Thermische Optimierung des Prozess-Scheduling für Multicore-Prozessoren

Daniel Jäckle, Axel Sikora

***Zusammenfassung*—Die zunehmende Anzahl von Transistoren mit immer kleineren Strukturgrößen führt zu einer zunehmenden Leistungsaufnahme in modernen Prozessoren. Das gilt insbesondere für High-End Prozessoren, die mit einer hohen Taktfrequenz betrieben werden. Die aufgenommene Leistung wird in Wärme umgewandelt, die in einer Temperaturerhöhung der Prozessoren resultiert. Hohe Betriebstemperaturen verursachen u.a. eine verringerte Rechenleistung, eine kürzere Lebensdauer des Prozessors und höhere Leckströme. Aus diesen Gründen wird aktives, dynamisches thermisches Management immer wichtiger. Dieser Beitrag stellt eine Erweiterung zu dem Standard-Linux-Scheduler in der Kernel-Version 3.0 für eingebettete Systeme vor: einen PID-Regler, der unter Angabe einer Solltemperatur eine dynamische Frequenz- und Spannungsskalierung durchführt. Die Experimente auf dem Freescale i.MX6 Quadcore-Prozessor zeigen, dass der PID-Regler die Betriebstemperatur des Prozessors an die Solltemperatur regeln kann. Er ist die Grundlage für eine in Zukunft zu entwickelnde prädiktive Regelung.**

***Schlüsselwörter*—Dynamisches thermisches Management, PID-Regler, Multicore, Linux-Scheduler.**

I. EINLEITUNG

Dynamisches thermisches Management (DTM) wird eingesetzt, um den Prozessor vor Schaden durch hohe Betriebstemperaturen zu bewahren. Einerseits gilt es dabei, die durchschnittliche Temperatur gering zu halten und andererseits Spitzenwerte zu vermeiden - unter der Berücksichtigung einer maximalen Rechenleistung und eines kleinen zusätzlichen Rechenaufwands. Es ist damit ein wichtiger Bestandteil in modernen Prozessoren. In den letzten Jahren wurden verschiedene Hardware-basierte und Software-basierte Verfahren entwickelt. Software-basierte Verfahren verwenden Funktionen und Eigenschaften der Software. Insbesondere werden dafür Merkmale des Betriebssystems verwendet und mit Algorithmen zur Einbeziehung thermischer Eigenschaften erweitert, wie beispielsweise thermisches Thread- und Prozess-Scheduling [1]-[4] oder Prozessmigrationstechniken [5]-[7]. Thermisches Prozess-Scheduling beeinflusst die vom Standard-Scheduler vorgegebene Ausführungsreihenfolge der Prozesse, indem es thermische Eigenschaften der Prozesse in das Scheduling-Verfahren einbezieht. Bei Prozessmigrationstechniken werden die thermischen Eigenschaften verwendet, um die Erwärmung auf den verschiedenen Kernen in einem Multicore-Prozessor auszugleichen. Hardware-basierte Verfahren greifen auf Ressourcen bzw. Regelparameter des Prozessors zurück, wie beispielsweise dynamische Frequenz- und Spannungsskalierung (DVFS) [8]-[11], Clock Gating [12] oder Fetch Toggling [13]. DVFS führt eine Skalierung der Taktfrequenz und der Spannungsversorgung einzelner oder mehrerer Kerne in einem Multicore-Prozessor durch, während Clock Gating das Taktsignal für einzelne Komponenten des Prozessors aktiviert bzw. deaktiviert [12]. Die Verfahren im so genannten dynamischen thermischen Management können in regelungstechnische und nicht regelungstechnische Ansätze unterteilt werden [13]. Neben einer Übersicht von verschiedenen Techniken wird in [13] beschrieben, dass regelungstechnische Ansätze in DTM-Techniken

