

Energiebewusster Softwareentwurf für Eingebettete Systeme

Paul Ehrlich, Stephan Radke
{paul.ehrlich, stephan.radke}@eas.iis.fraunhofer.de
Fraunhofer-Institut für Integrierte Schaltungen IIS

KURZFASSUNG

Die Verlustleistung beim Entwurf batteriebetriebener eingebetteter Systeme ist von großer Bedeutung. Aktuelle Ansätze konzentrieren sich bei der Leistungsreduktion entweder auf die Hard- oder auf die Softwaredomäne. Jedoch müssen beide Bereiche in Relation zueinander betrachtet werden. Dazu präsentiert die vorliegende Arbeit ein neues Konzept, welches diese Beziehung während der Softwareentwicklung berücksichtigt. Es wird eine Abbildung des Leistungsverbrauchs der Hardware an den dafür verantwortlichen Quellcode zur Verfügung gestellt. Auswirkungen auf die Systemverlustleistung durch Änderung der Applikationssoftware werden hierbei direkt sichtbar. Dadurch wird eine Optimierung des Systems unterstützt. Mit Hilfe eines Beispiels wird das Potenzial der Methode gezeigt.

1. EINFÜHRUNG

Embedded System Design (ESD) ist ein wichtiges Forschungsgebiet [1] [2]. Es umfasst den Entwurfsablauf selbst sowie wissenschaftliche Innovationen in den einzelnen Teilschritten. Im Implementierungsschritt des ESD werden Hardware- und in Softwarekomponenten partitioniert. Das sogenannte *Hardware/Software* (HW/SW) Co-Design wird wiederholt ausgeführt. Nach jeder Iteration zeigt eine Systemsimulation, ob alle spezifizierten Randbedingungen erfüllt wurden. Falls Verletzungen vorliegen, wird das System angepasst, bis alle Bedingungen erfüllt sind.

Im ESD haben nicht funktionale Anforderungen eine hohe Bedeutung. Speziell die Randbedingungen zum Energieverbrauch sind essenziell für batteriebetriebene Systeme mit extrem niedriger Leistungsaufnahme. Beispielsweise sind Unterwasser-Sensor-Netzwerke oder Waldbrandfrüherkennungssysteme nur realisierbar, wenn sie über lange Zeit autonom betrieben werden können. Jedes Quäntchen Energie muss dabei mit Bedacht genutzt werden. Während der Implementierung im ESD durchlaufen solche Systeme deshalb die zeitaufwendige Partitionierungsschleife mehrfach.

Die Software des Mikrocontrollers, die das System maßgeblich kontrolliert, ist für den Leistungsverbrauch von großer Wichtigkeit. Bei heutigen Ansätzen fehlt jedoch die Verbindung zwischen Hard- und Software. Der Designer muss den Quellcode manuell analysieren und die Hardware mit den verantwortlichen Softwareteilen verknüpfen. Dazu werden Powertraces des hierarchischen Systems analysiert, kritische Softwareteile lokalisiert und gegebenenfalls optimiert. Zusätzlich muss der Designer nach jeder ESD Iteration die Partitionierung überdenken.

Im Paper wird eine neuartige Methode zur Unterstützung der Verbrauchsabschätzung vorgestellt. Aus der Softwareperspektive wird eine Visualisierung des gesamtheitlichen Energieverbrauchs des Systems ermöglicht und somit die Verbindung zwischen Hard- und Software geschlossen. Dazu werden Austauschformate in Form von *Extensible Markup Language* (XML)s definiert, welche den Datenaustausch standardisieren. Die Informationen der XMLs können direkt in einer *Integrated Development Environment* (IDE) dargestellt werden. Ohne Beschränkung der Allgemeinheit werden die Hardware Verhaltensmodelle in SystemC [3] beschrieben. Neben der hohen Simulationsperformance ermöglicht diese abstrakte Systembeschreibung einen flexiblen Austausch von Komponenten.

Die nachfolgende Arbeit gliedert sich wie folgt: Kapitel 2 beschäftigt sich mit dem Stand der Technik im Bereich des energiebewussten Entwurfs. Darauf folgt die Beschreibung unseres Lösungsansatzes in Kapitel 3. Kapitel 4 untersucht und veranschaulicht das Potenzial des Ansatzes anhand verschiedener Applikationen. Abschließend werden in Kapitel 5 die Ergebnisse zusammengefasst.

2. STAND DER TECHNIK

Bei der Analyse und Optimierung der Verlustleistung konzentrieren sich heutige Ansätze auf zwei separate Bereiche im ESD.

