

# RTOS - Aktion



Eine Gemeinschaftsaktion der Firmen:

Ein Werkzeug ist nur die eine Seite der Medaille. Die Methodik der Anwendung die andere.

## Tools + Schulung = Erfolg

Um Ihnen die Entscheidung, in Ihr Software-Architektur-Design zu investieren, leichter zu machen und einen erfolgreichen Einstieg zu ermöglichen, gibt es bei Willert bis zum Ende des Jahres 2009 zu den Software-Architektur-Produkten und Schulungen besonders günstige Komplettpakete:

- **Schulung**  
(RTOS und Software-Architektur)  
2 oder 3 Tage  
(Inhalte siehe graues Feld -->)  
  
ab 499,- €
- **RTOS + Schulung**  
Keil RL - ARM (RTOS)  
2 Tage SW-Arch.- u. RTOS - Schulung  
  
3.400,- €
- **Compiler + RTOS + Schulung**  
Keil MDK-ARM (Compiler)  
Keil RL-ARM (RTOS)  
Keil Ulink2 + Eva-Board  
3 Tage SW-Arch.- und RTOS-  
Schulung incl. Cortex-Architektur  
  
6.750,- €

Bestellung zu o.g. Preisen nur bei :

**Willert Software Tools GmbH**

rschaak@willert.de

Tel.: 05722 - 9678 60

## Embedded Software-Architektur mit dem Keil RTX Real-Time Kernel

Schulung am 02.- 04.11.2009 in Leipzig

### Einführung in Keil µVision (KEIL RealView MDK) und dem ARM Cortex-M3

1. Tag:

- ARM Cortex-M3 Core (Architektur, Befehlssatz, Interrupts)
- Unterschiede ARM7, ARM9 und Cortex-M3
- Keil RealView MDK Tool-Chain
- Keil µVision Simulation und Target-Debugging
- Praktische Beispiele mit der Familie STM32 (GPIO, ADC, CAN, UART, Interrupt-Handling)

### RTOS KEIL RTX Real-Time Kernel

2. und 3. (halber) Tag:

- Latent „gewachsene“ Software - Grenzen und Probleme
- Embedded-Software-Architekturdesign
- Software - Schichten und Möglichkeiten der Entkopplung
- RTOS Grundlagen
- Scheduling und Kommunikation
- Keil RTX Real Time Kernel
  - Grundlagen
  - Konfiguration und Ressourcen
  - Tasks, Messages und Events
  - Echtzeitanforderungen
- Praktische Beispiele mit dem RTX Real-Time-Kernel

### RL- ARM RealView® Real-Time Library

3. (halber) Tag:

- Einführung und Überblick der RL-ARM RTL (TcpNet, FlashFileSystem, CAN, USB )
- Praktisches Beispiel mit der RealView® Real-Time Library (Vorführung)

**2.+ 3. Tag 499,-€ / 3 Tage 599,- €**

Anmelden unter:

**[www.willert.de/anmeldung](http://www.willert.de/anmeldung)**

Alle Preise sind Netto-Preise zzgl. der gesetzl. Mwst

# Kriterien zur Auswahl eines RTOS

---

Am Anfang einer jeden Design-Entscheidung steht in der Regel eine Systemanalyse. Bei der Fragestellung, ob ein RTOS, und wenn ja welches, eingesetzt werden soll, verhält es sich nicht anders. Die Grundlage für die Auswahl eines Betriebssystems basiert auf der Wahl des Architekturdesigns und den Anforderungen der Applikation in dieser Hinsicht.

Wie bereits im Leitartikel dieser Newsletter geschrieben ist das Laufzeitarchitekturdesign eine effiziente Möglichkeit der latent steigenden Komplexität zu begegnen. Um sich für ein geeignetes Architektur-Design zu entscheiden sind folgende Punkte zu beachten:

1. Welche Nebenläufigkeiten müssen in dem System realisiert werden
2. Wie sind die Reaktionszeit bzw. Periodizität jeder einzelnen Nebenläufigkeit
3. Was sind die maximalen Laufzeiten der einzelnen Nebenläufigkeiten
4. Welcher Art ist der überwiegende Charakter der Sensorik und Aktorik

Punkt 1 und 2 sind in der Regel auch schon bei Projektbeginn hinreichend genau analysierbar. Die Laufzeit einzelner Nebenläufigkeiten hingegen lässt sich oft erst zum Zeitpunkt der Implementation genau angeben. Aber das, ist wie wir später sehen werden, nicht ganz so wichtig. Vorerst reichen grobe Schätzungen.