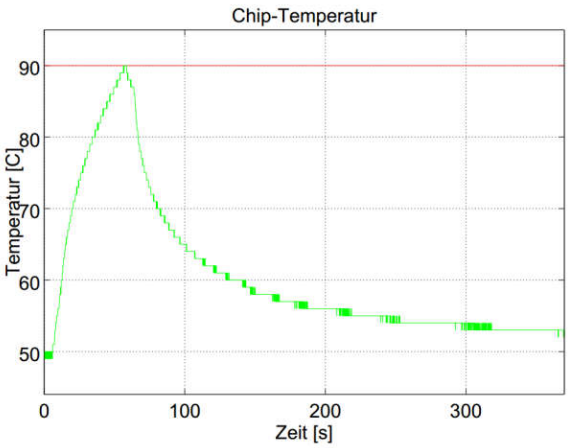

Abbildung 1: Temperaturentwicklung auf dem i.MX6 Quadcore-Prozessor ohne Kühlkörper bei einer Umgebungstemperatur von 21 °C. Im Zeitintervall von 0 s bis 60 s werden vier rechenintensive Prozesse ausgeführt.

Daniel Jäckle, daniel.jaeckle@hs-offenburg.de, Axel Sikora, axel.sikora@hs-offenburg.de, Badstraße 24, 77652 Offenburg.

erfolgreich eingesetzt werden können. Darüber hinaus stellen die Autoren von [13] mehrere PID-Regler auf Basis von Fetch Toggling vor. Basierend auf den Ergebnissen von [13] wurde nun ein PID-Regler unter Verwendung von DVFS als Stellglied entwickelt. Dafür wurde der Standard-Linux-Scheduler in der Kernel-Version 3.0 so erweitert, dass bei jedem Prozesswechsel eine Abtastung erfolgt. In [9] wird ebenfalls eine DTM-Technik vorgestellt, die bei jedem Prozesswechsel in Linux der Kernel-Version 2.6.29.1 ausgeführt wird. Allerdings wird in [9] kein PID-Regler eingesetzt, sondern ein prädiktives Verfahren, das auf Basis von thermischer Prozessklassifizierung DVFS durchführt. Die Ergebnisse von [9] zeigen jedoch, dass es möglich ist, den Scheduler mit einem DVFS-basierten DTM zu erweitern.

II. Problematik

Die grundlegende Problematik der Wärmeentwicklung von Multicore-Prozessoren wird in einem Beispiel in Abbildung 1 verdeutlicht. Die Abbildung zeigt ein Diagramm mit der Temperaturentwicklung des i.MX6 Quadcore-Prozessors beim Ausführen einer Instanz eines rechenintensiven Programms auf jedem Kern. Die Temperatur erreicht nach ca. 60 s die maximale Betriebstemperatur von 90 °C. Für die Temperaturmessungen wurde der interne Temperatursensor des Prozessors verwendet. Die Temperaturentwicklung steht im direkten Zusammenhang mit der aufgenommenen elektrischen Leistung des Prozessors und ist damit abhängig von der prozessorspezifischen Kapazität, der Taktfrequenz und der Versorgungsspannung. Die prozessorspezifische Kapazität wird durch die Summe der Kapazitäten der internen Logikgatter des Prozessors definiert. Diese Kapazitäten werden durch Schaltvorgänge aufgeladen und entladen. Die folgende Gleichung definiert die dynamische, aufgenommene Leistung mit der Abhängigkeit von der Kapazität C, der Taktfrequenz f und der Spannung U [1]:

$$P_{dynamisch} = CfU^2 \quad (1)$$

Daraus resultiert die Bausteintemperatur, die abhängig von der Umgebungstemperatur und der thermischen Leitfähigkeit zur Umgebung ist. Diese Leitfähigkeit bestimmt, wie viel Wärme des Prozessors an die Umgebung abgegeben werden kann, und ist abhängig von verschiedenen Parametern, wie z.B. Größe und Aufbau des Kühlkörpers und Konvektion. Ein DTM-Verfahren muss diese Faktoren und Parameter, die verantwortlich für die Erwärmung des Prozessors sind, berücksichtigen und in einem thermischen Modell beschreiben. In vielen Fällen ist dies nicht analytisch möglich. Oft sind Parameter bzw. Messgrößen nicht bekannt (z. B. die prozessorspezifische Kapazität) oder müssen aufwändig gemessen werden (z.B. die Umgebungstemperatur). Soll ein DTM-Verfahren auf verschiedenen Plattformen eingesetzt werden, ist

