Energiemanagement in REFLEX

Karsten Walther

Oktober 2007

Tief eingebettete Systeme wie z.B. Sensornetzknoten beziehen aktuell ihre Energie aus Batterien bzw. sollen zukünftig Umgebungsenergie nutzen [5]. Prinzipiell steht wenig Energie zur Verfügung und das Gerät und insbesondere der verwendete Mikrocontroller muss möglichst sparsam mit dieser Ressource umgehen. Dies ist eine nicht triviale Aufgabe und die Steuerung ist typischerweise tief im System verwurzelt. In dem Vortrag wird ein Energiemanagementkonzept für ereignisgetriebene Systeme am Beispiel von REFLEX vorgestellt. Die Energiesparmassnahmen erfolgen weitgehend implizit und ermöglichen eine einfache Entwicklung energieeffizienter Programme. In dem Konzept werden weiterhin die Architektureigenschaften typischer Mikrokontroller berücksichtigt.

1 Einleitung

Mikrocontrollerbasierte Systeme sind schon seit langer Zeit Bestandteile des täglichen Lebens und ihren Anwendung scheint noch lange nicht erschöpft. Ein dennoch bestehendes Problem ist die Energieversorgung bei neuen Anwendungen wie Sensornetzwerke oder kabellosen Automatisierungslösungen. Im Gegensatz zu leistungsfähigeren mobilen Geräten sollen die dort eingesetzten Systeme bei gleichzeitig wesentlich kleinerer Batteriekapazität über mehrere Monate funktionieren .

Dies führt dazu, dass möglichst kleine sparsame Mikrocontroller eingesetzt werden. Die Verwendung solcher Mikrocontroller alleine reicht jedoch nicht für Systemlaufzeiten von mehreren Monaten oder Jahren aus. Dafür müssen die vorhandenen Energiesparmodi der Hardware auch genutzt werden. Dabei stellen sich 2 große Probleme. Erstens die Diversität der Hardware, d.h. es muss eine geeignete Abstraktion gefunden werden, welche ein plattformunabhängiges aber dennoch effektives Energiemanagement möglich macht. Zweitens sollten die Mechanismen weitestgehend implizit wirken, damit der Anwendungsprogrammierer sich auf die Lösung seines eigentlichen Problems konzentrieren kann.

Im Folgenden wird ein Konzept zum Energiemanagement in tief eingebetteten Systemen und der beispielhaften Umsetzung für REFLEX [7] vorgestellt. Grundlegend ist dabei die ereignisgetriebene Abarbeitung, welche gut geeignet für typische tief eingebettete Systeme ist [6, 1].

2 Grundlagen

2.1 Technische Grundlagen

Mikrocontroller haben mehrere für das Energiemanagement relevante Eigenschaften. Sie werden z.B. mit wesentlich geringeren Taktfrequenzen betrieben als moderne Mikroprozessoren und haben auch keinen Cache. Weiterhin verfügen Mikrocontroller über integrierte Module auf dem Chip wie z.B. AD-Wandler, welche schnell an- und abgeschaltet werden können. Und es existieren meist mehrere Schlafmodi in denen ganze Modulgruppen des Controllers abgeschaltet werden können [3]. Bei dem Texas Instruments MSP430 wie auch beim Freescale HCS12 sinkt die Leistungsaufnahme im tiefsten Schlafmodus zum Beispiel auf weniger als ein Promille der Leistungsaufnahme des aktiven Modus [1].

In vielen leistungsfähigeren Geräten werden Techniken wie Dynamic Voltage Scaling (DVS) und/oder Dynamic Frequency Scaling (DFS) genutzt. Untersuchungen dieser Techniken für Low-End Mikrocontroller z.B. [4, 2] haben gezeigt, dass der Einsatz dieser Techniken dort nicht sehr vielversprechend ist. Das liegt an der geringen Last, die solche Systeme verursachen, welche die für DVS notwendige externe Beschaltung inneffektiv werden lässt. Im typischen Leistungsbereich der Plattformen liegt der Wirkungsgrad bei ca. 70%. Ein Festspannungsregler hingegen kommt auch im Niedriglastbereich auf über 90% Wirkungsgrad.

Gegen DFS spricht der nur linear zur Taktfrequemz sinkende Stromverbrauch, welcher somit nur in Kombination mit DVS Einsparung bringt aber gleichzeitig das System komplexer macht. So beruhen viele Abläufe, wie z.B. die Kommunikation mit externen Sensoren über proprietäre Schnittstellen, auf engen zeitlichen Grenzen. Schwankende Taktfrequenzen würden die Programmierung solcher Treiber nur weiter erschweren.

Daher ist die Nutzung der von den Mikrocontrollern zur Verfügung gestellten Stromsparmodi der vielversprechenste Weg [4]. Insbesondere da im Gegensatz zu leistungfähigeren Systemen mit externen Komponenten die Chip-internen schnell an- und abgeschaltet werden können.

2.2 Ereignissflussmodell

Das für die Untersuchungen verwendete Betriebssystem REFLEX stellt ein Ereignissflussmodell als Programmierschnittstelle zur Verfügung. Typische Anwendungen können damit wie in Abbildung 1 dargestellt werden.

