

»MONOLITHISCHE« ECHTZEITBETRIEBSSYSTEME ERWEITERN

»Apps« fürs RTOS

Meist besitzen sehr kompakte Echtzeitbetriebssysteme (RTOS) eine Architektur, die den Applikationscode direkt an die von ihm genutzten RTOS-Services linkt und daraus ein zusammenhängendes, ausführbares Image bildet. Allerdings besteht zunehmend der Wunsch, auch solche Anwendungen später im Feld zu updaten. Das »App«-Konzept, wie vom »iPhone« bekannt, kann da helfen. Solche downloadfähigen Applikationsmodule verleihen kleinen Embedded Systemen die Eigenschaften großer Systeme.

JOHN A. CARBONE

Embedded Systeme, die in der Unterhaltungselektronik, Medizintechnik oder in industriellen Applikationen zum Einsatz kommen, müssen oft in Echtzeit reagieren können, damit der Nutzer sie positiv wahrnimmt. Dabei ist es unerheblich, ob es um die Basisband-Funkkommunikation eines Smartphones, die Verarbeitung von Ultraschallbildern oder die Videoinspektion an einer Fertigungslinie geht. Diese und viele weitere Systeme müssen Eingangsinformationen rasch verarbeiten und mit der

Ausgabe von Informationen oder mit bestimmten Aktionen reagieren, sei es für einen menschlichen Anwender oder eine andere Maschine. Systeme dieser Art sind mit stromsparenden Prozessoren ausgestattet und erledigen ihre Aufgaben häufig mit relativ wenig Speicher, was die Entwickler nicht selten veranlasst, ein Echtzeitbetriebssystem (Real-Time Operating System, RTOS) zu verwenden. Das RTOS verwaltet die Tasks oder Threads der Applikation, widmet sich dem Interrupt-Handling und sorgt für die Kommunikation und Synchronisation der Threads untereinander.

Echtzeitbetriebssysteme gibt es in sämtlichen Größen und Varianten, sehr umfangreiche wie beispielsweise »VxWorks« von Wind River bis zu sehr kompakten wie »ThreadX« von Express Logic. Größere RTOS-Produkte bringen zahlreiche von Desktop-Systemen übernommene Features mit, die in kompakten Echtzeitbetriebssystemen jedoch meist fehlen, da sie mehr Code und mehr Speicher erfordern und die Reaktionsgeschwindigkeit verringern. Ein kompaktes RTOS ist außerdem üblicherweise als eine Bibliothek aus Diensten (Services) angelegt, auf die die Applikation mit direkten Funktionsaufrufen zugreift (Bild 1). Unterstützt werden diese RTOS-Services durch eine Infrastruktur aus Scheduling und Kommunikationsfunktionen. Die meisten RTOS-Produkte mit kleinem Footprint besitzen eine Ar-

chitektur, die den Applikationscode direkt an die von ihm genutzten RTOS-Services linkt und daraus ein zusammenhängendes, ausführbares Image bildet. Die Applikation referenziert dabei explizit die von ihr benötigten Dienste und nutzt hierfür Funktionsaufrufe im Rahmen eines vom RTOS definierten API. Diese Dienste sind mit der Applikation aus der RTOS-Bibliothek gelinkt, und es entsteht ein zusammenhängendes, ausführbares Image (meist im .elf-Format). Für die Entwicklung wird dieses Image in den Speicher des Zielsystems heruntergeladen und ausgeführt. Anders ist es bei der Serienfertigung: Dort brennt man ein ROM mit dem Image, das dann beim Einschalten der Applikation gestartet wird.