Der Charakter der Aktorik und Sensorik wird zeitkontinuierlich oder ereignisgetrieben unterschieden.

## Systemanalyse

### 1. Schritt: Vorentscheidung zeit- oder ereignisgetriebenes Design

An erster Stelle gilt es herauszufinden, ob der überwiegende Teil der Sensorik in seiner Natur tendenziell zeitkontinuierlichen oder ereignisorientierten Charakter besitzt. Das gibt uns den ersten Hinweis auf unsere innere Laufzeitarchitektur, denn sie sollte grundsätzlich analog zur Art der Anbindung der Peripherie sein. Hierzu eine wichtige Anmerkung: Seit Jahren gibt es einen ständigen Trend hin zu

ereignisorientierter Anbindung der Außenwelt. Im Zweifel würde ich mich also immer für eine ereignisgetriebene Architektur entscheiden. Folgend ein paar grobe Richtlinien.

- Passive Sensoren sind vom Charakter eher zeitkontinuierlich, aktive Sensoren eher ereignisgetrieben
- Über Bussysteme angebundene Sensorik ist vom Charakter her, von wenigen Ausnahmen abgesehen, ereignisgetrieben
- Schalter an einem digitalen I/O Port sind zeitkontinuierlich, Taster an einem Interrupt-Eingang ereignisgetrieben

Von der äußeren Anbindung der Peripherie sollte auch die innere Architektur abgeleitet werden, da es zu einem grundsätzlich harmonischeren Architekturdesign führt. Aber Achtung: Historisch gesehen wurden die meisten Designs zeitlich kontinuierlich realisiert. Aus diesem Grund ist oft auch die Peripherie zeitlich kontinuierlich angebunden, obwohl der inhärente Charakter ereignisgetrieben ist. Noch heute ist es häufig so, dass Sensoren aus historischen Gründen über einen Bus pollend abgefragt werden, obwohl der Sensor ein Signal schicken könnte. Hier gilt es die Systeme von Altlasten zu befreien.

### 2. Schritt: Analyse der Nebenläufigkeiten

Nun analysieren wir die Nebenläufigkeiten, die unser System erfüllen muss. Dazu bietet es sich an eine kleine Excel-Liste zu erstellen in der die Nebenläufigkeit, deren Zeitanforderung und die geschätzte Laufzeit eingetragen werden.

Wichtige Eigenschaften, die uns helfen sind:

**Reaktionszeit bzw. Periodizität:** Damit ist die Zeit gemeint, in der eine Nebenläufigkeit reagieren muss. Das kann entweder ereignisorientiert sein oder in einem wiederkehrenden Zeitraster, dann wird diese Zeit auch Zykluszeit genannt.

**Laufzeit:** Das ist die Zeit, die die eigentliche Nebenläufigkeit für ihre eigene Abarbeitung benötigt.

**Implementations-Ebene:** Hier kennen wir grundsätzlich die Interrupt-Ebene und Programm-Ebene. In diesem Fall wird die Programm-Ebene als RTOS-Ebene bezeichnet.

