Embedded Realtime OS FreeRTOS auf STM32F4

Michael Ebert
Ad-hoc Networks GmbH
ebert@ad-hoc.network

Christoph Bläßer
Bundesamt für Sicherheit in der Informationstechnik
christoph.blaesser@gmx.de

Stichwörter
RTOS, ARM , STM32, Real Time.

KURZFASSUNG
Im Rahmen des vorliegenden Papers wird das Echtzeitbetriebssystem FreeRtos vorgestellt. Hierzu werden zu Beginn die allgemeinen Vorgaben für Echtzeitbetriebssysteme beschrieben. Im Verlauf des Textes wird an ausgewählten Beispielen dargestellt, wie FreeRos diese Anforderungen berücksichtigt und durch geeignete Programmfunktionen umsetzt.

1. ECHTZEITBETRIEBSSYSTEME

1.1 Vorteile eines Echtzeitbetriebssysteme

Mit der steigenden Leistungsfähigkeit von modernen , steigen auch die Anfordernungen an die Software die auf diese Systeme aufsetzt. Viele dieser Anwendungen verlangen trotz ihrer Komplexität, dass Teile des Programmablauf in bestimmten zeitlichen Grenzen ausgeführt wird und somit vorhersehbar und deterministisch sind. Es gibt prinzipiell drei verschiedene Modelle wie der Programmablauf einer embedded Anwendung gestaltet sein kann, siehe Abbildung 1. Anwendungen können in einer einzigen Schleife laufen oder aber in event-gesteuerten nebenläufigen eigenständigen Programmabschnitten (Thread oder Task) ausgeführt werden. Diese Nebenläufigkeit ist nur durch einen RTOS Kernel (Scheduler) möglich. Für viele kleine Anwendungen kann die Nutzung einer einzigen Schleife durchaus sinnvoll sein, sollten beispielsweise die Ressourcen so knapp sein, dass ein Overhead an Funktionalität ausgeschlossen werden muss. Ein großer Nachteil der „einschleifen Variante“ ist die permanente Nutzung des Prozessors. Besonders bei neuen akkubetrieben Geräten z.B. IoT Device wird sehr genau auf die Energieaufnahme geachtet. Ein RTOS bietet hingegen Funktionen mit denen sehr leicht ermittelt werden kann, ob ein Gerät in einen Schlafmodus wechseln kann, dies wird in Abschnitt 4.2 an Beispielen von FreeRtos und einem ARM uProzessor demonstriert.
!TODO!
Neben der Echtzeitfähig gibt es aber noch viele weitere Vorzüge für den Einsatz eines Echtzeitbetriebssystems. Durch das Herunterbrechen der Anwendung in Task entstehen viele kleine Module, die jeweils eine kleine Teilaufgabe des Gesamtsystems übernehmen. Die Anwendungen werden so skalierbar und erweiterbar. Die Entwicklungsarbeit lässt sich leicht auf Teams verteilen, die unabhängig ihre Module (Tasks) entwickeln und testen. Dies ermöglicht auch den Einsatz von agilen Entwicklungsmethoden wie Scrum in der embedded Enticklung. Abschnitt 5 Echtzeitanalyse detailliert erläutert.

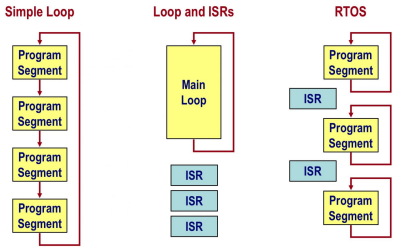

Abbildung 1. Übersicht Programmabläufe
Quelle: `http://www.embedded.com/`

RTOS aufsetzt, erfüllt somit nicht sofort Harten oder Weichen Echtzeitkriterien. Diese Kriterien zu erfüllen liegt beim Entwickler selbst, das Echtzeit Betriebssystem stellt nur die nötigen Werkzeuge zur Verfügung. Wie geprüft werden kann ob Echtzeitkriterien erfüllt werden, wird in Kapitel X näher erleutert.

2. GRUNDLAGEN

2.1 Allgemeine Anforderungen an Betriebssysteme

Betriebssysteme verwalten den Hardwarezugriff und stellen sicher, dass eingesetzte Software die benötigte Rechenzeit zur Verfügung gestellt bekommt. Gleichzeitig regeln Sie den Hardwarezugriff und organiseren den konkurierenden Zugriff, beispielsweise auf Netzwerkkarten und Festplatten. Sie stellen Funktionen für die Interprozesskommunikation bereit udn übernehmen grundlegende Aufgaben wie die Organisation von Arbeitsspeicher.

2.2 Anforderungen an Echtzeitbetriebssysteme

Echtzeitbetriebssysteme kommen zum Einsatz wenn neben den oben genannten Anforderungen an ein normales Betriebssystem weitere Anforderungen gestellt werden, die ein normales Betriebssystem nicht berücksichtigt. Dies können beispielsweise garantiert berechenbare Reaktionszeiten sein wie sie in der Fabrikation oder im Automobilbereich gefordert werden oder geringe Leistungsaufnahmen wie bei Komponenten des Internet of Things (IoT). Insgesamt wird zwischen Harten und Weichen Echtzeitkriterien unterschieden. Diese Gliedern sich wie folgt:
- Antwortzeiten
- Start-zu-Start Jitter von Tasks und Software-Komponenten
- Mehrfachaktivierung von Tasks
- Ausführungsreihenfolge von Software-Komponenten
- Latenzen von Wirkketten (Event Chains)
- Verlust von Daten
- Alter der Daten
- Datenkonsistenz
- Wiederverwendung von Daten(gewollt oder unbeabsichtigt)
- Verlust von Interrupts

- Blockierung von Interrupts

2.3 Sonstige Eigenschaften von Echtzeitbetriebssystemen

Aufgrund der Eingangs geschilderten Einsatzbereiche ist leich zu erkennen, dass Echzeitbetriebssysteme häufigin Umgebungen zum Einsatz kommen, in denen besondere Anforderungen an die Hardware gestellt werden. Häufig verfügt die Hardware nur über begrenzte Speicherkapazitäten, über geringe Wärmeableitfähigkeiten und damit geringe Rechenleistung. Die zur Verfügung stehende Energie muss bei der Entwicklung ebenfalls berücksichtigt werden. Vor diesem Hintergrund benötigen Echtzeitbetriebssysteme nur wenig Speicherplatz und implementieren Funktionen um den Prozessor und die angeschlossene Peripherie nur kurzzeitig zu belasten und in der restlichen Zeit in den Ruhezustand zu verseten.

3. FREERTOS

3.1 Geschichte

3.2 Zielsysteme

3.3 Fokus bei der Entwicklung

3.4 Ordner und Dateistrukturen

3.5 Scheduling

3.6 Memory Management

3.7 Interprozess Kommunikation

3.8 Software Timer

3.9 Interrupt Handling

3.10 Sonstige Maßnahmen zur Sicherstellung der Echtzeitfunktionalität

4. ENTWICKLUNGSUMGEBUNG

4.1 Eingesetzte Komponenten

4.2 Low Power Modes auf STM32F4 (ARM Cortex M3)

4.3 Installtion

5. ECHTZEITANALYSE

6. DEBUGGING VON ECHTZEITSYSTEMEN

7. ECHZEIT PATTERN

8. CODING HINDERNISSE

9. ZUSAMMENFASSUNG