»Monolithische« Konzepte sind unflexibel

Dieses »monolithische« Konzept ist in Sachen Zeit- und Platzaufwand durchaus effizient, doch mangelt es ihm an Flexibilität. Jede Änderung an der Applikation oder dem RTOS erfordert ein erneutes Linken sowie ein erneutes Herunterladen beziehungsweise Brennen des gesamten Images. Während der Entwicklung mag dies Routine sein, aber nach erfolgter Produktion ergeben sich doch einige Einschränkungen. Bei Desktop-Betriebssystemen wie Windows und Linux, aber auch bei größeren Echtzeitbetriebssystemen sind die Verhältnisse anders: Sie sehen eine Zweiteilung in Betriebssystem und Applikation vor (Bild 2). Ein residenter Kernel enthält hier

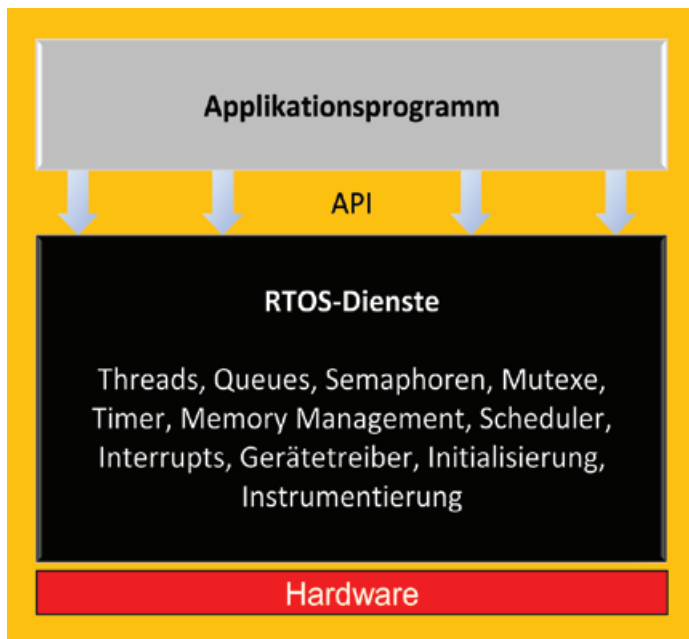


Bild 1: Die meisten Echtzeitbetriebssysteme stellen bestimmte Dienste und eine zugrunde liegende Infrastruktur zur Verfügung, die den Applikationen den Betrieb auf einer Hardwareplattform ermöglichen

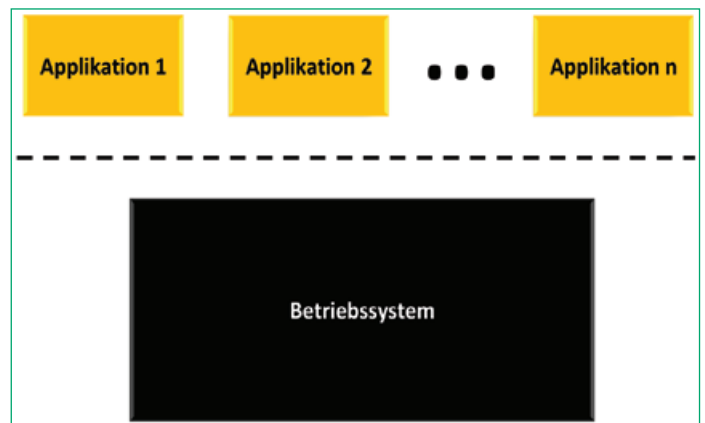


Bild 2: Große Echtzeitbetriebssysteme sind ebenso wie Desktop-Betriebssysteme in einen eigenständigen Kernel und unabhängige Applikations-Executables unterteilt, die über eine Trap-Schnittstelle auf den Kernel zugreifen