Nebenläufigkeit	Geforderte Reaktionszeit bzw. Periodizität	Geschätzte Laufzeit der Abarbeitung	Implementations-Ebene	Scheduling Aufschlag
Motor Regelung	100 µs Zykl.	?	?	?
Display Ansteuerung	100 ms Ev.	1 ms	RTOS	10 µs
Tastatur Ansteuerung	100 ms Zykl.	100 µs	RTOS	10 µs
Drucküberwachung Pneumatik	10 ms Ev.	1 ms	RTOS	10 µs
Temperatur Überwachung Motorschutz	100 ms Ev.	100 µs	RTOS	10 µs
FFT Regelabweichung Selbstlernroutine	10 s Ev.	100 ms	RTOS	10 µs
Stromregelung für Pneumatikventile	1 ms Zykl.	?	RTOS	10 µs
...	...	...	...	...
CAN Sende- und Empfangsroutinen	10 µs Ev.	?	ISR	800 ns
PWM Motor Ansteuerung	100 µs Zykl.	?	ISR	800 ns
Abfrage Digital Input	1 ms Ev.	1 µs	ISR	800 ns
AD Konvertierung	1 µs Zykl.	1 µs	ISR	800 ns
...	...	...	...	...

### Abbildung: Beispiel Systemanalyse der Laufzeitarchitektur

*(Für eine erste Systemanalyse reicht eine grobe Abschätzung. Werte, die zu diesem Zeitpunkt auch nicht grob geschätzt werden können sind mit einem ? versehen. Im Verlauf des Projektes können die Zahlen dann mit dem Grad der Erfahrung nachgebessert werden. Routinen, die als HW-Treiber ohnehin auf der Interrupt-Ebene realisiert werden sollen, werden sofort als ISR gekennzeichnet.)*

**Scheduling Aufschlag:** Jede Unterbrechung einer Nebenläufigkeit durch eine andere, beinhaltet einen Laufzeit-Overhead. Die Zeit, die eine Unterbrechung verursacht wird auch als Scheduling - oder Kontext Switch Time bezeichnet. Diese ist bei der Auswahl eines Scheduling-Patterns zu berücksichtigen.

Verschiedene RTOS-Systeme können hier wenige µs benötigen, aber auch einige ms.

Aus dem Beispiel der Laufzeitarchitektur-Analyse (Abbildung) werden nun folgende Systemanforderungen abgeleitet:

1. Gibt es Laufzeiten, die länger sind als die kürzesten Reaktionszeiten. Jede

Überschneidung erfordert Unterbrechbarkeit (Preemption) eines Systems.

In obiger Tabelle erkennen wir, dass die Laufzeit der FFT um den Faktor 1000 mal größer ist, als die Reaktionszeiten z.B. der Motor-Regelung. Wäre das die einzige Zeitüberschneidung, dann könnte man sich überlegen die Motor Regelung als ISR (Interrupt-Service-Routine) auf der Interrupt-Ebene zu implementieren und auf einen Scheduling-Pattern auf der Programm-Ebene zu verzichten.

Aber wir sehen, es gibt noch eine weitere Überschneidung: Die Drucküberwachung und die Displayansteuerung mit den geschätzten Laufzeiten im Bereich von 1 ms überschneiden sich mit der Reaktionszeit der Stromregelung der Pneumatik-Ventile. Das gleiche gilt für die Drucküberwachung Pneumatik.

Erfahrungsgemäß wird es im Verlauf der Implementation bzw. im weiteren Projektverlauf tendenziell zu weiteren Überschneidungen kommen, die in einer Grobanalyse nicht erkannt werden. Ich persönlich würde bereits ab zwei potentiellen Überschneidungen den Einsatz eines preemptiven Scheduling-Pattern in Betracht ziehen.

2. Wird auf Basis der obigen ersten Analyse der Einsatz eines RTOS in Betracht gezogen, dann werden nun die auf der Hardware-Architektur möglichen Schedulingzeiten der verfügbaren Betriebssysteme herangezogen, um die notwendigen Reaktionszeiten abzusichern.

Für ein aktuelles System auf Basis einer ARM 7 Architektur mit dem RTOS embOS z.B., läge die Kontext Switch Time bei ca. 8  $\mu$ s. Bei gleichem RTOS auf einem Atmel AVR läge sie bei ca. 50  $\mu$ s. Nun gibt es grundsätzlich verschiedene mögliche Ansätze. An dieser Stelle exemplarisch zwei davon:

