Evaluation und Erweiterung von Echtzeit-betriebssystemen

In eingebetteten Lösungen steigt der Anspruch an die Funktionalität permanent an. Dies führt zu zusätzlichen Ansprüchen an die Softwarekomponente des Systems. Insbesondere die Gewährleistung der Echtzeitfähigkeit, Portabilität, Wartbarkeit, Wiederverwertung und die Erweiterbarkeit der Software stellen eine Herausforderung für den Programmierer dar, wenn es um die Entwicklung und Umsetzung der Architektur geht.

Die Leistung des Systems alleine, kann keine definierte Antwortzeit garantieren.
Auf einer ARM Architektur basierende Controller bieten eine Menge an Features, die bei Einsatz eines Echtzeitbetriebssystems ohne große Performanceverluste garantierte Antwortzeiten ermöglichen [1]. Die Auswahl sowie die Verfügbarkeit solcher Rechentechnik und die Zahl der Einsatzgebiete in eingebetteten Systemen werden zusehends größer.

Als eine Middleware stellt ein Echtzeitbetriebssystem einen Kompromiss zwischen allen genannten Anforderungen dar. Durch die Bereitstellung von Schnittstellen und Mechanismen zur effizienten Ressourcenverwaltung, kann das Echtzeitbetriebssystem einen erheblichen Beitrag zur Minimierung der Reaktionszeit des Gesamtsystems beitragen.
Das Echtzeitbetriebssystem stellt einen Scheduler zur priorisierten Verwaltung von Tasks bereit. Durch Multitasking entstehende Kontextwechsel entstehen zeitlichen Verzögerungen zwischen den Ereignissen im System. Gleichzeitig wird jedoch gewährleistet, dass diese Unterbrechungen beim Scheduling angemessen berücksichtigt werden.

Ereignisse, wie externe Interrupts sowie systeminterne Vorgänge zur Wahrung der Datenintegrität, können diesen zeitlichen Verzug vergrößern. Dadurch wird ein Echtzeitbetriebssystem für den Einsatz in Echtzeitsystemen mit harten Echtzeitanforderungen oft untauglich.

Der Markt von Echtzeitbetriebssystemen wird von vielen kommerziellen Herstellern, als auch von Open Source Projekten bedient. Die angebotenen Echtzeitbetriebssysteme haben unterschiedliche Leistungsmerkmale. Die Angaben der Hersteller zu den Leistungsdaten der Systeme sind allerdings nicht ausreichend, um eine qualifizierte Auswahl eines Betriebssystems zu ermöglichen.

Es existieren mehrere Methoden zur Evaluierung des durch ein Echtzeitbetriebssystem entstehenden Overheads wie Rhealstone Benchmark [2], MiBench [3] oder Hartstone Benchmark [4]. Diese liefern jedoch nur ein Ergebnis in Form eines Durchschnitts. Solche gemittelten Ergebnisse sind wenig aussagekräftig und nicht verwertbar für die Planung konkreter Aufgaben [4]. Es wird ein Benchmark benötigt, der der aktuellen Anwendung angepasst ist.

Deshalb soll eine Methode für die Evaluierung von Echtzeitbetriebssystemen entwickelt werden, um den Einfluss der externen Ereignisse auf die maximale Ausführungszeit des Gesamtsystems zu ermitteln. Ebenso soll es diese Methode erlauben, Anwendungen und Testfälle unabhängig vom Echtzeitbetriebssystem zu gestalten. Ein Open Source Echtzeitbetriebssystem soll mit dieser Methode analysiert werden und dessen Performance, in Bezug auf die Einhaltung harter Echtzeitbedingungen verbessert werden. Es wird erwartet, dass die Verwaltung der Prioritäten Hauptursache für den maximalen Verzug sind.

Die in dieser Arbeit verwendete Anwendung, dient der Steuerung eines Gelenks in einem Industriemanipulator.

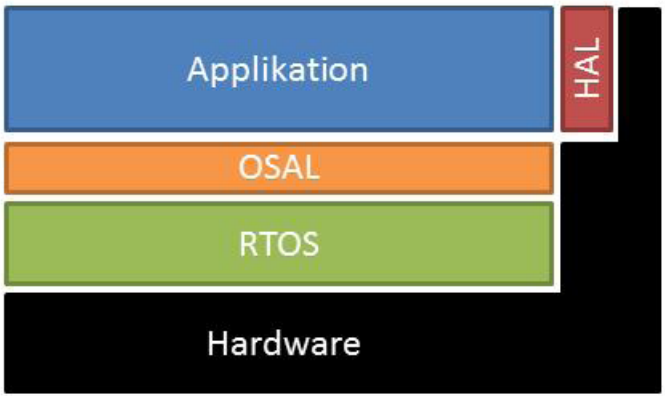

Abbildung 1: Aufbau der Applikation

Dieses Gelenk enthält zwei Motoren. Versagt das Gelenk auf Grund der Überschreitung der zugelassenen Masse, so muss das System in der Lage sein, mit minimalem Verzug zu reagieren, um den entstehenden Schaden zu begrenzen. Ereignisse wie eine A/D Wandlung, Überschreitung eines Stromwerts, oder Erreichen

der Endposition des Gelenkes, zusammen mit dem ohnehin ständig mitlaufendem Systemtaktgeber des Prozessors, können bei gleichzeitigem Auftreten zu starken Verzögerung in der Ausführung der verschiedenen Tasks führen.