alle Betriebssystemdienste, die den Applikationen zur Verfügung stehen oder wiederum von anderen Diensten im Kernel benötigt werden, und das Ganze ist zu einer spezifischen ausführbaren Datei gelinkt. Diese ausführbare Kernel-Datei bootet das System, läuft permanent und schafft eine Grundlage für die dynamisch geladenen und ausgeführten Applikationen. Üblicherweise dient virtueller Speicher für das Demand-Paging von und zum Massenspeicher (im Fall von Desktop-Systemen) oder die Abgrenzung zwischen mehreren Usern (in Embedded Systemen). Dieses Konzept kommt in mobilen Geräten wie dem »iPhone« oder »iPad« von Apple zum Einsatz, auf die aus dem Netz neue Applikationen, »Apps« genannt, geladen werden können. Das Betriebssystem läuft in dem Gerät und unterstützt die Benutzeroberfläche, welche die Auswahl der jeweils gewünschten heruntergeladenen App ermöglicht. Die ausgewählte App läuft anschließend zusammen mit dem Betriebssystem auf der CPU. Ähnlich ist es bei großen RTOS-basierten Systemen, die ihre Applikationen vom RTOS-Kernel abgrenzen und ihnen in einer virtuellen Speicherumgebung meist eigene Speicherbereiche zuweisen. Eine sinnvolle Eigenschaft großer Echtzeitbetriebssysteme, die man auch bei Desktop-Betriebssystemen vorfindet, ist ihre Fähigkeit zum dynamischen Herunterladen

JOHN A. CARBONE



ist Vice President of Marketing bei Express Logic

von Applikationen in ein laufendes System. In solcherart gegliederten Architekturen läuft der Kernel als eine eigene Instanz. Die Applikationen arbeiten unabhängig vom Kernel, nutzen aber die Betriebssystemdienste für den Zugriff auf Hardware-Ressourcen und für deren Nutzung. Auch in Embedded Systemen findet man dieses Herunterladen sowie das dynamische Hinzufügen oder Auswechseln von Applikationen vor, wenn große RTOS-Produkte in der Telekommunikationsinfrastruktur oder in Aerospace-Applikationen zum Einsatz kommen. Diese Auslegung ergibt nicht nur einen hohen Grad an Modularität, sondern lässt auch Updates an laufenden Systemen im Feld zu.

Eignet sich der »Trap«-Mechanismus?

Allerdings sind diese Applikationen weder monolithisch noch an das RTOS gebunden. Der Zugriff auf die RTOS-Dienste erfolgt daher mit einem »Trap«-Mechanismus. Darunter versteht man einen Software-Interrupt, der von einem der zahlreichen, von einer Architektur zu anderen unterschiedlichen Mechanismen ausgelöst wird. Norma-

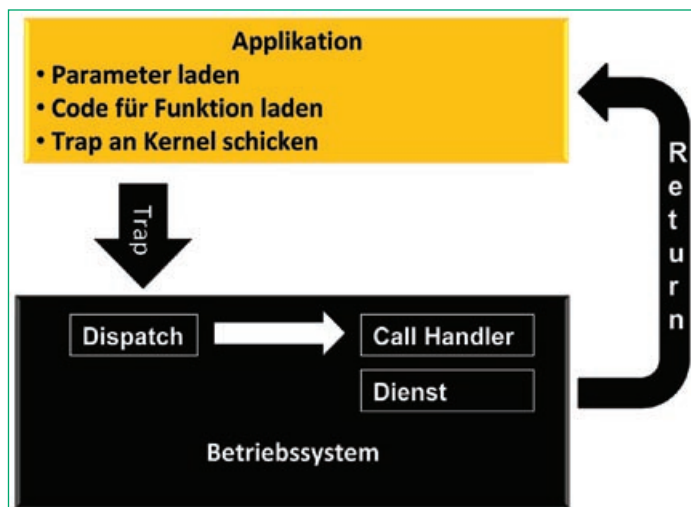


Bild 3: Mithilfe eines Trap-Mechanismus kann Programmcode außerhalb des Kernels mit dem Kernel kommunizieren. Dies ist aber sehr aufwändig und daher für kleine Betriebssysteme ungeeignet.

TechTour.Net

Seit 10 Jahren



Vergleichsangebote per Mausclick
www.TechTour.net

Leiterplattenbestückung EMS
Leiterplatten
Kabelkonfektion
Folientastaturen
Schaltmatten
Gehäuse Technik
Baugruppenfertigung

NORTEC

13. Fachmesse für Produktionstechnik
25. – 28. Januar 2012 | Hamburg

Treffpunkt der
Entscheider des Nordens.