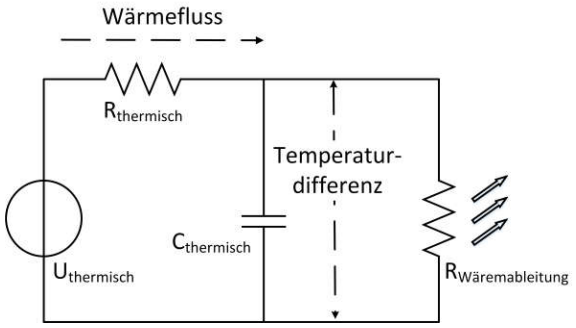

Abbildung 2: Modellierung der Erwärmung des i.MX6 Quadcore-Prozessors mit einem thermischen RC-Glied [2].

weiterführend zu beachten, dass die Parameter bei verschiedenen Prozessoren abweichen und je nach Einsatzgebiet unterschiedliche Umgebungsparameter vorherrschen. Eine Temperaturkurve wie in Abbildung 1 kann bei anderen Prozessoren oder Platinen eine andere Charakteristik aufweisen, da die Temperaturabgabe an die Umgebung von verschiedenen Faktoren wie passive Kühlkomponenten, Printed Circuit Board (PCB) Design und dem Gehäuse abhängt [1].

Nicht alle Prozessoren unterstützen interne Temperatursensoren oder haben Regelungsmechanismen für eine Frequenzskalierung. Beispielsweise können Multicore-Prozessoren keinen, einen oder mehrere interne Temperatursensoren haben oder können die Taktfrequenz einzeln für jeden Kern (lokal) oder zusammen für alle Kerne steuern (global).

Zusätzlich muss der Zielkonflikt zwischen der möglichst guten Nutzung der Rechenleistung und einem möglichst geringen zusätzlichen Rechenaufwand eines Verfahrens zur Temperaturregelung betrachtet werden. Zusammenfassend ist DTM ein komplexes Problem, das von verschiedenen Faktoren abhängig ist. In den nächsten Kapiteln beschreiben wir, wie wir diese Problematik in einem Grundaufbau mit einem PID-Regler lösen.

III. Thermisches Modell

Viele bekannte Arbeiten verwenden thermische Modelle, um die Betriebstemperaturen von Prozessoren zu berechnen. In [2] wird ein Verfahren vorgestellt, das die Temperaturerwärmung in Analogie zu einem thermischen RC-Glied modelliert. Dadurch können die Temperaturen für verschiedene Komponenten eines Prozessors durch Leistungsmessung berechnet werden, beispielsweise für einen Kern eines Multicore-Prozessors [2]. Das Verfahren aus [2] vergleicht den Wärmefluss mit einem Strom, der durch einen thermischen Widerstand fließt und eine Temperaturdifferenz hervorruft, die mit einer Spannung korrespondiert. Abbildung 2 zeigt die Modellierung eines Kerns mit einem thermischen RC-Glied. In einem Multicore-Prozessor kann der Wärmefluss und die resultierende Temperaturdifferenz für jeden Kern einzeln wie in Abbildung 2 beschrieben werden. Des

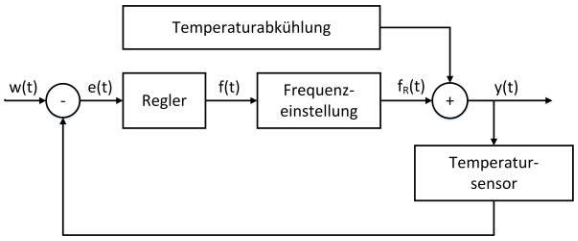

Abbildung 3: PID-Regler: Regelkreis.

