timer-Interrupt erzeugt, wenn er benötigt wird.

In embOS-Ultra gibt es jedoch eine Möglichkeit, den traditionellen Tick Count zu ersetzen. Um den Tick Count zu replizieren, kann nämlich die zyklusbasierte Zeit abgefragt und durch die Taktfrequenz geteilt werden – z.B. bei inem 400-MHz-System: OS_TIME_ GetCycles() / 400.000. Um das Ganze noch zu vereinfachen, gibt es dafür sogar eine API-Funktion, die diesen Wert liefert und die praktischerweise OS_TIME_GetTime_ms() heißt.
Die meisten einfachen RTOS verwenden eine Timer-Schaltung, die so konfiguriert ist, dass sie periodische Interrupts erzeugt, typischerweise eben einmal pro Millisekunde. Anders embOS-Ultra, es verwendet im Grunde wei Timer-Schaltungen. Ein Timer wird für die Langzeitstabilität verwendet. Dieser Timer läuft im kontinuierlichen Modus und erzeugt keine Interrupts. Der andere Timer arbeitet im Single-Shot-Modus, d. h. er zählt von einem konfigurierten Wert auf 0 herunter oder von 0 bis zu einer konfigurierten Zählergrenze und erzeugt dann einen einzelnen Interrupt. Wird ein Timer verwendet, der zunächst von 0 bis zu einer konfigurierten Grenze zählt, und dann von dort aus weiter bis zum nächsten konfigurierbaren Grenzwert, so genügt sogar ein Hardware-Timer für den Betrieb von embOS-Ultra. Die meisten Embedded-Systeme haben jedoch mehr als genug Timer zur Verfügung, sodass auch die Verwendung zweier Timer üblicherweise kein Problem darstellt. Anders als herkömmliche RTOS rechnet embOS-Ultra bei der Anzahl von Taktzyklen mit 64-bit-Werten statt mit 32-bit-Werten, die für das Mitzählen von System Ticks noch ausreichen. Der daraus resultierende Leistungsverlust ist minimal und auf modernen 32-bit-CPUs in der Praxis nicht signifikant. Auch die Befürchtung von Überläufen der 64-bit-Werte nach einer bestimmten Zeit sind unbegründet: Selbst bei einer schnellen CPU mit 1-GHz-Takt tritt ein Überlauf erst nach 264 Taktzyklen auf, was ungefähr 585 Jahren entspricht. Aber auch für den Fall, dass ein Embedded-System 1.000 Jahre

lang laufen müsste, gibt es Lösungen, z. B. die Verwendung einer niedrigeren Taktfrequenz. Wenn die Taktfrequenz statt 1 GHz nur 100 MHz betragen würde, könnten bereits 5.850 Jahre erreicht werden – und der Entwickler hätte immer noch eine hohe zeitliche Auflösung von 10 ns.

Migration von einem herkömmlichen RTOS

In embOS-Ultra wurde das bestehende API im Vergleich zu embOS unverändert gelassen. Bestehende Funktionen verhalten sich im neuen taktzyklus-basierten embOS-Ultra daher genauso wie im traditionellen embOS. Das bedeutet, dass API-Funktionen wie OS_TASK_Delay() immer noch das gleiche, auf Millisekunden ausgerichtete Timing ergeben. Dadurch wird sichergestellt, dass sich das Timing einer Anwendung, die von embOS auf embOS-Ultra migriert wird, nicht ändert.