NORTEC 2012:
Nicht verpassen!

nortec-hamburg.de



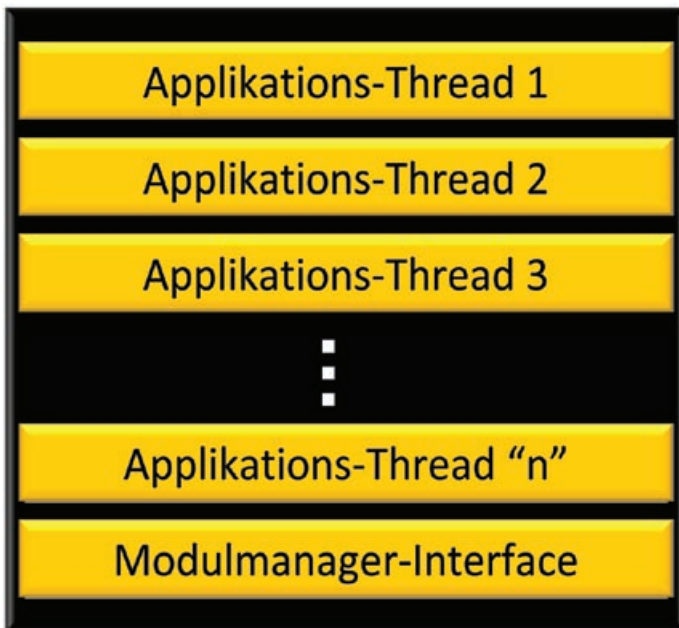


Bild 4: Herunterladbare Applikationsmodule sind ausführbare Dateien mit einem oder mehreren Applikations-Threads und einem Schnittstellenmechanismus für den Zugriff auf die Kernel-Dienste

lerweise fängt eine Trap Fehler ab, bevor sich diese weiter fortpflanzen, oder hält das System an, wenn es nicht möglich ist, die angeforderte Operation auszuführen. Eine Division durch Null oder das Laden falsch ausgerichteter Daten sind Beispiele für illegale Operationen, die ebenso wie ein externer Interrupt einen Interrupt-Handler aktivieren würden. In anderen Fällen lässt sich mit einem Befehl (z.B. einem Software-Interrupt oder einem »swik«-Befehl) absichtlich eine Trap auslösen.

Wenn das OS die Anforderung eines Betriebssystemdiensts durch eine Applikation bearbeiten muss, kann einer dieser Mechanismen zum absichtlichen Auslösen einer Trap verwendet werden. Möchte eine Applikation beispielsweise einen Dienst nutzen, der zwar im Kernel verfügbar ist, sich aber nicht direkt von der Applikation aufrufen lässt, kann die Applikation eine Trap auslösen (Bild 3). Der Trap-Handler fragt dann ein Register ab (hierin legt der Prozessor in der Regel unmittelbar vor dem Generieren der Trap einen identifizierbaren Wert ab), um zu ermitteln, welches Ereignis die Trap ausgelöst hat. Stellt der Handler fest, dass es um einen Dienstaufwurf geht, wertet er die Inhalte weiterer Register aus (die vom anfordernden

Modul geladen wurden) und stellt dadurch fest, welcher Dienst angefordert wird. Anschließend holt er die Parameter für den betreffenden Dienst aus den Registern, die von der Applikation zuvor geladen wurden. Schließlich ruft der Trap-Handler den Dienst (der Bestandteil seiner ausführbaren Datei ist) als gelinkte Funktion mit den spezifizierten Parametern auf.