- A. Aus Software Engineering Gesichtspunkten soll ein RTOS eingesetzt werden. Das hätte dann Auswirkungen auf die Wahl der CPU. In diesem Fall würde der AVR grenzwertig in der Prozessorleistung sein.
- B. Aus HW-Kostengründen steht als CPU bereits der AVR fest, dann würde die Reaktionszeit der Motor-Regelung mit 100  $\mu$ s gegenüber der Schedulingzeit von 50  $\mu$ s grenzwertig sein. In diesem Fall müsste diese Funktion als ISR implementiert werden. Warum? Nehmen wir einmal die Schedulingzeit von

50  $\mu$ s, dann ergibt sich bei einer zyklischen Reaktionszeit von 100  $\mu$ s bereits 50 % Belastung der Rechenleistung der CPU, wenn diese Routine auf RTOS-Ebene implementiert werden soll. Eine Faustformel sagt, dass die minimalen Reaktionszeiten Faktor 10 länger sein sollten, als die Schedulingzeit.

Auch die Routinen „Stromregelung für Pneumatik-Ventile“ müssen als ISR implementiert werden. Um ein Gitter in der Reaktionszeit zu vermeiden, muss diese Routine die Drucküberwachung, die Displayansteuerung und die FFT unterbrechen können.

Im Fall B. würden mehr Routinen auf der Interrupt-Ebene implementiert werden müssen, als auf der eigentlichen Programm-Ebene.

3. Wir erkennen, dass der überwiegende Teil der Nebenläufigkeiten ereignisgetriebenen Charakter hat. Die Sensoranbindung hält sich die Waage. In diesem Fall würde ich zu einer ereignisgetriebenen Implementation des Laufzeitarchitekturdesigns tendieren.

Grundsätzlich lassen sich die beiden Welten zeit- und ereignisgetrieben ineinander transformieren. Aber das bedeutet Aufwand. Dieser Aufwand kann leicht vermieden werden, wenn der grundsätzliche Charakter der Laufzeit günstig gewählt wird.

## Resümee

Die Zeitanalyse der Nebenläufigkeiten liefert die grundlegenden Informationen für die HW- und SW-Architektur-Entscheidungen. Hier noch einmal die wichtigsten Basisinformationen für eine Architekturentscheidung:

- Überschneidungen von Reaktionszeit und Laufzeit erfordern preemptives Verhalten und sprechen, wenn sie häufiger vorkommen, für den Einsatz eines RTOS
- Reaktionszeiten der Nebenläufigkeiten und Schedulingzeiten des Laufzeitsystems geben Hinweise auf die Implementations-Ebene der Nebenläufigkeiten. Grundsätzlich sollte die Interrupt-Ebene wenigen zeitkritischen Routinen vorbehalten bleiben.
- Die Multiplikation der Reaktionszeiten mit der Summe aus Laufzeit und Schedulingzeit gibt eine grobe Aussage über die notwendige Rechenperformance der HW (In der Praxis können grobe Richtwerte

häufig aus den Erfahrungen vorheriger Projekte abgeleitet werden).

- Die grundsätzlichen Charaktere der Peripherie und der Nebenläufigkeiten geben den Ausschlag für ein grundsätzliches zeit- oder ereignisgetriebenes Laufzeitarchitekturdesign.
- Bei mehr als fünf Nebenläufigkeiten sollte heute der Einsatz eines RTOS immer in Erwägung gezogen werden.

Weitere Informationen zu Scheduling Pattern:

<http://de.wikipedia.org/wiki/Prozess-Scheduler>

## Welches RTOS ist das Richtige?

### Reaktionszeiten

Sie haben sich dafür entschieden ein RTOS einzusetzen und jetzt gilt es das Richtige auszuwählen. *(An dieser Stelle wird vorausgesetzt, dass das RTOS die Basis-Scheduling-Mechanismen unterstützt. Das ist heute bei nahezu allen am Markt verfügbaren Betriebssystemen der Fall, sollte aber im Zweifel überprüft werden.)*

Aus unserer Systemanalyse der Nebenläufigkeiten kennen wir die geforderte Scheduling-Zeit. Hier trennen sich schon einmal grob die verschiedenen Systeme. Grundsätzlich könnten Real Time Operating Systeme, wie der Name bereits sagt, ihrem Echtzeitverhalten entsprechend zugeordnet werden.