Weiteren definiert [2] die Berechnung der prozessorspezifischen, thermischen Konstanten *R* und *C*. Der i.MX6 Quadcore-Prozessor hat nur einen internen, digitalen Temperatursensor. Dadurch kann die Temperatur des gesamten Prozessors ausgelesen werden. Um den zusätzlichen Rechenaufwand gering zu halten, verwenden wir für die Implementierung der Temperaturberechnung kein komplexes thermisches Modell, sondern direkt die Messwerte des integrierten Temperatursensors. Diese Technik hat zum einen den Nachteil, dass man nicht für jeden Kern die zugehörige Betriebstemperatur messen kann. Zum anderen hat der Temperatursensor nur eine Auflösung von 1 °C. Der große Vorteil liegt allerdings darin, dass keine weiteren Berechnungen zur Bestimmung der Temperatur durchgeführt werden müssen. Weiterführend ist dieses Modell nur abhängig von einem Temperatursensor. Es müssen keine spezifischen Parameter des Prozessors bekannt sein. Dadurch ist es einfach, das Verfahren auf andere Prozessoren zu übertragen.

IV. Verfügbare Regelungsmechanismen

Der i.MX6 Quadcore-Prozessor unterstützt einen Sicherheitsmechanismus, der den Prozessor vor Überhitzung schützt. Wenn die maximale Betriebstemperatur von 90 °C erreicht ist, wird automatisch die nächste, niedrigere Taktfrequenz der Kerne eingestellt. Die Messkurve in Abbildung 1 zeigt, dass die Prozessortemperatur nach dem Erreichen der maximalen Betriebstemperatur wieder fällt. Der i.MX6 skaliert automatisch die Taktfrequenzen für alle Kerne herunter. Neben globalem DVFS mit den drei Taktfrequenzen 996 MHz, 792 MHz und 396 MHz unterstützt der i.MX6 Quadcore-Prozessor noch Clock und Power Gating, GPU Clock Management und die Aktivierung bzw. Deaktivierung von internen Spannungsreglern [1]. Diese Arbeit konzentriert sich auf DVFS.

V. PID-Regler

A. Der Regelkreis

Abbildung 3 zeigt den Regelkreis des PID-Reglers. Der i.MX6 Quadcore-Prozessor unterstützt globales DVFS und verfügt über einen integrierten Temperatursensor. Somit wird der Regelkreis für die Regelung der globalen Temperatur des i.MX6 Prozessors verwendet. Der Temperatursensor stellt das Messglied dar. Die Regelabweichung ist eine Temperaturdifferenz, die als Basis für die Berechnung der nächsten Taktfrequenz verwendet wird. Im Regler wird die Regelabweichung in eine Frequenzabweichung $e(t)$ umgerechnet. Der Regler berechnet mit der Frequenzabweichung die Stellgröße $f(t)$:

$$f(t) = K_P e(t) + \sum_{i=0}^{t} K_I T_I e(i) + \frac{K_D}{T_D}(e(t) - e(t-1)) \qquad (2)$$

Der Anteil des numerischen Integrals wird mit einer minimalen bzw. maximalen Grenze limitiert, so dass er keine unverhältnismäßigen Werte annehmen kann. Die Stellgröße $f(t)$ kann kontinuierliche Werte annehmen. Da der i.MX6 Quadcore-Prozessor nur drei Taktfrequenzen unterstützt, wird die Frequenz $f(t)$ in eine Taktfrequenz $f_R(t)$ umgerechnet, die der i.MX6 Prozessor unterstützt. Die zur Taktfrequenz zugehörige Betriebsspannung wird ebenfalls gesetzt.

$$f_R(t) = \begin{cases} 996\ \text{MHz} \mid f(t) >= 996\ \text{MHz} \\ 792\ \text{MHz} \mid 996\ \text{MHz} > f(t) > 396\ \text{MHz} \\ 396\ \text{MHz} \mid f(t) <= 396\ \text{MHz} \end{cases} \qquad (3)$$

Die Abgabe von Wärme an die Umgebungstemperatur wird im Regelkreis als Störgröße modelliert. Die Abgabe ist abhängig von der aktuellen Umgebungstemperatur, welche nur mit einem weiteren, externen Temperatursensor gemessen werden kann.