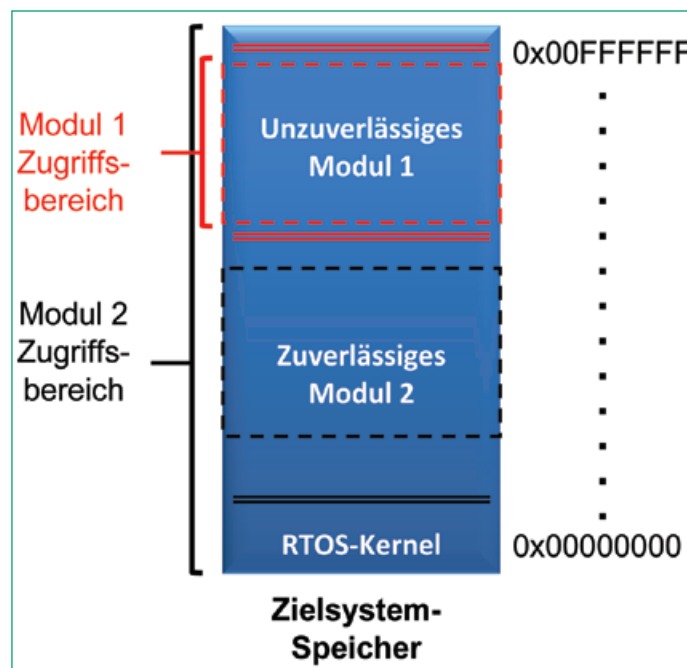


Bild 6: Mithilfe einer MMU (Memory Management Unit) oder MPU (Memory Protection Unit) lassen sich Module und das RTOS vor fehlerhaften Zugriffen durch andere Module schützen

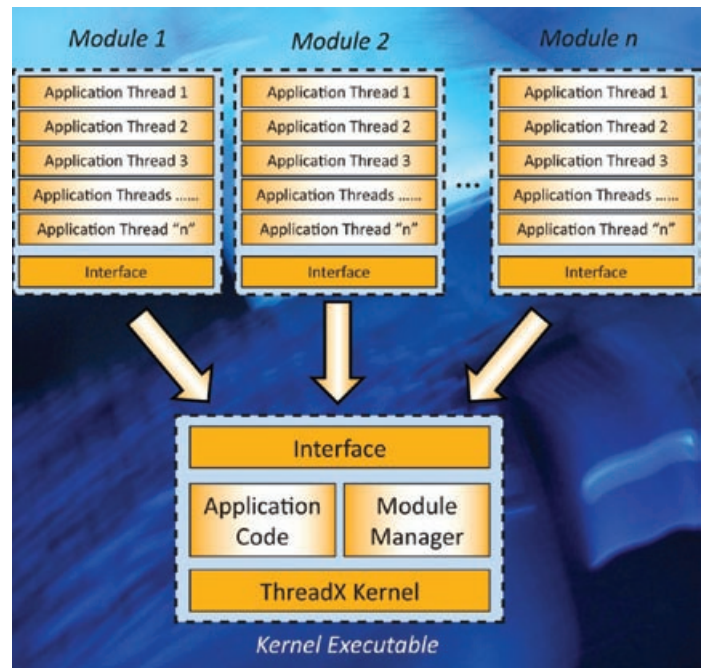


Bild 5: Kritische Applikationen lassen sich mit dem Kernel linken, während man andere in separate Module auslagert

Dieser gesamte Prozess erfordert eine Interrupt-Behandlung, einen gewissen Verarbeitungsaufwand und einen Funktionsaufruf und anschließend das gleiche in umgekehrter Reihenfolge. Im Falle eines Desktop-Betriebssystems oder eines großen RTOS fällt dieser Aufwand in Anbetracht des großen Umfangs an weiterem Code, der ständig für Routineaufgaben zu verarbeiten ist,

nicht ins Gewicht. Irrelevant ist dieser Aufwand auch mit Blick darauf, dass die Erledigung der Aufgaben für das System keine Dringlichkeit hat. Ein großes Betriebssystem kann man mit dem trägen Verhalten eines Lastwagens vergleichen: Dass dieser nicht sehr stark beschleunigen kann und auch nicht besonders wenig ist, stellt kein Problem dar, denn die meiste Zeit rollt er über die Autobahn dahin.