Ein Bereich beschäftigt sich mit dem Verbrauch des Systems, d.h. von dessen Hardware. Dabei gibt es zahlreiche Ansätze zur Analyse und Reduktion des Leistungsverbrauchs auf verschiedenen Stufen des Gajski-Diagramms [4]. Die Ansätze [5][6][7][8] beschreiben Leistungsanalysen der Hardware auf Systemebene und *Register Transfer Level* (RTL). Generell abstrahieren alle Ansätze die Leistungsaufnahme zu diskreten Leistungszuständen. Eine Auswertung der Simulation zeigt den gesamten Verbrauch der Hardware, was wir in unserem Ansatz zur Optimierung nutzen. Dabei wird speziell das Power-Framework [7] wieder verwendet. Von Bedeutung sind dabei die Leistungszustände und dessen Umschaltzeitpunkte.

Der zweite Bereich beschäftigt sich mit der Leistungsoptimierung in der Software des Mikrocontrollers. In den Ansätzen [9][10][11] wird der ausführbare Code analysiert, um die Leistungsaufnahme des Controllers zu minimieren. Jedoch bleibt die externe Peripherie unberücksichtigt. Für unsere Lösung verwenden wir den Leistungsprofiler aus [10], welcher einen Mechanismus zur Annotation von Leistungswerten an den Quellcode zur Verfügung stellt.

In unserem Beitrag werden beide Bereiche, d.h. Hard- und Software, in Relation gebracht, um den Verbrauch des gesamten Systems abzuschätzen.

3. METHODIK

Im Folgenden wird unsere Lösung beschrieben, d.h. wie die Leistungseigenschaften der Hardwaresimulation genutzt werden, um während der Softwareentwicklung zu unterstützen. Es wird eine Beispielplattform vorgestellt mit der anschließend unsere Lösung veranschaulicht wird.

3.1. BEISPIELPLATTFORM

Die Methodik wird anhand des Beispiels in Abb. 1a veranschaulicht, dessen Hardware in SystemC beschrieben ist. Den Kern bildet der Mikrocontroller (*MSP430F1611*). Auf diesem wird die Software ausgeführt, die die externen Komponenten steuert bzw. auswertet: eine *Light-Emitting Diode* (LED), ein Taster und ein *Humidity and Temperature Sensor* (SHT). Die LED und der Taster sind direkt über Pins mit dem Mikrocontroller verbunden, wohingegen mit dem SHT via *Inter-Integrated Circuit* (I2C) kommuniziert wird.

Die Software des Systems misst kontinuierlich die Umgebungstemperatur. Bei Druck des Tasters, wird diese als Referenz gespeichert. Weicht eine nachfolgende Messung um eine vorgegebene Toleranz ab, signalisiert das System dies mit Hilfe der LED. Die Hauptschleife des Algorithmus ist in Abb. 1b aufgeführt.

3.2. WORKFLOW

Der Ansatz besteht aus vier wesentlichen Schritten, welche in Abb. 1c gezeigt werden: die *Quellcode Kompilierung*, die *Simulation*, die *Verarbeitung* und die *Annotation*. In den folgenden Unterkapiteln wird auf jeden Schritt detailliert eingegangen.

3.2.1. QUELLCODE KOMPILIERUNG

Die Software (C-Code) repräsentiert den Ausgangspunkt des ersten Schritts im Workflow, in welchem diese kompiliert wird. Das entstandene Assemblerprogramm wird mit den Debuginformationen in einer elf-Datei [12] gespeichert. Sie werden für den Annotationsprozess benötigt und bilden zusammen die Schnittstelle zum nächsten Schritt. Für das Beispiel wurde hierfür der *mSP430-gcc* in der Version 4.45 verwendet. Als Format für die Debuginformationen kam stabs+ [13] zum Einsatz.

3.2.2. SIMULATION

Die Soft- und Hardwaredomäne der Simulation beeinflussen sich gegenseitig. Sie werden jedoch getrennt diskutiert, um die Datenquellen dieser Verbindung hervorzuheben.

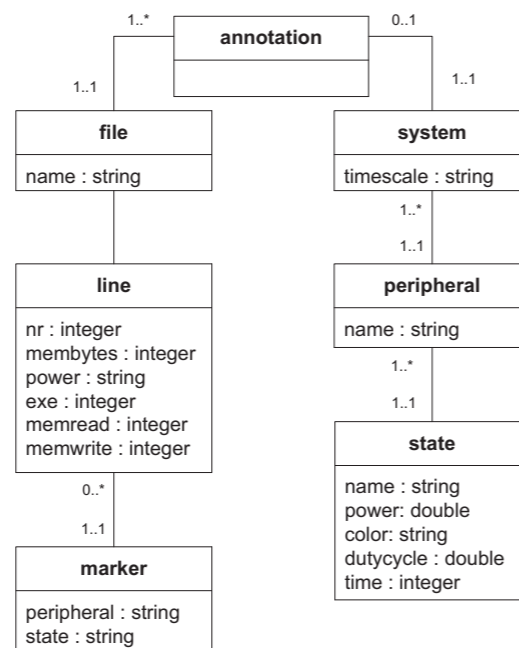
3.2.4. ANNOTATION

Um die Leistungswerte in einer IDE zu visualisieren, wird ein zweites XML beispielhaft in Abb. 3a gezeigt und mithilfe des XML Schemas in Abb. 3b definiert.