B. Erweiterung des Schedulingalgorithmus

Der Regelkreis wird direkt in den Schedulingalgorithmus von Linux implementiert. Der Ausführungszeitpunkt für den Regelkreis ist ein Prozesswechsel. In Linux führt jeder Kern den Scheduler mit einem zugehörigen Set von Prozessen einzeln aus. Wann genau ein Prozesswechsel stattfindet, hängt von dem aktuellen Zustand des Systems bzw. von den auf den Kernen ausführbaren Prozessen ab. Die Linux-Kernel-Version 3.0 verwendet einen modularen Scheduler, der Prozesse in Klassen einteilt [14] [15]. Konventionelle Prozesse werden von dem Completely Fair Scheduler (CFS) behandelt und Echtzeitprozesse vom Real-Time Scheduler. In Abhängigkeit von dieser Klassifizierung berechnet bzw. legt der jeweilige Scheduler fest, wie lange ein Prozess maximal auf einem Kern ausgeführt wird, bis er wieder unterbrochen wird [15]. Ebenfalls wird damit definiert, welche anderen Prozesse, die auf die Ausführung warten, den aktuell ausgeführten Prozess unterbrechen dürfen. So kann beispielsweise die Ausführung eines konventionellen Prozesses durch einen Real-Time-Prozess unterbrochen werden, noch bevor die maximale Ausführungszeit abgelaufen ist. Die minimale Ausführungszeit (minimale Granularität) ist für Prozesse der CFS-Klasse auf eine Millisekunde festgelegt und wird zur Laufzeit dynamisch berechnet [15]. Prozesse der Real-Time-Klasse werden entweder als Round-Robin oder First-In, First-Out deklariert. Round-Robin-Prozesse haben eine maximale Standardausführungszeit von 100 ms [15], während

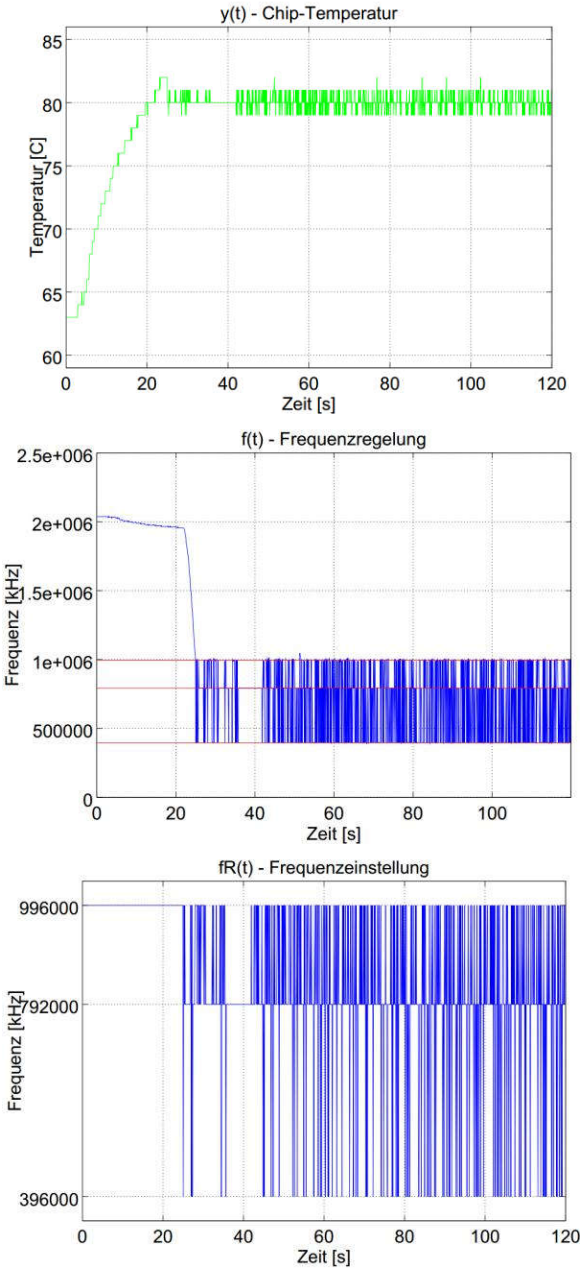

Abbildung 4: PID-Regler nach Gleichung (2) mit K_P = 0,4, K_I = 0,1, K_D = 0,2 und $T_I = T_D = 1$ bei einer Umgebungstemperatur von 21 °C.