Bei kleinen RTOS-Produkten für harte Echtzeitanwendungen dagegen kommt es auf eine schnelle Reaktion mit wenig Ballast an. Solche gleichen eher Autos für den dichten Stadtverkehr, die auf Wendigkeit angewiesen sind. Eine Trap-Schnittstelle ist daher viel zu aufwändig, weil sie gegenüber der hohen Effizienz der direkt verknüpften Zugriffsmethode übermäßig stark zu Buche schlägt.

Modul-Manager bearbeitet Aufrufe effizienter

Bei einem Echtzeitbetriebssystem mit kleinem Footprint muss man die dynamisch herunterladbaren Applikationen dazu bringen, effiziente Aufrufe an RTOS-Dienste zu richten, die sich in einem abgegrenzten, separat gelinkten Code-Element befinden. Erforderlich ist hierfür eine Schnittstelle, die den Applikationen die Dienste effizient

zur Verfügung stellt und gleichzeitig die Vorteile des dynamischen Downloads bietet.

Um kleinen Echtzeitbetriebssystemen eine solche Architektur zu verleihen, wird eine »Applikationsmodul«-Struktur benötigt. Applikationsmodule bestehen aus einem oder mehreren Applikations-Threads, die nicht mit dem Kernel gelinkt, sondern als separate ausführbare Dateien angelegt sind, die bei Bedarf in den Speicher des Zielsystems geladen werden (Bild 4). Die Module nutzen die Dienste des Kernels über eine Schnittstelle zum Modul-Manager (Bild 5). Dieser im Kernel-Image enthaltene Agent lädt und initialisiert die Module und koordiniert alle Anforderungen von RTOS-Diensten durch die Module. Die Threads innerhalb der Module führen ihre Dienstaufträge so aus, als wäre der Dienst direkt an die Applikation gelinkt. In dem Modul werden diese Aufrufe jedoch von einer Schnittstellenkomponente behandelt, die mit dem Modul-Manager kommuniziert. Durch den Wegfall des Trap-Mechanismus lassen sich Dienstaufträge mit wenig Verarbeitungsaufwand verarbeiten.

Vom Modul-Manager gibt es nur eine einzige Instanz. Dennoch können unbegrenzt viele Module gleichzeitig geladen sein, und auch für die Zahl der Threads pro Modul gibt es keine Obergrenze. Man erreicht hiermit, dass der Kernel in einer abgegrenzten Verarbeitungsinstanz residiert, die fortlaufend aktiv ist, um Aufrufe seitens der Module bedienen zu können.

Um die Effizienz zu maximieren, können die Applikations-Threads alternativ auch an den Kernel gelinkt werden und zusammen mit dem Kernel als Bestandteil dessen ausführbarer Image-Datei im Speicher des Zielsystems residieren. Diese Option vermeidet das Laden der Module, die diese Threads enthalten, und ergibt die effizienteste Schnittstelle. Allerdings wird das Kernel-Image hierdurch größer, sodass weniger Speicherplatz für die Module bleibt. Auch wird die Chance vergeben, die Applikations-Threads dynamisch auszutauschen.

Mithilfe downloadfähiger Applikationsmodule kann das RTOS zusätzliche Applikations-Threads neben jenen, die an den Kernel gelinkt sind, dynamisch laden und verarbeiten. Die Applikationen erhalten hierbei zusätzliche Funktionen, ohne einen größeren Footprint oder einen erhöhten Speicherbedarf in Kauf nehmen zu müssen. Überdies bleibt der Vorteil einer effizienten Schnittstelle für Dienstaufträge erhalten. Nicht zuletzt lassen sich bereits im Einsatz befindliche Systeme bedarfsorientiert rekonfigurieren und updaten. Die Technik der downloadfähigen Applikationsmodule eignet sich besonders für Situationen, in denen der Applikationscode nicht in den verfügbaren Speicher passt, in denen nach der Installation eines Produkts neue Module hinzugefügt werden müssen oder wenn partielle Firmware-Updates notwendig sind.

Ein weiterer Vorteil des Herunterladens separater Applikationsmodule ist die Tatsache, dass die einzelnen Module von verschiedenen Teams oder einzelnen Programmierern entwickelt werden können. Das jeweilige Team kann sich dann auf einen bestimmten Aspekt der Gesamtfunktion eines Produkts konzentrieren, ohne sich mit Details befassen zu müssen.

