Echtzeitprogrammierung und Echtzeitverhalten von Keil RTX

Frank Erdrich
Hochschule Offenburg
frank@erdrich.net

19. Dezember 2008

Abstract: Echtzeitsysteme werden immer komplexer. Darum bedarf es Hilfsmittel wie Echtzeitbetriebssysteme, die die Komplexität vereinfachen. Das vorliegende Paper zeigt anhand gängiger Verfahren zur Echtzeitprogrammierung auf, inwieweit das Echtzeitbetriebssystem RTX von Keil Echtzeitprogrammierung und Echtzeitverhalten unterstützt.

1 Einleitung

In gleichem Maße, wie die Entwicklungsgeschwindigkeit heutiger Produkte, im speziellen eingebettete Systeme bestehend aus Hard- und Software, steigt, so steigt auch die Komplexität der Systeme.

Um die steigende Komplexität und die Echtzeitanforderungen eines eingebetteten Systems in den Griff zu bekommen, werden kleine Betriebssysteme eingesetzt. Das Betriebssystem, RTOS[1] genannt, ermöglicht u.a. die Parallelisierung von Aufgaben. Diese Betriebssysteme müssen aber gewissen Anforderungen genügen, um als Echtzeitbetriebssystem eingesetzt werden zu können.

1 Real Time Operating System

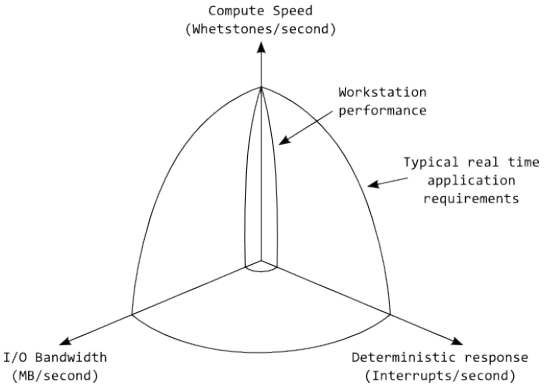

Abbildung 1: Requirements Space einer Echtzeitanwendung

Bild 1 (nach [HGCD97]) zeigt beispielsweise den Anforderungsraum an eine Echtzeitanwendung. Während es bei einem normalen Rechner (beispielsweise einem Office-Computer) hauptsächlich auf die Verarbeitungsgeschwindigkeit ankommt, ist in Echtzeitsystemen auch die I/O-Geschwindigkeit sowie eine deterministische Antwortzeit wichtig, da beide auf die Ausführungszeit einer Applikation größeren Einfluss haben und diese empfindlich stören können.

In den nachfolgenden Kapiteln wird der derzeitige Stand der Echtzeitprogrammierung und des Echtzeitverhaltens von Echtzeitbetriebssystemen vorgestellt. Anschließend werden diese Konzepte beim Keil RTOS RTX näher beleuchtet und ausgewertet.

2 Echtzeit und Echtzeitsysteme

Nicht jedes System ist ein Echtzeitsystem. Oft wird der Begriff Echtzeit fälschlicherweise gebraucht oder mißbraucht. Die *DIN*[1] *44300* definiert den Begriff Echtzeit wie folgt:

"Unter Echtzeit versteht man den Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen."

2.1 Echtzeitsysteme

Das Oxford Dictionary of Computing definiert Echtzeitsysteme wie folgt: "Ein Echtzeitsystem ist ein System, bei dem der Zeitpunkt, zu dem Ausgaben erzeugt werden, bedeutend ist. Das liegt für gewöhnlich daran, dass die Eingabe mit einigen Änderungen der physikalischen Welt korrespondiert und die Ausgabe sich auf die Änderungen beziehen muss. Die Verzögerung zwischen der Zeit der Eingabe und der Zeit der Ausgabe muss ausreichend klein für eine akzeptable Rechtzeitigkeit sein."