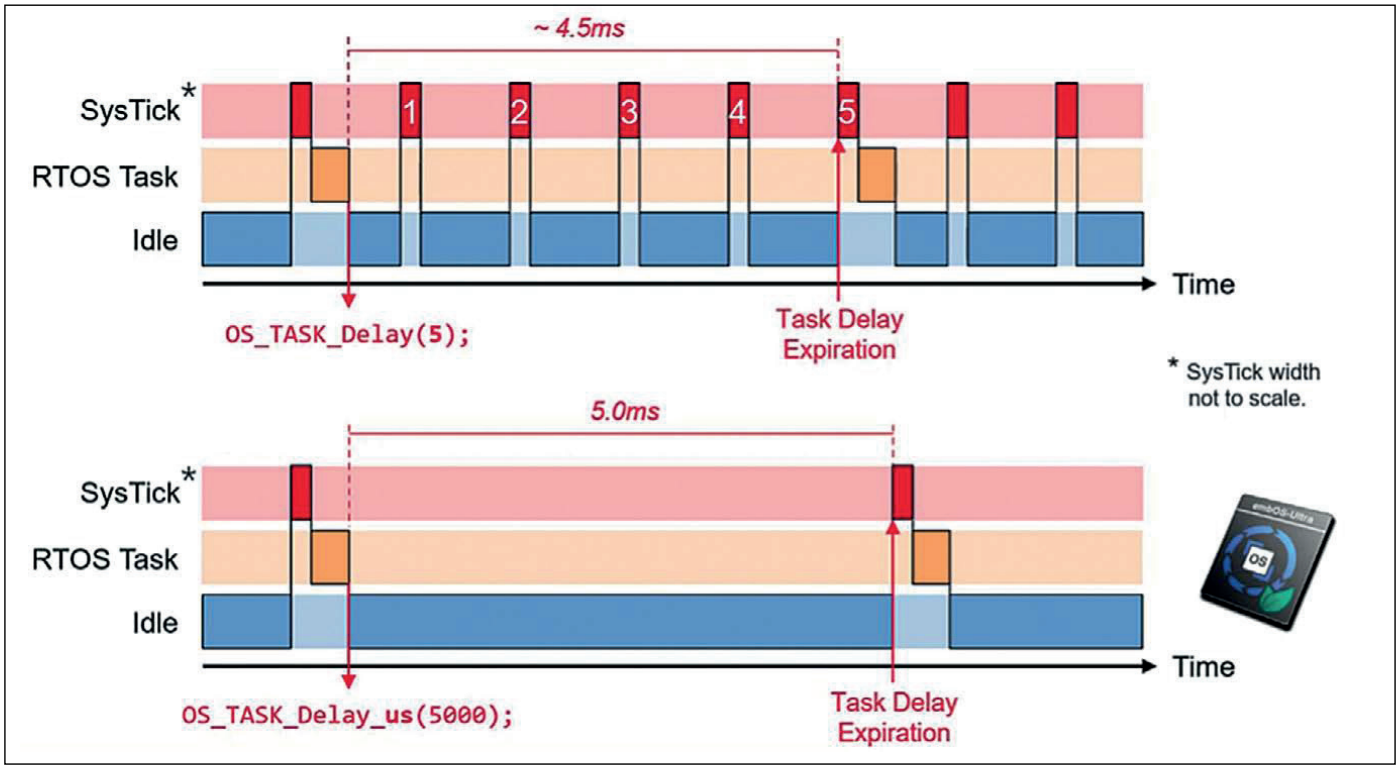

Bild 3. Ein Beispiel für die höhere Präzision von embOS-Ultra. Ein Delay von 5 ms kann bei einem herkömmlichen RTOS in Wirklichkeit nur 4,5 ms dauern, da eine programmierte Verzögerung nicht zwischen zwei System-Tick-Interrupts enden kann, sondern erst mit dem nächsten System-Tick-Interrupt, der dann den Scheduler auslöst (oben). Bei embOS-Ultra wird die programmierte Verzögerung hingegen exakt eingehalten (unten). (Bild: Segger Microcontroller)