einem First-In, First-Out-Prozess keine maximale Ausführungszeit zugeordnet ist – ein solcher Prozess wird so lange ausgeführt, bis er von einem Prozess der gleichen Klasse mit einer höheren Priorität unterbrochen wird, oder selbstständig seine Ausführung beendet [15]. Der Abtastzeitpunkt ist dadurch nicht auf ein konstantes Intervall festgelegt, sondern kann ebenso wie ein Prozesswechsel an unterschiedlichen Zeitpunkten auftreten.

VI. Experimente

Alle Experimente wurden mit gleichem Rechenaufwand auf dem i.MX6 Prozessor mit den Parametern aus Tabelle 1 durchgeführt. Das Messintervall in Tabelle 1 beschreibt das Intervall, bei dem die Taktfrequenz und die Betriebstemperatur für die Diagramme gemessen wurden. Mit dem PID-Regler aus Gleichung (3) und den Parametern K_P=0,4, K_I=0,1 und K_D=0,2 konnten wir die besten Ergebnisse erzielen, die in Abbildung 4 dargestellt sind. Zu sehen ist die stabile Regelung auf 80 °C, wobei das Quantisierungsrauschen auf Grund der Auflösung des Temperatursensors von 1 °C deutlich erkennbar ist. Der PID-Regler ist in der Lage, mit einer geeigneten Wahl der Parameter K_P, K_I, und K_D für maximale Aufheizszenarien und typische Abkühlungscharakteristiken die eingestelle Betriebstemperatur mit einem Fehler von max. 1 K einzuhalten. Die Größe dieses Fehlers ist im gegebenen Fall vom Quantisierungsfehler des Temperatursensors abhängig.

Tabelle 1: Parameter der Experimente

Umgebungstemperatur	21 °C
Solltemperatur	80 °C
Programminstanzen	4
Sättigungswerte des Integrals	+/- 100% des Sollwerts
Messintervall	100 ms
T_I, T_D	1

VII. Fazit

Diese Arbeit stellt einen PID-Regler vor, der durch eine thermische Optimierung des Prozess-Scheduling in Linux Anwendung in der Regelung der Betriebstemperatur von Multicore-Prozessoren findet. Der PID-Regler ist der Grundstein für weitere Entwicklungen. Im nächsten Schritt soll der PID-Regler durch eine Prädiktion erweitert werden, die auf thermischen Charakteristiken von Prozessen basiert und sich das thermische Modell aus [2] zu Nutze machen soll. Darauf aufbauend sollen Prozessklassen bezüglich der Taktfrequenz bevorzugt, weitere Regelungsgrößen mit einbezogen und die thermische Wechselwirkung von Prozessorkernen untereinander modelliert werden. Schließlich soll das thermische Management mit Energiemanagement verbunden werden.

Literaturverzeichnis

[1] Freescale Smiconductor, „i.MX 6 Series Thermal Management Guidelines," *Application Note* AN4579.

[2] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, D. Tarjan, „Temperature-Aware Microarchitecture," *International Symposium on Computer Architecture*, San Diego, Juni 2003.

[3] Y. Inchoon, L. C. Chun, E. J. Kim, „Predictive dynamic thermal management for multicore systems," *Design Automation Conference*, Anaheim, Juni 2008.

[4] J. Yang, Z. Xiuyi, C. Marek, Z. Youtao, J. Lingling, „Dynamic Thermal Management through Task Scheduling," *IEEE International Symposium on Performance Analysis of Systems and Software,* Austin, Juni 2008.