Ein Echtzeitsytem ist also ein System, dass auf eine Änderung der Umgebung innerhalb einer gewissen, kurzen und deterministischen, Zeitspanne reagieren muss. Ein Echtzeitsystem muss also nicht nur logisch korrekte Ergebnisse erzeugen, sie müssen auch innerhalb einer vorgegeben Zeit vorliegen. Ist dies, selbst bei richtigem Ergebnis, nicht der Fall, ist das Verhalten fehlerhaft. [HGCD97]

Aus dieser Erklärung ergibt sich die **Rechtzeitigkeit** als Anforderung an Echtzeitsysteme. Die Zeitbedingungen lassen sich in verschiedenen Varianten definieren [WB05].

Angabe eines genauen Zeitpunktes Eine Aktion hat genau zu einem definierten Zeitpunkt stattzufinden, nicht früher oder später.

Angabe eines spätesten Zeitpunktes Eine Aktion muss bis spätestens zu diesem Zeitpunkt durchgeführt sein, kann aber auch früher beendet werden.

Angabe eines frühesten Zeitpunktes Eine Aktion darf erst ab oder nach dem definierten Zeitpunkt ausgeführt werden, jedoch nicht davor.

Angabe eines Zeitintervalls Eine Aktion muss innerhalb zweier definierter Zeitpunkte ausgeführt werden.

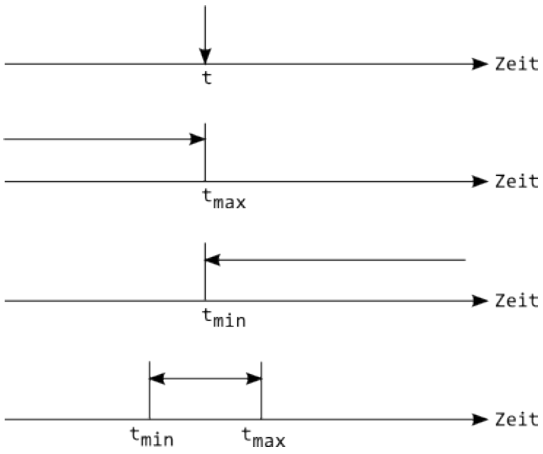

Abbildung 2: Zeitbedingungen von oben nach unten: genauer, spätester, frühester Zeitpunkt, Zeitintervall (nach [WB05])

Desweiteren lassen sich diese Bedingungen in periodische und aperiodische Zeitbedingungen unterteilen. Während periodische Bedingungen in regelmäßigen Abständen wiederholen, treten aperiodische Bedingungen unregelmäßig auf.

Zusätzlich sind noch absolute Zeitbedingungen, also Bedingungen, die zu einer absoluten Zeit ausgeführt werden müssen und relative Zeitbedingungen, die von einem vorherigen Ereignis abhängen, zu erwähnen.

In einem Echtzeitsytem können die genannten Zeitbedingungen in jeder Kombination auftreten. Um die Rechtzeitigkeit gewährleisten zu können muss ein Echtzeitsystem zwei wichtige Eigenschaften aufweisen, hinreichende Verarbeitungsgeschwindigkeit und zeitliche Vorhersagbarkeit. Nur wenn diese Eigenschaf-

1 Deutsche Industrienorm

ten erfüllt werden ist eine Einhaltung aller geforderter Zeitbedingungen gewährleistet, wobei sich eines auf das andere stützt. Eine hohe Verarbeitungsgeschwindigkeit ist ohne zeitliche Vorhersagbarkeit für ein Echtzeitsystem bedeutunglos, da eine hohe Verarbeitungsgeschwindigkeit nicht automatisch eine Einhaltung aller Deadlines nach sich zieht.