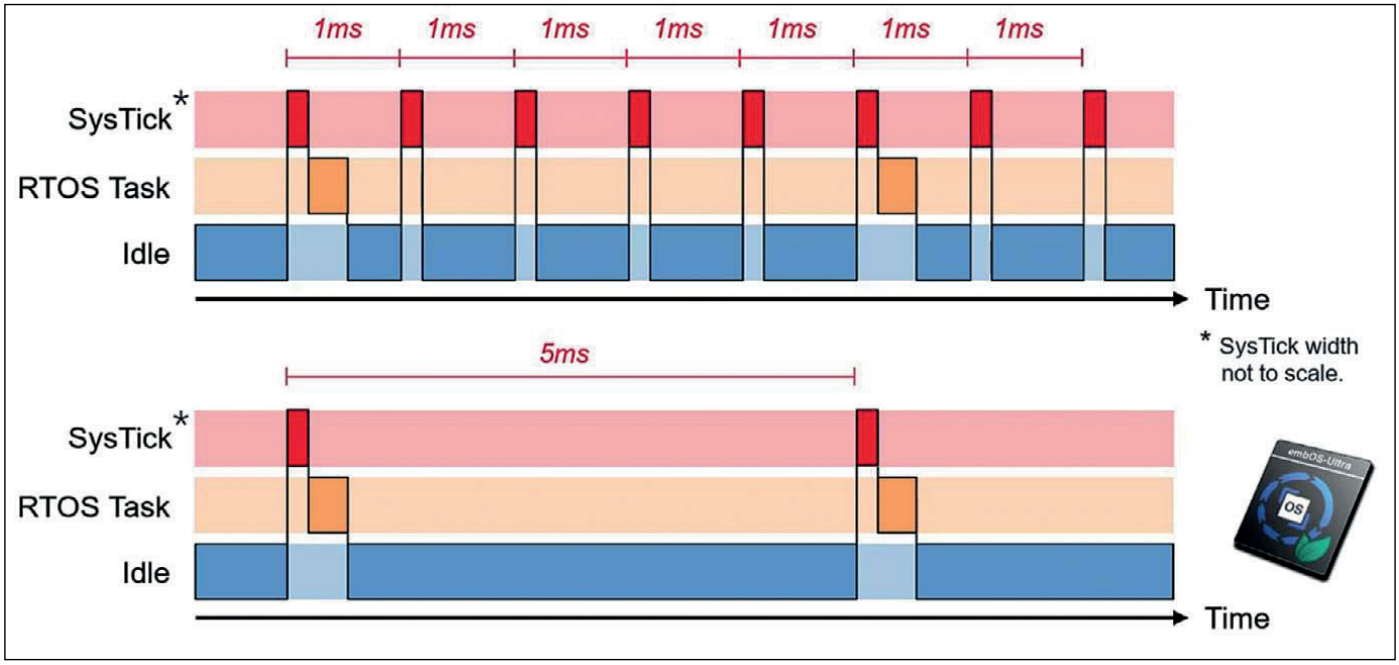

Bild 4: Mit embOS-Ultra (unten) bleibt die CPU deutlich länger im Energiesparmodus und wird weniger oft durch Interrupts (rot) aufgeweckt. Die Folgen sind mehr Rechenleistung für die Anwendung und weniger Energieaufnahme. (Bild: Segger Microcontroller)

Um jedoch die Vorteile des neuen Cycle-Resolution-Scheduling zu nutzen, wurde das API um Funktionen wie OS_TASK_Delay_ms(), OS_TASK_Delay_us(), OS_TASK_Delay_Cycles() erweitert, die eine viel genauere Zeitmessung ermöglichen. Das Gleiche wurde auch für die vom RTOS bereitgestellten Software-Timer getan. Das Ergebnis ist das Beste aus beiden Welten: ein genaueres Timing für geänderte und/oder erweiterte Anwendungen, bei gleichzeitig 100-prozentiger Kompatiblität für Anwendungen, die nicht modifiziert werden sollen.

Energie- und Ressourceneffizienz sind die Treiber

Eines von Seggers Unternehmenszielen ist es, klimaneutral zu sein und zu bleiben. Dies bedeutet auch, die eigenen Produkte – beispielsweise die Debug-Probes J-Link, J-Trace und die Flash-Programmiergeräte – noch energieeffizienter zu machen, obwohl sie schon heute insgesamt weniger Energie brauchen, als mancher Lüfter in Konkurrenzprodukten alleine. Um dieses Ziel zu erreichen, wurde u. a. auch beim RTOS angesetzt und mit embOS-Ultra das gewünschte Ergebnis erreicht, noch mehr Energie einzusparen.

Seit vielen Jahren verkauft Segger das RTOS embOS nicht nur an Kunden, sondern setzt es wie auch andere Komponenten des All-in-one Embedded-OS emPower OS in den eigenen Produkten J-Link und Flasher ein. Damit ist Segger erster Nutznießer jeder Verbesserung im emPower OS und liefert seinen Kunden ein RTOS, das sich bereits in der Praxis bei den eigenen Produkten bewährt hat. Das neue RTOS embOS-Ultra ist für viele CPU- und Compiler-Kombinationen, darunter ARM-Cortex-A/R/M oder auch RISC-V, verfügbar. Eine embOSUltra- Portierung enthält zudem eine Vielzahl Board-Support-Packages für verschiedene Prozessor- und Evaluierungsmodule. So können Nutzer embOS-Ultra ohne zusätzlichen Aufwand direkt in Betrieb nehmen. Last but not least wurde embOS-Ultra auf minimalen Speicherbedarf in RAM und ROM sowie auf hohe Geschwindigkeit und Vielseitigkeit optimiert. Während des gesamten Entwicklungsprozesses von embOS-Ultra wurden die begrenzten Ressourcen von Mikrocontrollern stets im Auge behalten. Die interne Struktur wurde in einer Vielzahl von Anwendungen mit unterschiedlichen Kunden optimiert, um den Anforderungen der Industrie gerecht zu werden. Durch die hochgradige Modularität werden nur die Funktionen, die benötigt werden, in eine Anwendung eingebunden, wodurch die ROM-Größe sehr klein gehalten wird. Ein paar Dateien werden zudem im Quellcode mitgeliefert, um sicher-zustellen, dass Kunden durch die Verwendung von embOS-Bibliotheken keine Flexibilität einbüßen und das System vollständig an ihre Bedürfnisse anpassen können.

Literatur

[1] *Schubert, H.:* **Konzepte gegen Liefer-engpässe – Modularität ist Trumpf.** *elektronik.de*, **4. Oktober 2021, www.elektroniknet.de/embedded/software/modularitaet-ist-trumpf.190206.html**

Frank Riemenschneider

studierte Elektrotechnik mit dem Schwerpunkt Prozessorarchitekturen an der Leibniz-Universität in Hannover. Seit März 2021 ist er bei Segger Microcontroller als Marketing- und PR-Manager beschäftigt.
frank.riemenschneider@segger.com