Die Systemknoten (*system*) speichern Informationen über dessen Peripherien und Zustände. Um statistischen Informationen aufzunehmen, wurden zwei neue Attribute eingeführt, welche die verbrachte Zeit in einem Zustand (*time*) und dessen Tastverhältnis (*dutycycle*) speichern. Außerdem wird die Einheit der Zeit (*timescale*) definiert. Ebenso werden die Informationen bzgl. der Software abgelegt. Das XML kann die Annotation von mehreren Quellcode-dateien (siehe *file*) enthalten. Jeder wird eindeutig über das Namenattribut (*name*) beschrieben. Jeder Dateiknoten besitzt mindestens einen Zeilenknoten (*line*), der wiederum einen oder mehrere Marker-knoten (*marker*) besitzen kann. Attribute des Zeilenknotens sind dessen Zeilennummer (*nr*), Speichergröße in Byte (*membytes*), verbraucht Energie (*power*), die Anzahl an Ausführungen (*exe*) sowie dessen Anzahl an Schreib- und Lesezugriffen des Speichers (*memwrite*). Die Zeilennummer (*nr*) dient der Identifikation und die restlichen Attribute speichern Informationen des Profilers. Die Marker enthalten wie dargestellt zwei Attribute (*perpheral, state*) zur Verknüpfung mit entsprechendem Peripheriezustand.

```
<annotation>
  <file name='test.c' >
    ...
    <line nr='54' membytes='3' power='3.8pJ' exe='1305'
      memread='2' memwrite='0' >
      <marker peripheral='SHT' />
    </line>
    <line nr='55' membytes='40' power='6.1nJ' exe='300'
      memread='9' memwrite='8' />
    <line nr='58' membytes='5' power='955pJ' exe='300'
      memread='2' memwrite='0' />
    <line nr='60' membytes='2' power='110pJ' exe='31'
      memread='1' memwrite='0' />
    <line nr='61' membytes='2' power='9.8pJ' exe='2'
      memread='0' memwrite='1' >
      <marker peripheral='LED' state='on' />
    </line>
    ...
  </file>
  <file name='include.c' >
    ...
  </file>
  <system timescale="ms">
    <peripheral name="LED" powerscale="mW">
      <state name="on" power="66.0" dutycycle="0.102"
        time="1224"/>
      <state name="off" power="0.01" dutycycle="0.898"
        time="10776"/>
    </peripheral>
    ...
  </system>
</annotation>
```

(a)



(b)

Abbildung 3: a) Einfache beispielhafte Instanz des annotations.XML b) UML Diagramm zur Repräsentation des XML Schemas vom annotation.xml

4. ERGEBNISSE

Mit dem vorgestellten Annotations-XML können verschiedene Visualisierungen in gängigen IDEs realisiert werden. Prototypisch wird es hier für das Eclipse IDE gezeigt (siehe Abb. 4). Es wird ein Ausschnitt des Beispielcode gezeigt, der mit dem vorgestellten Workflow verarbeitet wurde. Zur Optimierung der Verlustleistung bietet das IDE folgende Komponenten: eine Annotationsspalte mit verbrauchter Energie und Aufrufhäufigkeit, ein Tooltip mit Hintergrundinformation, hierarchische Marker für die Hardwarenutzung der Software und eine Tabelle mit Information zu Tastverhältnissen, Energie und Zeit der Peripheriezustände. Die statistischen Informationen zusammen mit den Markern erlauben es dem Entwickler, kritische Codestellen zu identifizieren. Außerdem werden Änderungen im Code und die resultierende Verlustleistung direkt in der IDE sichtbar.

Im Beispiel wurden drei Codesektionen ermittelt, welche zur Leistungsoptimierung genutzt werden können. Die Ansteuerung der LED wurde umgeschrieben, so dass sie blinkt anstatt permanent zu leuchten. Damit wurde die Hälfte der Leucht-Leistung eingespart. Außerdem wurde die Temperaturauflösung von 12bit auf

11bit gesenkt des SHT gesenkt. Dies verkürzt die Zeit einer Messung von 17ms auf 9ms und im gleichen Verhältnis das Tastverhältnis des *Measurement Mode*. Außerdem wurde die Anzahl der Messungen pro Sekunde von 25 auf 13 gesenkt werden, wodurch das Tastverhältnis weiter sinkt. Im Ergebnis heißt dies, dass der SHT nur noch 9% seiner Zeit misst. Alle drei Beispiele benötigen nur geringe Änderungen in der Software, führen aber zu einer Energieersparnis in der Peripherie um 51%.