Trotz aller Vorkehrungen und Vorsichtsmassnahmen beim Design des Echtzeitsystems kann es passieren, dass eine Deadline, eine Echtzeitbedingung, nicht eingehalten wird. Aus diesem Grund unterscheidet man nach Strenge und Qualität der einzuhaltenden Zeitbedingung in harte und weiche Echtzeit. [WB05] [Sch05]

Bei **harter Echtzeit** ist es zwingend erforderlich, die gesetzte Deadline einzuhalten, da ein Überschreiten katastrophale Folgen für das System und die Umgebung hat. Eine harte Echtzeitbedingung ist somit eine Bedingung, die in allen Fällen vom System erfüllt werden muss. Ein Airbag beispielsweise muss innerhalb der definierten Zeit auslösen, da ein Überschreiten der Deadline eine stärkere Verletzung des Fahrgastes nach sich ziehen kann.

Anders als bei harter Echtzeit wirkt sich die Einhaltung von Deadlines nicht katastrophal auf **weiche Echtzeitsysteme** aus, sie würden weiterhin korrekt funktionieren. Eine gelegentliche Verletzung ist tolerierbar. Beispielsweise ist eine Bildstörung in einem Film, bedingt durch eine nicht eingehaltene Deadline zwar störend, gefährdet aber nicht die Umwelt.

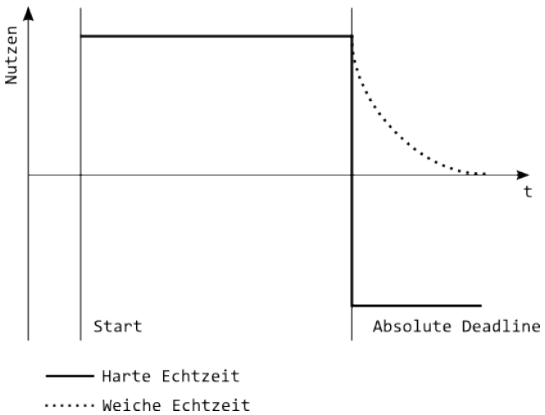

Abbildung 3: Harte und weiche Echtzeit

Neben der Rechtzeitigkeit muss ein Echtzeitsystem auch die Anforderungen nach **Gleichzeitigkeit** und **Verfügbarkeit** erfüllen. Da in einem Echtzeitsystem in der Regel mehrere Ereignisse gleichzeitig eintreffen und behandelt werden müssen, muss auch in diesem Fall die Rechzeitigkeit einer jeder Deadline gewährleistet sein. Diese Anforderung kann durch die Gleichzeitigkeit erfüllt werden. Zur Gewährleistung der Gleichzeitigkeit gibts es mehrere Möglichkeiten:

- Parallelverarbeitung in einem Mehrprozessorsystem
- Quasi-parallele Verarbeitung in einem Mehrprozessorsystem
- Quasi-parallele Verarbeitung in einem Einprozessorsystem

Während die erste Möglichkeit eine wirkliche Parallelverarbeitung darstellt, müssen sich bei den anderen beiden Möglichkeiten die anstehenden Aufgaben den oder die Prozessoren teilen. Diese Aufgabe übernimmt eine Hard- oder Softwarekomponente, der sogenannte Echtzeitscheduler. Dieser verteilt die Aufgaben so, dass nach Möglichkeit alle Deadlines eingehalten werden.

Um eine garantierte Einhaltung der Zeitbedingungen zu gewährleisten, muss das Echtzeitsystem unterbrechungsfrei arbeiten. Das kann, je nach Einsatzort, eine Betriebszeit von ein paar Minuten bis zu 24h pro Tag sein. Eine Unterbrechung des Betriebs durch z.B. Reorganisation des Systems (Speicher) ist nicht zulässig, es müssen entsprechende Vorkehrungen getroffen werden, wie die Verwendung von reorganisationsfreien Algorithmen oder die Unterteilung der Reorganisation in kleine Schritte.

3 Echtzeitprogrammierung

Um das Zeitverhalten in Echtzeitsystemen zu realisieren, existieren 2 Ansätze zur Programmierung, synchrone und asynchrone Programmierung.

