Konzipierung und Realisierung eines asymmetrischen Multiprozessorsystems zur Erprobung der Regelung von bürstenlosen Gleichstrommotoren mittels aus einem MATLAB-Simulink-Modell generiertem Code unter Verwendung des Echtzeitbetriebssystems FreeRTOS und Datenaustausch mit dem übergeordneten Linux-System

Kevin Keppler*, **Reinhard Keller**, **Walter Lindermeir**

Fakultät Informationstechnik der Hochschule Esslingen – University of Applied Sciences

Sommersemester 2017

Durch die fortschreitende Entwicklung der Mikrokontroller und Mikroprozessoren sowie die damit verbundene Reduzierung der Kosten für Rechenleistung ist es mög-lich, rechen-intensive Algorithmen in eingebetteten Syste-men anzuwenden. Des Weiteren können durch die Verwendung eines Linux-Betriebssystems Visualisierungsaufgaben mit Hilfe von Anwen-dungsprogrammen einfacher gelöst werden.

Ein solches System wird bei der Firma ebm-papst benötigt, um die Robustheit von bürstenlosen Gleichstrommotoren zu erhö-hen. Hierfür werden verschiedene Verfahren mit komplexen Regelalgorithmen getestet. Das System ermöglicht es, den erhöhten Rechen-aufwand schnell und effizient zu kompensie-ren.

Aufgrund der sehr geringen Zykluszei-ten für die PWM-Ansteuerung der Antriebs-regelung bestehen hohe Anforderungen an das Echtzeitverhalten. Außerdem sollen die berechneten Ströme und Spannungen unter einem-Linux Betriebssystem visualisiert wer-den.

Aufbau des Systems

Verwendet wird ein System on Modul (SOM), welches aus einem Applikationsprozessor vom Typ NXP i.MX 6Quad besteht. Der Prozes-sor besitzt vier Cortex-A9-Prozessoren, wel-che mit einer Taktfrequenz von 1GHz betrie-ben werden. Außerdem verfügt das System über 2GB Arbeitsspeicher.

Zur Ansteuerung des Motors werden als Peripherie sieben AD-Wandler und drei dazu synchron arbeitende PWM-Ausgänge benötigt. Da das SOM nicht über solch eine Peripherie verfügt, wird ein weiterer Mikrokontroller be-nötigt, welcher über eine SPI- Schnittstelle mit dem SOM kommuniziert.

Um die geforderte Visualisierungsaufga-be und die geforderte Echtzeitverhalten zu gewährleisten, wird eine asymmetrische Mul-tiprozessorarchitektur verwendet, wel-che in der nachfolgenden Abbildung (Abbildung 1) dargestellt ist.

In der Abbildung ist ersichtlich, dass auf den Prozessorkernen CPU0, CPU2 und CPU3 eine Linux-Distribution und auf der CPU1 ein FreeRTOS-Betriebssystem ausgeführt wird. Dabei haben alle Prozessoren Zugriff auf den Arbeitsspeicher und die Peripherie. Im Arbeits-speicher ist für jedes Betriebssystem ein fes-ter Bereich definiert. Für den Datenaustausch zwischen den Betriebssystemen ist zudem ein gemeinsamer Speicherbereich *(Shared Memo-ry)* reserviert. Des Weiteren können jedem Pro-zessor Ressourcen zugeteiltet werden.

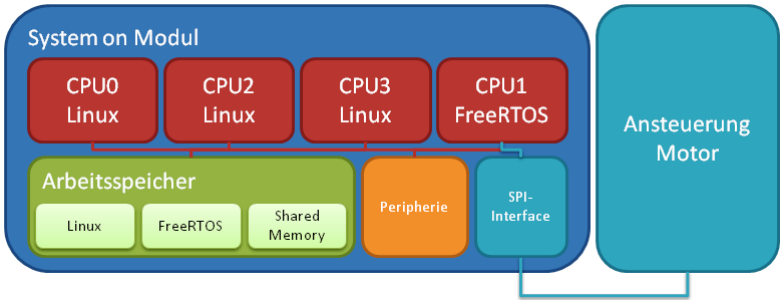

Abbildung 1: Asymmetrische Multiprozessorarchitektur

*Diese Arbeit wurde durchgeführt bei der Firma ebm-papst GmbH & Co. KG, Mulfingen

Funktionsweise

Das asymmetrische Multiprozesssing wird durch das *Remoteproc-Framework* ermöglicht, welches im Kernel, des auf dem Host Prozessors ausgeführten Betriebssystems (Linux Distribution), implementiert ist. Durch das Frame-work kann eine Firmware in den *Remote Prozessor* (FreeRTOS) geladen und dieser anschließend gestartet werden. Des Weiteren werden dem *Remote Prozessor* die zur Ausführung benötigten Ressourcen zugeordnet. Bei der in Abbildung 1 dargestellten Architektur ist dies für das SPI-Interface angewendet.

Zur Kommunikation zwischen den Prozessoren wird das *Remote Processor Messaging Framework* (RPMsg) verwendet, welches ebenfalls im Kernel der Host-Prozessoren implementiert ist. Für das Framework werden zwei Puffer im gemeinsamen Speicherbereich angelegt. Der eine Puffer ist für die Host-Prozessoren zum Lesen und für den *Remote Prozessor* zum Schreiben vorgesehen. Beim anderen Puffer ist dies genau umgekehrt. Über zwei Mailboxen signalisieren die Prozessoren, ob neue Daten zum Verarbeiten im Puffer liegen. [1]

Ablauf Steuerung

Der Ablauf der Steuerung ist in Abbildung 2 dargestellt. Hierbei ist der Peripherie-Controller als Master für die SPI-Kommunikation konfiguriert.

Zunächst setzt der Peripherie-Controller den neuen Wert für das PWM-Signal und wandelt synchron dazu die Strom- und Spannungswerte unter Verwendung des AD-Wandlers. Anschließend werden die neuen Werte an FreeRTOS gesendet, wo diese über zwei Buffer in einer ISR verarbeitet werden.

Nun können die neuen Werte berechnet und in das Datenregister ge-schrieben werden. Nachdem der Peri-pherie-Controller die Daten empfangen hat, kann der nächste Zyklus beginnen. Das FreeRTOS muss in der erneut aufgerufen ISR das Empfangsregister der SPI-Schnittstelle auslesen, um den FIFO-Speicher zu leeren. Hierfür ist Zeit bis zum erneuten Aufruf der ISR *ISR Senden*.

Erzeugung des Programmcodes

Der ausführbare C-Code zur Regelung wird aus Matlab-Simulink-Modellen mittels des Simulink-Coders generiert.

Die wichtigsten Funktionen des erzeugten Codes sind die Init- und die Step-Funktion. Die Initfunktion wird zum Initialisieren der Variablen einmalig über FreeRTOS aufgerufen und die Step-Funktion wird für jede Berechnung ausge-führt. Vor dem Ausführen der Step-Funktion müssen die zur Berechnung benötigten Daten in eine globale Struktur geschrieben werden. Nach der Berechnung können die neuen Werte aus der Struktur gelesen werden.

Falls die globale Struktur in den verschiedenen Modellen identisch ist, müssen für neue Algorithmen nur die Init- und Stepfunktion ausgetauscht werden.

Abbildung 2: Ablauf der Steuerung

[1] Texas Instruments (Hrsg.): PRU-ICSS Remoteproc and RPMsg. Online verfügbar unter http://processors.wiki.ti.com/index.php/PRU-ICSS_Remoteproc_and_RPMsg, Zugriff: 14.10.2016

Bildquellen:

- Abbildung 1,2: Eigene Abbildung