[5] M. D. Powell, M. Gomaa, T. N. Vijaykumar, „Heat-and-run: leveraging smt and cmp to manage power density through the operating system," *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, Boston, Oktober 2004.

[6] A. K. Coskun, T. S. Rosing, C. K. Gross, „Utilizing Predictors for Efficient Thermal Management in Multiprocessor SoCs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* Dezember 2009.

[7] J. Choi, C. Y. Cher, F. Hubertus, H. F. Hamann, A. J. Weger, P. Bose, „Thermal-aware task scheduling at the system software level," *ACM/IEEE International Symposium on Low Power Electronics and Design*, Portland, August 2007.

[8] J. Donald, M. Martonosi, „Techniques for multicore thermal management: Classification and new exploration," *International Symposium on Computer Architecture,* Washington, Juni 2006.

[9] L. Shu, X. Li, „Temperature-aware energy minimization technique through dynamic voltage frequency scaling for embedded systems," *International Conference on Education Technology and Computer*, Shanghai, Juni 2010.

[10] Y. Liu, H. Yang, R. P. Dick, W. Hui, L. Shang, „Thermal vs Energy Optimization for DVFS-Enabled Processors in Embedded Systems," *International Symposium on Quality Electronic Design,* San Jose, März 2007.

[11] J. S. Lee, K. Skadron, S. W. Chung, „Predictive Temperature-Aware DVFS," *IEEE Transactions on Computers,* Januar 2010.

[12] W. Liao, L. He, K. M. Lepak, „Temperature and supply Voltage aware performance and power modeling at microarchitecture level," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* Juni 2005.

[13] K. Skadron, T. F. Abdelzaher, M. R. Stan, „Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management," *International Symposium on High-Performance Computer Architecture,* Boston, Januar 2002.

[14] D. P. Bovet, M. Cesati, „Understanding the Linux Kernel, 3rd Edition," O'Reilly, November 2005.

[15] R. Love, „Linux Kernel Development, 3rd Edition," Addison-Wesley Professional, 2010.

Daniel Jäckle erhielt den Bachelor of Engineering im Jahr 2010 von der Dualen Hochschule Baden-Württemberg in Lörrach. Derzeit studiert er an der Albert-Ludwigs-Universität in Freiburg im Masterstudiengang Angewandte Informatik und schreibt derzeit seine Masterarbeit im Labor Embedded Systeme und Kommunikationselektronik an der Fachhochschule in Offenburg.

Dr.-Ing. Axel Sikora studierte an der RWTH Aachen Allgemeine Elektrotechnik (Dipl.-Ing.) und das Zusatzstudium zum Wirtschaftsingenieur (Dipl. Wirt.-Ing.). Er promovierte am Fraunhofer Institut für Mikroelektronische Schaltungen und Systeme in Duisburg über ein Thema der digitalen Schaltungsentwicklung auf SOI-Technologien. Anschließend arbeitete er in verschiedenen Positionen in der Telekommunikations- und Halbleiterindustrie. In 2000 wurde er als Professor an die Berufsakademie Lörrach (heute Duale Hochschule Baden-Württemberg Lörrach) berufen, wo er ab 2001 den Studiengang Informationstechnik aufbaute und leitete. 2002 gründete er das Steinbeis-Transferzentrum für Embedded Design und Networking. 2011 wurde er auf die Professur für Embedded Systeme und Kommunikationselektronik an der Hochschule Offenburg berufen und beschäftigt sich dort in Lehre und Forschung mit der Konzeption, der Implementierung und der Verifikation von sicheren und zuverlässigen drahtlosen und drahtgebundenen Kommunikationsprotokollen.

Dr. Sikora ist Autor, Koautor, Herausgeber und Mitherausgeber von zahlreichen Fachbüchern und Veröffentlichungen. Er arbeitet in den Programmkomitees verschiedener Workshops und Konferenzen mit, u.a. ist er Mitglied im Steering Board der Embedded World Conference.