3.1 Synchrone Programmierung

Synchrone Programmierung, auch zeitgesteuerte Programmierung genannt, legt das zeitliche Verhalten periodisch auszuführender Aktionen bereits vor ihrer Ausführung fest. Eine periodische Unterbrechnung durch einen Timer sorgt dafür, dass verschiedene Teilprogramme nacheinander ausgeführt werden. Solch ein System lässt sich vollständig vor seiner Ausführung planen, die Einhaltung aller Deadlines kann somit gewährleistet werden. Jedoch müssen Ereignisse zyklisch von den Teilprogrammen selbst abgefragt werden. Solch ein Polling birgt die Gefahr, Ereignisse oder die Reihenfolge des Auftretens der Ereignisse zu verlieren. Es sind somit entsprechende Gegenmaßnahmen zu treffen, wie beispielsweise Pufferung der Ereignisse. Ein bekannter Vertreter ist das Time Triggered Protocol (TTP) [Kop02] sowie FlexRay.

Synchrone Programmierung lässt sich also wie folgt charakterisieren [WB05]:

- festes, vorhersagbares Zeitverhalten
- einfache Analyse und Test des Systems
- hervorragend bei zyklischen Abläufen anwendbar
- richtige Planung kann Rechtzeitigkeit und Gleichzeitigkeit garantieren
- besonders für sicherheitsrelevante Aufgaben geeignet
- geringe Flexibilität gegenüber Änderungen der Aufgabenstellung
- Reaktion auf aperiodische Ereignisse ist nicht vorgesehen

3.2 Asynchrone Programmierung

Bei der asynchronen Programmierung, auch ereignisgesteuerte Programmierung genannt, lösen Ereignisse eine Unterbrechung, einen Interrupt, im System aus. Dadurch kann auf asynchrone Ereignisse, also Ereignisse, deren Auftreten nicht vorherbestimmbar ist, ohne große zeitliche Verzögerung reagiert werden. Diese Struktur ist deutlich flexibler als die der synchronen Programmierung, reagiert aber anfällig auf viele gleichzeitige Events (event showers). Zudem ist der Programmablauf schwer plan- und nachvollziehbar.

Asynchrone Programmierung lässt sich durch Echtzeitscheduling realisieren. Es existieren verschiedene Strategien zum Scheduling, u.a. Rate Monotonic Scheduling oder Fixed Priority Preemeptive Scheduling. Zwei Strategien werden im Abschnitt Echtzeitbetriebssysteme näher betrachtet.

Die Charakterisierung der asynchronen Programmierung lautet demnach wie folgt [WB05]:

- flexibler Programmablauf und flexible Programmstruktur
- Reaktion auf periodische sowie aperiodische Ereignisse
- Rechtzeitigkeit kann nicht in jedem Fall im Voraus garantiert werden
- je niederer die Priorität einer Aufgabe, desto größer werden die möglichen Zeitschwankungen
- Analyse und Test des Systems schwieriger als bei synchroner Programmierung

4 RTOS - Real-Time Operating System

Echtzeitprogrammierung lässt sich auf verschiedene Arten implementieren. Ein Ansatz ist die direkte Programmierung eines interruptgesteuerten Systems ohne auf ein RTOS zurückzugreifen [Wir01], oder die Verwendung eines Echtzeitbetriebssystems (RTOS). Welche Methode zu bevorzugen ist, hängt stark vom Echtzeitsystem selbst ab, u.a. von der Komplexität des Systems, der benötigten Verarbeitungsgeschwindigkeit und von vielem mehr.

Wann ist ein Betriebssystem ein Echtzeitbetriebssystem? Im Posix-Standard **1003.1b** ist folgende Definition zu finden: “Realtime in operating systems: the ability of the operating system to provide a required level of service in a bounded response time.” Das bedeutet, dass ein Betriebssystem genau dann

ein Echtzeitbetriebsssytem, kurz RTOS (Real-Time Operating System), ist, wenn es einen gewissen Umfang an Services bereitstellt und innerhalb einer gewissen Zeit auf Ereignisse reagiert, also Deadlines einhält.