Fehlfunktionen lassen sich vermeiden

Nachteilig an einem solchen Konzept ist, dass das Risiko von Fehlfunktionen steigt, wenn das Modul in das System eingefügt wird und mit dem Kernel sowie mit anderen Modulen interagieren muss. Die Entwickler hegen verstärkte Bedenken, ob sich die externen Module mit den anderen vertragen. Noch schlimmer als ein Programm-Bug, der Fehlfunktionen verursacht oder einen Absturz des Moduls verursacht, ist ein Bug, der ein anderes Modul oder gar den RTOS-Kernel selbst kontaminiert.

Ungeachtet aller Tests bleibt das Risiko einer zufälligen Korruption des Stacks oder des Speichers infolge eines falsch berechneten Zeigers, einer falsch berechneten Arraygrenze oder eines Stack-Überlaufs erhalten. Derartige Fehler können katastrophale Folgen haben und außerdem schwierig zu finden sein – besonders dann, wenn nur ein Teil des Entwicklungsteams mit dem betreffenden Modul vertraut ist. Noch schwieriger wird es, wenn die Ursache für einen solchen Fehler in den betreffenden Code zurückverfolgt wird und sich dieser in einem ganz anderen Modul befindet. Da das Einkreisen der Ursache solcher katastrophalen Fehler so enorm schwierig ist, ist es umso wichtiger, diese Fehler von vornherein zu vermeiden.

Um Systeme vor dieser Art zufälliger Beeinträchtigung zu schützen, ist es sinnvoll, die Threads eines Moduls vom Zugriff auf Speicherstellen außerhalb des eigenen Bereichs abzuhalten. Stattdessen sollten RTOS-Dienste für das Message-Passing angeboten werden. Diese Schutzmaßnahmen bewahren Module und den Kernel vor unbeabsichtigter Korruption durch fehlerhafte Schreib- oder Sprung-Operationen und auf Wunsch auch durch falsche Lesevorgänge. Mit der MMU (Memory Management Unit) oder MPU (Memory Protection Unit) des Systems lassen sich Speichergrenzregister einrichten, die dafür sorgen, dass der Code eines Moduls nur auf den eigenen Speicherbereich zugreifen kann (Bild 6). (rh)

Express Logic
Telefon: 0 51 43/91 13 04
www.rtos.com

You CAN get it...

Hardware und Software für CAN-Bus-Anwendungen...



PCAN-miniPCI

CAN-Interface für Mini PCI-Steckplätze. Optional mit galvanischer Trennung. Als Ein- und Zweikanal-Karte erhältlich.

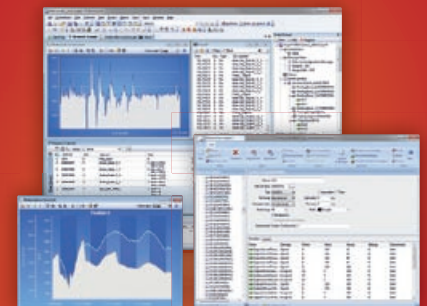
ab 200 €



PCAN-Router

Frei programmierbarer CAN-Router mit 2 High-Speed-CAN-Kanälen. Mit D-Sub- oder Phoenix-Anschlusssteckern erhältlich.

ab 200 €



PCAN-Explorer 5

Universeller CAN-Monitor, Tracer, symbolische Nachrichtendarstellung, VBScript-Schnittstelle, erweiterbar durch Add-ins (z. B. Plotter Add-in).

ab 450 €

Alle Preise verstehen sich zzgl. MwSt., Porto und Verpackung. Irrtümer und technische Änderungen vorbehalten.

www.peak-system.com

PEAK
System

Otto-Röhm-Str. 69
64293 Darmstadt / Germany
Tel.: +49 6151 8173-20
Fax: +49 6151 8173-29
info@peak-system.com