1. Ressourceneffiziente Betriebssysteme mit wenig Diensten und sehr schnellem Zeitverhalten.
2. Betriebssysteme, die viele Dienste und Treiber zur Verfügung stellen, dafür aber auch entsprechende Ressourcen benötigen und entsprechend langsam reagieren.

An dieser Stelle grob exemplarisch die Reaktionszeiten verschiedener Kategorien von Betriebssystemen orientiert an einer ARM7 HW-Architektur:

Schedulingzeiten und Interrupt-Latenzzeiten liegen bei den folgenden kleinen effizienten Betriebssystemen im Bereich von wenigen  $\mu$ s: ARTX (enthalten in der RT Lib der Firma Keil/ARM), embOS ( Fa. Segger), FreeRTOS, (FreeRTOS.ORG), CMX RTX (Fa. CMX Systems),  $\mu$ C OS (Fa. Micrium)

Schedulingzeiten und Interrupt-Latenzzeiten liegen bei den folgenden komplexeren Betriebssystemen im Bereich von einigen 100  $\mu$ s bis hin zu einigen ms: Windows CE (.NET), LINUX, VxWorks.

Dazwischen bietet der Markt eine große Auswahl an Betriebssystemen: EUROS, PXROS, QNX, Enea OSE ... um nur einige zu nennen.

Die Abstimmung der notwendigen Reaktionszeiten in Kombination mit dem RTOS und einer entsprechend performten HW-Architektur ist die Basis einer jeden Laufzeit-Architekturdesign- Entscheidung. Was hilft Ihnen der Einsatz eines RTOS, welches in Kombination mit der gewählten Hardware-Architektur nicht die notwendigen Schedulingzeiten erfüllt und sie dann die Hälfte Ihrer Nebenläufigkeiten auf der Interrupt-Ebene implementieren müssen, wo sie die meisten Dienste eines RTOS nicht in Anspruch nehmen können?

### Abdeckung der benötigten Dienste und Funktionen

Heutige Applikationen müssen über Bussysteme kommunizieren, sollen PC-Kompatible auf Speicherkarten lesen und schreiben können, haben komplexe Displaysteuerung, sollen dynamisch Softwareupdates durchführen, Web Services zur Verfügung stellen...

Für viele dieser Anforderungen bieten die unterschiedlichen Betriebssystemhersteller bereits fertige Lösungen. Der Grad der Abdeckung der eigenen Anforderungen an Diensten und Treibern fließt selbstverständlich in den Entscheidungsprozess mit ein. Hier können die komplexeren Betriebssysteme wie Windows CE oder LINUX ihre Karten ausspielen.

### Betriebsbewährtheit

Ein weiteres nicht unwichtiges Entscheidungskriterium ist die Betriebsbewährtheit des RTOS in Kombination mit Compiler, IDE und Prozessor - Architektur.

An dieser Stelle gibt es besser oder schlechter bewährte Kombinationen. Muss ein RTOS erst noch für eine spezifische Kombination angepasst werden, dann sollten Sie sich ernsthaft überlegen, ob Sie nicht auf die bereits vorhandene Kombination setzen und gegebenenfalls die Wahl des Compilers überdenken oder ein anderes RTOS in Betracht ziehen.

Sie möchten z.B. die ARM7 Architektur auf Basis des GNU Compilers einsetzen und haben ein RTOS in die engere Wahl gezogen, welches jedoch nur

den Keil/ARM Compiler unterstützt. (In dieser Kombination wird das RTOS schon von zahlreichen Kunden eingesetzt).

In diesem Fall würde ich persönlich immer die Compilerentscheidung der RTOS Auswahl hinten anstellen. Es kann natürlich sein, dass triftige Gründe für einen bestimmten Compiler sprechen. Aber 3.000,- € Ersparnis für den kostenlosen GNU Compiler wären für mich kein triftiger Grund, eher schon die Übernahme eines Projektes, dass auf Basis des GNU Compilers entwickelt wurde.