Ein RTOS muss eine Unterstützung für Scheduling, Ressourcenmanagement, Intertaskkommunikation und Synchronisation, exaktes Timing und eine Kapselung der Hardware (HAL[1]) unterstützen. [SR04] [Kal08]

Der wohl wichtigste Teil eines RTOS ist der Scheduler, der das Taskmanagement übernimmt und die einzelnen Tasks, kleine Teilprogramme, die einen Aspekt des Systems abbilden, verwaltet und ihnen zu gegebener Zeit CPU-Zeit zuweist. Ein Scheduler sollte verschiedene Eigenschaften erfüllen, u.a. Fairness (jeder Task bekommt CPU-Zeit), Effizienz (max. Auslastung der CPU), Durchsatz (möglichst viele Tasks in einer bestimmten Zeit bearbeiten), Terminerfüllung (einhalten von Deadlines).

Diese Zuweisung der CPU-Zeit an die Tasks geschieht nach verschiedenen Scheduling-Algorithmen [Wal08]:

Kooperatives Scheduling Bei diesem Scheduling-Verfahren wird dem nächsten als bereit markierten Task die CPU zugewiesen. Dieser hält so lange die CPU, bis er sie von sich aus wieder freigibt.

Round Robin Round Robin beschreibt ein Scheduling nach dem Zeitschlitzverfahren. Alle bereiten Tasks werden in einer Warteschlange (FiFo[2]) verwaltet. Der vorderste Task, also derjenige, der sich bereits am längsten in der Warteschlange befindet, bekommt einen Zeitschlitz lang CPU-Zeit und Zugriff auf die Ressourcen. Ist der Zeitschlitz vorüber oder gibt der Task von sich aus die CPU wieder frei, wird der Task hinten in der Warteschlange eingereiht und der Nächste bekommt die CPU zugeteilt.

Präemptives Scheduling Ein höher priorisierter Task kann einen Task mit kleinerer Priorität verdrängen, wenn er in den Status bereit wechselt, also beispielsweise ein Event empfängt oder eine Ressource freigegeben wird, die von diesem Task angefordert war. Der höherpriore Task hält die CPU so lange, bis ein noch höher priorisierter Task ihn verdrängt oder er die CPU abgibt, weil er auf eine Ressource wartet oder mit seiner Arbeit fertig ist.

Ein weiterer Kernel-Service neben dem Scheduler ist die Intertaskkommunikation und Synchronisation. Er bietet die Möglichkeit, Informationen zwischen Tasks über Mailboxes oder Events auszutauschen und, falls mehrere Tasks kooperativ auf eine Ressource zugreifen müssen, diese über Semaphore und Mutexe zu schützen, sodass immer nur ein Task zu einer Zeit auf diese Ressourcen zugreifen kann.

Zusätzlich bieten die meisten Echtzeitbetriebssysteme Zeitmanagementfunktionen wie Delay-Timer und timeouts.

Einige Echtzeitbetriebssysteme bieten auch ein erweitertes Speichermanagement in Form statisch angelegter Speicherbereiche, da dynamische Allokierung zu einer Fragmentierung des Speichers führt.

5 Echtzeit mit Keil RTOS RTX

Keil bietet, neben vielen anderen Anbietern, ein Echtzeitbetriebssystem an. Das RTOS aus der RL-ARM[3] nennt sich RL-RTX oder einfach nur RTX[4] und wurde für Microcontroller der ARM7-, ARM9- und Cortex-M3-Serie entwickelt. Nachfolgend soll gezeigt werden, ob und inwieweit Echtzeitprogrammierung damit möglich ist und wie das Echtzeitverhalten des RTX ist. [Kei08] Alle Angaben beziehen sich auf die Verwendung eines ARM7 Controllers.

Um das RTOS RTX anwenden zu können, müssen verschiedene Parameter in einer Konfigurationsdatei eingestellt werden, darunter die maximale Anzahl der gleichzeitig laufenden Tasks, die Stackgröße der Tasks, der Timer, der für den System-Tick verwendet werden soll, den System-Tick Intervall,

1 Hardware Abstraction Layer
2 First in First out
3 Real-Time Library for ARM
4 Real-Time eXecutive

den Scheduling-Algorithmus und einiges mehr, auf das hier nicht eingegangen werden kann. Gespeichert sind diese Daten in der Datei `RTX_Config.c`, die für jedes Projekt neu eingestellt werden sollte.