Das Evaluierungssystem besteht aus zwei Hardwarekomponenten. Die erste Komponente stellt das Zielsystem dar. Die Softwareapplikation des Zielsystems benutzt ein Echtzeitbetriebssystem als Middleware, das zur Verteilung von Ressourcen zwischen den einzelnen Aufgaben dient. Des Weiteren kann das Zielsystem über 16 Kanäle Unterbrechungen empfangen. Jede Unterbrechung leitet eine entsprechende Reaktion ein und kann unterschiedlich priorisiert werden.

Zur Gewährleistung der Unabhängigkeit der Softwareapplikation vom verwendeten Echtzeitbetriebssystem, wird eine einheitliche Betriebssystem Abstraktionsschicht (OSAL) entwickelt. Der OSAL bietet allgemeine Basisfunktionen eines Echtzeitbetriebssystems zur Erzeugung von Tasks, Taskmanagement, sowie Interprozesskommunikation. Der OSAL ermöglicht es, ohne Veränderungen der Applikation, zwischen Echtzeitbetriebssystemen zu wechseln. Abbildung 1 zeigt den Aufbau der Firmware des Zielsystems. Die Applikation hat keinen direkten Zugriff auf die Hardware. Der Hardware Abstraction Layer (HAL) abstrahiert die Hardware von der Applikation und steigert die Portabilität. Der Operating System Abstraction Layer (OSAL) abstrahiert das Echtzeitbetriebssystem (RTOS) von der Applikation und erlaubt den schnellen Tausch der Echtzeitbetriebssysteme. Des Weiteren kann der OSAL zur Entwicklung neuer Applikationen verwendet werden. Die angebotenen Basisifunktionen, tragen zur Verbesserung der Portabilität der Anwendung bei.

Abbildung 2 veranschaulicht den Gesamtaufbau des Messsystems. Die zweite Komponente ist ein Stimulator, der in der Lage ist, gezielte, sowie zufällige Ereignisse für das Zielsystem zu erzeugen. Dank des Stimulators können unterschiedliche Szenarien für das Zielsystem simuliert werden. Durch unterschiedliche Stimulationen besteht die Möglichkeit, die Reaktionszeit des Zielsystems zu prüfen und zu beurteilen.

Zielsystem und Stimulator sind über 16 Externe Interrupt Kanäle des Zielsystems (EXT) miteinander gekoppelt und synchronisiert. Die Synchronisation erlaubt es, die Reaktion des Echtzeitbetriebssystems von der Erzeugung bis zum Erkennen der Unterbrechung und Einleitung einer Reaktion zu registrieren und zu messen. Dank dem mit der Coresign-Einheit des ARM Prozessors verbundenen Debug Access Port (DAP) kann, mit Hilfe des Serial Wire Debug Protokolls (SWD) , der innere Zustand des Zielsystems sowie Simulators aufgenommen und in Echtzeit dargestellt werden. Mit diesen Mitteln kann die Messung anhand eines Traces nachvollzogen werden.

Die Messungen werden zeigen, ob und inwieweit sich die Ergebnisse des anwendungsbezogenen Benchmarks von denen des Rhealstone Benchmarks [2] unterscheiden und ob im Fall des untersuchten Open Source Betriebssystems tatsächlich, wie erwartet, die Verwaltung der Prioritäten Hauptursache für den maximalen Verzug ist.

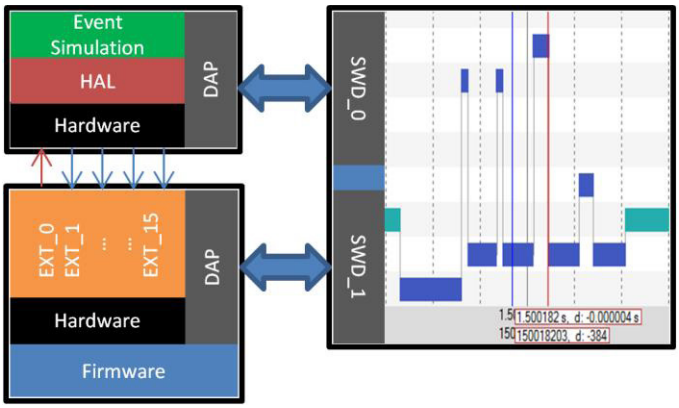

Abbildung 2: Aufbau des Messsystems

[1] Yiu, J. (2011). The Definitive Guide to the ARM Cortex-M0. Elsevier Science
[2] Dr. Dobb's Journal: Rhealstone: a Real-Time Benchmarking Proposal http://www.drdobbs.com/rhealstone-a-real-time-benchmarking-prop/184408081 Abruf: 03.12.2015
[3] MiBench. Embedded Benchmark http://wwweb.eecs.umich.edu/mibench/ Abruf: 03.12.2015
[4] Gumzej, R. (2010). Real-time Systems' Quality of Service: Introducing Quality of Service. London: Springer.

Verfasser: Edgar Burghardt