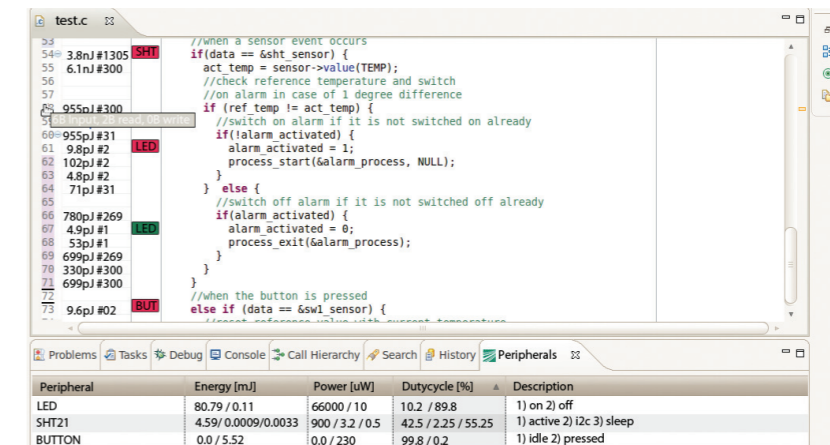


Abbildung 4: Visualisierungskonzept in das Eclipse IDE

5. ZUSAMMENFASSUNG

Um den langwierigen Prozess der Leistungsoptimierung eingebetteter Systeme zu unterstützen, wurde ein Workflow entwickelt, welcher Leistungscharakteristiken des Systems direkt am Quellcode des Mikrocontrollers sichtbar macht. Im Zusammenspiel mit einer Systembeschreibung resultiert der Workflow in einem IDE Visualisierungskonzept. Zwischen den einzelnen Schritten wurden Schnittstellen definiert, damit der Workflow allgemeingültig ist. Es wurden zwei XML-Formate vorgestellt. Eins beschreibt das Ergebnis des Simulationsschritts, das Andere beschreibt die Annotationsschnittstelle zum IDE. Die Vorteile des neuartigen Konzepts wurden beispielhaft demonstriert. Der vorgeschlagene Lösung weist großen Optimierungspotential auf, sodass in weiteren Arbeiten auch Multimastersysteme untersucht werden sollen.

Literatur

- [1] Peter Marwedel. *Embedded System Design*. Springer, 2006. ISBN 9780387300870.
- [2] Frank Vahid and Tony Givargis. *Embedded system design / a unified hardware software introduction*. Wiley, Hoboken, NJ, 2002. ISBN 9780471386780.
- [3] IEEE Standard for Standard SystemC Language Reference Manual. URL <http://www.accellera.org>.
- [4] R.A. Walker and D.E. Thomas. A Model of Design Representation and Synthesis. In *22nd Conference on Design Automation*, pages 453 – 459, june 1985.
- [5] H. Lebreton and P. Vivet. Power Modeling in SystemC at Transaction Level, Application to a DVFS Architecture. In *IEEE Computer Society Annual Symposium on VLSI*, pages 463 –466, april 2008.
- [6] M. Streubuhr, R. Rosales, R. Hasholzner, C. Haubelt, and J. Teich. ESL power and performance estimation for heterogeneous MPSoCs using SystemC. In *Forum on Specification and Design Languages*, pages 1 – 8, sept. 2011.
- [7] D. Greaves and M. Yasin. TLM POWER3: Power estimation methodology for SystemC TLM 2.0. In *Forum on Specification and Design Languages*, pages 106 –111, sept. 2012.
- [8] Robertas Damasevicius and Vytautas Stukys. Estimation of Power Consumption at Behavioral Modeling Level Using SystemC. *EURASIP J. Emb. Sys.*, 2007, 2007.
- [9] The Influence of Microprocessor Instructions on the Energy Consumption of Wireless Sensor Networks.
- [10] Ralf Hildebrandt. *Softwaremethoden zur Senkung der Verlustenergie in Microcontrollersystemen*. VDI-Verl., Düsseldorf, als ms. gedr. edition, 2007. ISBN 9783183380213.
- [11] EnergyAware Profiler. URL <http://www.energymicro.com>.
- [12] The stabs debug format. URL <http://docs.freebsd.org/info/stabs/stabs.pdf>.
- [13] Wolfgang Mauerer. *Linux Kernelarchitektur / Konzepte, Strukturen und Algorithmen von Kernel 2.6*. Hanser, München; Wien, 2004. ISBN 9783446225664.