Um die max. Stackgröße der Tasks angeben zu können, muss diese für jeden Task berechnet oder experimentell herausgefunden werden. Ein zu niedriger Wert sorgt für Stack-Overflows und undefiniertem Verhalten, ein zu hoher Wert verschwendet wertvolle Ressourcen in Form von RAM.

Die Intervallzeit für den System-Tick, also der Zeit, nach der bei Round Robin der aktuelle Task die CPU verliert und der Scheduler die CPU dem nächsten Task zuweist, sollte nicht zu klein eingestellt werden, da die CPU nur damit beschäftigt ist, Takswechsel durchzuführen, aber auch nicht zu groß, damit kein Task verhungert oder das Echtzeitverhalten nicht mehr garantiert werden kann.

Das RTX kann prinzipiell mit einer unbegrenzten Menge an Tasks arbeiten. Die maximale Anzahl an Tasks wird nur durch den Speicher, vor allem dem RAM (jeder Task hat einen eigenen Stack und belegt zusätzlich 52 Bytes an Verwaltungsdaten im RAM), des verwendeten Controllers begrenzt. Allerdings dürfen maximal 250 Tasks aktiv sein, also zur Ausführung durch die CPU bereit.

5.1 Echtzeitprogrammierung mit RTX

Der **Scheduler** des RTX unterstützt zwei Scheduling-Strategien, kooperatives Scheduling und Round Robin, wobei Round Robin prioritätsbasiert präemtpiv ist.

Beim kooperativen Scheduling findet nur ein Taskwechsel statt, wenn der gerade aktive Task die CPU mit dem Befehl `os_tsk_pass()` komplett abgibt oder mittels `os_dly_wait()` einen Timer aufzieht und auf das Timeout dieses Timers wartet.

Round Robin Multitasking funktioniert, wie bereit im Abschnitt 4 beschrieben, nach dem Zeitschlitzverfahren. Jeder Task bekommt einen Zeitschlitz lang die CPU zugeteilt und muss diese wieder abgeben, wenn der Zeitschlitz zu Ende ist. Beim Wechsel der Tasks wird der Kontext der aktuellen Task gesichert (u.a. Programmcounter, Stack), der Scheduler wählt den nächsten aktiven Task aus, stellt dessen Kontext wieder her und gibt die Kontrolle an den Task ab. Jedem Task kann beim Anlegen eine Priorität mitgegeben werden. Wird eine Task mit einer höheren Priorität als die der gerade aktuell Laufenden, aktiv, dann verdrängt diese neue Task die bisherige. Gründe für ein aktiv werden einer Task sind folgende:

- ablaufen eines Delays eines Tasks mit höherer Priorität
- ein Event wird durch die aktuelle Task für eine höher priorisierte Task oder durch einen Interrupt generiert
- ein Mutex wird freigegeben, auf den eine höher priorisierte Task wartet
- eine Nachricht, auf die eine höher priorisierte Task wartet, wird durch die aktuelle Task oder einen Interrupt an eine Mailbox übergeben
- eine Mailbox ist voll und eine höher priorisierte Task möchte eine Nachricht senden, dann wird diese aktiv, wenn eine andere Task oder in einem Interrupt eine Nachricht aus der Mailbox gelesen wird
- die Priorität des aktuellen Tasks wird verringert

Von der Wahl des Schedulingverfahren hängt ab, ob das Echtzeitsystem als synchrones oder asynchrones System definiert wird. Aber obwohl kooperatives Scheduling am ehesten ein synchrones System definiert, ist das mit Vorsicht zu betrachten. Wie im Abschnitt 3.1 gezeigt, werden die Tasks nacheinander mit bestimmten Zeitschlitzen ausgeführt, was aber eher durch Round Robin erreicht wird. Zusätzlich wird bei kooperativem Scheduling ein Taskswitch nur ausgeführt, wenn der aktive Task selbst die CPU abgibt. Zudem ist es durch die Funktion `os_dly_wait()` in einem gewissen Bereich auch möglich, auf asynchrone Events zu warten, wobei synchrone Systeme laut Definition nicht in der Lage sind, asynchrone Events zu bearbeiten.