## Weitere Entscheidungskriterien

### Sourcecode

Ist das RTOS im Sourcecode verfügbar? Viele Entwickler denken, der Sourcecode wird benötigt, um eigene Änderungen durchführen zu können. Diesen Grund halte ich für nicht so wichtig. Für die Wartung und Pflege ist der RTOS - Hersteller zuständig. Viel interessanter ist der Sourcecode z.B. beim Debuggen. Es kann sehr unangenehm sein, wenn Sie beim Single Stepp durch das System immer wieder im Disassembler landen, wenn Sie in die RTOS - Routinen steppen.

### System Level Debugging

Trotz einem guten Architektur-Design und einem betriebsbewährten RTOS wird Ihre Applikation nicht immer sofort fehlerfrei laufen. Hier hilft ein sogenannter System Level Debugger. In ihm können Sie die Ressourcen und Abläufe Ihrer Architektur dynamisch betrachten, um Fehler einfacher zu finden. Ein solcher Debugger lohnt sich auf jeden Fall und die ausgewählte Toolkette sollte einen System Level Debugger enthalten. Es hängt von der Wahl des RTOS und Debuggers ab. In manchen Fällen wird der Debugger mit dem RTOS geliefert, in anderen gibt es eine definierte Schnittstelle zum Debugger.

### Preis

Echtzeitbetriebssysteme müssen heute nicht mehr teuer sein. Es gibt viele Systeme, deren Preise im Bereich von wenigen tausend Euro pro Arbeitsplatz liegen. Für diesen Betrag lässt sich nicht einmal die Funktionalität eines simplen RTOS selber entwickeln.

Anders wird es, wenn zu den Entwicklungslizenzen noch sogenannte Vertriebslizenzen (auch Royalties genannt) hinzukommen. Einige der RTOS-Anbieter haben derartige Preismodelle. Dann kann der Einsatz eines RTOS schnell in den Bereich von mehreren zehntausend Euro ansteigen. Auch das

kann rentabel sein, wenn die gelieferte Lösung entsprechende Vorteile bietet. Diese Konzepte kommen aber zunehmend aus der Mode, weil sich häufig Alternativen anbieten.

Open Source Betriebssysteme, wie beispielsweise LINUX, sind oftmals nur auf den ersten Blick kostenlos. Diese Systeme müssen evtl. an eigene Hardware angebunden werden oder es muss ein sogenanntes Board Support Package gekauft werden. Am Ende entstehen auch dort Kosten, die nicht zu vernachlässigen sind.

## Abschließende Betrachtung

Im Lauf der Zeit sind die Anforderungen an heutige Applikationen schleichend, aber permanent gewachsen. Nicht unbedingt nur neue Funktionalitäten sind die Ursache für steigende Komplexität, sondern deren Ineinandergreifen.

Entkopplung ist der wirksamste Mechanismus dem zu begegnen. Die Grundlage für ein entkoppeltes Software System liegt im Architekturdesign und hier an erster Stelle das Laufzeit - Architekturdesign. In der Praxis ist aus diesem Grund der Einsatz eines RTOS eine sehr gute Möglichkeit der steigenden Komplexität zu begegnen.

Wenn Sie noch nicht ganz sicher sind, ob und wie auch in Ihrem Fall ein RTOS helfen kann das Architektur Design zu verbessern, dann nehmen Sie sich 1 Tag Zeit und besuchen unseren

Workshop:

### Software-Architektur-Design für Embedded Systeme

Termine: **22.09.2009 in Hannover**  
**23.09.2009 in Ulm**

Preis: **199,- € zzgl.gesetzl.Mwst**  
**incl. Verpflegung**

Infos unter: **[willert.de/events](http://willert.de/events)**

Anmeldungen:**[willert.de/anmeldung](http://willert.de/anmeldung)**

Ein Workshop, den jeder Embedded Software-entwickler besuchen sollte, der ein neues Projekt startet und von Anfang an die Grundlagen für eine langlebige Software-Architektur legen möchte.

Noch ausführlicher behandeln wir das Thema Software Architektur-Design in der neuen Ausgabe unserer Newsletter No 21:

<http://www.willert.de/newsletter/>