Präemptives Round Robin trifft aber die Definition für asynchrone Systeme. Damit ist es möglich, auf asynchrone Events zu reagieren, prioritätsbedingte Taskswitches durchzuführen und auch die Erweiterbarkeit ist problemlos möglich, indem einfach eine weitere Task erstellt wird (hier bietet RTX sogar das mehrfache Erstellen einer Task auf Basis einer einzigen Funktion).

5.2 Zusätzliche RTOS Services

Neben dem Scheduler, dem eigentlichen Kern-Service eines RTOS, sind im Keil RTX noch weitere Services integriert, die hier kurz vorgestellt werden.

Zur Interprozesskommunikation, also zum Datenaustausch zwischen einzelnen Tasks, sind Mailboxen, Events, Mutexe und Semaphoren vorhanden.

Events sind einfache Signale an Tasks, die keine weitere Information transportieren, während Mailboxen komplexe Nachrichten übertragen können.

Zur Synchronisation von Ressourcen zwischen Threads existieren Semaphoren und Mutexe. Beide sind Softwareobjekte, über die der Zugriff auf Ressourcen geregelt werden können. Ein Task kann ein Lock auf eine Ressource anlegen und verhindert damit, dass andere Tasks auf diese Ressource zugreifen können, solange dieser Lock besteht. Über Mutexe[1] kann man kritische Bereiche definieren, innerhalb derer z.B. Funktionen aufgerufen werden, die nicht mehrfach aufgerufen werden dürfen (nicht reentrant).

Timer ermöglichen es, einen Task in den Zustand *wartend* zu versetzen,d.h. der Task wird für eine festgelegte Zeit die CPU entzogen. Während dieser Zeit können andere Tasks abgearbeitet werden, bis der Timer abläuft und die ursprüngliche Task wieder aktiviert wird.

Zusätzlich gibt es noch Speicherallokierungsmethoden. Dabei werden statische Speicherbereiche angelegt, auf die zur Laufzeit über spezielle Reservierungsmethoden zugegriffen werden kann. Da die Speicherbereiche in vorher definierte Blöcke gewisser Größe erstellt werden und nur Blöcke dieser Größe reserviert werden können, ist eine Speicherfragmentierung ausgeschlossen.

1 mutual exclusion lock

5.3 Echtzeitverhalten

Um das Echtzeitverhalten eines RTOS bewerten zu können, sind zwei wichtige Größen zu betrachten. Eine Größe ist die **Task Switch Time**, die die Zeit definiert, die benötigt wird, um den Scheduler aufzurufen, der den Kontext des ursprünglichen Tasks sichert, den nächsten Task auswählt und dessen Kontext zurückschreibt. Je länger ein Taskswitch dauert, desto weniger CPU-Zeit haben die Tasks zur Verfügung, desto ineffizienter wird die CPU ausgenutzt. Optimal wäre eine Taskswitchzeit von einem Taktzyklus des Controllers, was aber höchstens durch Hardwareunterstützung in diese Regionen kommt.

Die zweite Größe ist die **Interrupt-Latenzzeit**. Diese Zeit gibt an, wie lange es ab dem Zeitpunkt der Interruptgenerierung dauert, bis dieser abgearbeitet wird. Eine geringe Interruptlatenz ist wichtig, wenn schnell auf ein Ereignis reagiert werden muss. Architekturbedingt bringt der ARM7-Kern schon eine Interruptlatenz von 48 Taktzyklen mit.

Tabelle 1 zeigt einen Vergleich der Daten des Keil RTOS [Kei08] mit dem Segger embOS [Seg08] sowie dem CMX Systems CMX-RTX [Sys08].

RTOS	*Task switch*	*IRQ Latenz*
RTX	396	66
embOS	283	n.d.
CMX-RTX	186	84

Tabelle 1: Zeitvergleiche in Taktzyklen

Obwohl deutliche Differenzen zu sehen sind, liegen die Taskswitchzeiten bei den angegebenen Echtzeitbetriebssystemen unter $6\mu s$ bei $72MHz$ Taktfrequenz. Die Interruptlatenzzeiten liegen nochmal deutlich darunter. Ob diese Zeiten für ein Echtzeitsystem ausreichend sind, wird durch die minimale Reaktionszeit des Systems auf ein Event bestimmt und ist von System zu System unterschiedlich.

6 Zusammenfassung

Das Keil RTOS RTX bietet im Prinzip alles, was zur Echtzeitprogrammierung notwendig ist. Es ist keine direkte Unterstützung für synchrone Programmierung vorhanden, was aber aufgrund der Tatsache, dass solche Systeme in aller Regel ohne RTOS programmiert werden, kein Problem darstellt. Mit prioritätsbasiertem Round Robin Scheduling liefert es für die asynchrone Programmierung eine aktuelle Technik, die so in ähnlicher Form auch in aktuellen Desktop-Betriebssystemen eingesetzt wird.

Auch im Echtzeitverhalten bietet es durchaus gute Werte und braucht sich vor anderen Echtzeitbetriebssystemen nicht zu verstecken. Ob die gebotene Echtzeit jedoch für verschiedene Projekte ausreicht, muss von Fall zu Fall berechnet werden (beispielweise mit dem in [PS97] vorgestellten Verfahren zur Berechnung der WCET [1]). Lediglich die Unterstützung für andere Prozessoren außer ARM7/9 und Cortex-M3 ist nicht vorhanden. Hiermit ist leider kein einfacher Plattformwechsel möglich.

Literatur

[HGCD97] Halang, Wolfgang A. ; Gumzej, Roman ; Colnaric, Mathaz ; Drzuovec, Marjan: Measuring the Performance of Real-Time Systems / FernUniversität Hagen, University of Maribor. 1997. – Forschungsbericht

[Kal08] Kalinsky, David: Introduction to Real-Time Operating Systems. In: *Embedded World Conference 2008*, 2008

[Kei08] Keil: *RL-ARM User's Guide*. http://www.keil.com/support/man/docs/rlarm/rlarm_ar_artxarm.htm. Version: 2008

[Kop02] Kopetz, Hermann: Time Triggered Architecture. In: *ERCIM NEWS* (2002), Jan., Nr. 52, S. 24–25

[PS97] Puschner, Peter P. ; Schedl, Anton V.: Computing Maximum Task Execution Times - A Graph-Based Approach / Technische Universität Wien. 1997. – Forschungsbericht

[Sch05] Scholz, Peter: *Softwareentwicklung eingebetteter Systeme*. Springer, 2005. – ISBN 3–540–23405–5

[Seg08] Segger: *embOS General Info*. http://www.segger.com/embos_general.html. Version: 2008

[SR04] Stankovic, John A. ; Rajkumar, R.: Real-Time Operating Systems / University of Virginia, Carnegie Mellon University. 2004. – Forschungsbericht

[Sys08] Systems, CMX: *CMX-RTX System Specification*. http://www.cmx.com/trget24.htm. Version: 2008

[Wal08] Walls, Collin: An Introduction to Real-Time Operating Systems. In: *Embedded World Conference 2008*, 2008

[WB05] Wörn, Heinz ; Brinkschulte, Uwe: *Echtzeitsysteme*. Springer, 2005. – ISBN 3–540–20588–8, 978–3–540–20588–3

[Wir01] Wirth, Niklaus: Embedded Systems and Real-Time Programming. In: *Embedded Software, First International Workshop*, EMSOFT, 2001

1 Worst Case Execution Time