[1] siehe Datenblatt

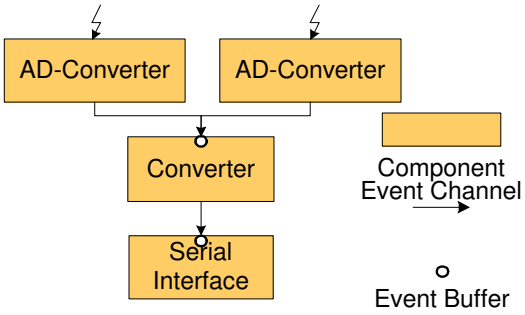

Figure 1: Event Flow Scheme of REFLEX

In diesem Modell existieren Softwaremodule, welche über Ereigniskanäle miteinander kommunizieren. In REFLEX sind das vom Laufzeitsystem verwaltbare passive Objekte. Dabei gibt es zwei Arten von Modulen. Dies sind Interrupthandler, welche infolge eines Hardwareereignisses ausgeführt werden und Aktivitäten, welche nach einem Softwareereignis gemäß einer Schedulingstrategie aufgerufen werden. Die Ereigniskanäle entkoppeln die zeitliche Ausführung von verbundenen Aktivitäten/Interrupthandler voneinander. Dazu müssen die Ereignisse, welche typischerweise mit Daten verknüpft sind, in den Kanälen gepuffert werden und können damit asynchron von dem empfangenen Modul ausgewertet werden.

REFLEX und andere ereignisgetriebene Systeme sind inherent stromsparend, da eine Verarbeitung im System nur durchgeführt wird, wenn tatsächlich etwas zu tun ist. Dies gilt insbesondere in Systemen mit wenig Last, z.B. Sensornetzwerkapplikationen.

3 Das Energiesparkonzept

Grundidee ist die Nutzung von Wissen aus der ereignisbasierten Abarbeitung eines Programms. So werden z.B. Aktivitäten im System nur ausgeführt, wenn an deren Eingängen Ereignisse aufgetreten sind, welche allesamt direkt oder indirekt von Interrupts abhängig sind. Dadurch kann ausschließlich durch das Steuern der Interruptquellen der Energieverbrauch des Systems entscheidend beeinflusst werden.

3.1 Interruptgruppen

Zur gezielten Steuerung werden Interruptquellen in Gruppen eingeordnet, welche je nach Systemmodi aktiviert oder deaktiviert werden. Vordefinierte Gruppen sind *SECONDARY*, *DREAM*, *RTI* und *UPDATE*. Dabei feuern sekundäre Interruptquellen nicht selbsständig, sondern erst in Folge von Softwareroutinen. Die Treiber solcher Komponenten sind daher selbst für das An- und Abschalten verantwortlich. Ausgabeinterrupts sind typische Vertreter dieser Gruppen. Interrupts der Gruppe *DREAM*

werden benutzt, um aus dem Traummodus (wake-on-anything-wanted) aufzuwachen, jeder nicht sekundäre Interrupt kann dieser Gruppe zugeordnet werden. Interrupts der Gruppe *RTI* werden durch die Hardware bestimmt. Normalerweise sind das die Real-Time-Clock, der Watchdog und RESET. Diese Interrupts können den Mikrocontroller auch aus typischerweise vorhandenen Tiefschlafmodi wecken, in welchen nur eine Echtzeituhr in Betrieb ist. Diese 3 Interruptgruppen werden durch das im System integrierte Energiemanagementsystem genutzt.

Die letzte Gruppe *UPDATE* ist bereits anwendungsorientiert und bestimmt einen Satz von Interruptquellen, welche während einer bestimmten Programmphase aktiv sein sollen. In diesem Fall sind das alle Interrupts, welche während einer Aktualisierung der Knotensoftware über eine I/O-Schnittstelle benötigt werden. Weitere solche Gruppen können vom Anwender angelegt werden. Beim Umschalten zwischen diesen Gruppenmodi werden entsprechend dem gewünschten Modus Interruptquellen an- bzw. abgeschaltet.

3.2 Schlafmodi

Die vordefinierten Gruppen werden vom System für allgemein verwendbare Schlafmodi genutzt. In REFLEX werden die folgenden 3 Schlafmodi angeboten, welche explizit verwendet werden können.

- *dream* (wake on anything)
- *sleep* (wake on RTC)
- *stop* (wake on RESET)

Dream dient dabei als Modus, in welchem Interrupts der Gruppe *DREAM* den Controller wieder wecken können. Der *sleep*-Modus berücksichtigt, dass die meisten Low-End Mikrocontroller einen speziellen Schlafmodus unterstützen in welchem nur ein kleiner Clock-Schaltkreis in Betrieb bleibt. Daher kann auch nur ein RTC-Interrupt (Real-Time-Clock) den Controller wecken. Dabei kann optional noch angegeben werden, wie viele RTC-Ticks das System schlafen soll. Der letzte Modus schaltet den gesamten Controller ab. Dieser Ruhezustand kann dann nur über externe Interrupts wie *RESET* wieder verlassen werden. Dies ist z.B. nützlich für batteriebetriebene oder kapazitiv arbeitende Schalter.

Der Clou an der Schnittstelle ist jedoch, dass sie nicht synchron funktioniert, d.h. der Controller wird nicht adhoc schlafen gelegt. Bevor dies geschieht, lässt das Energiemanagementsystem erst alle noch ausstehenden Aktivitäten zu Ende ausführen, lässt aber keine Interrupts mehr zu, welche im Tiefschlafmodus deaktiviert sind. Somit muss der Anwendungsprogrammierer nicht selbst überprüfen und sich eventuell benachrichtigen lassen, wann der Schlafmodus betreten werden kann.

3.3 Nullasterkennung

Die letzte Komponente im Energiemanagement ist die Nullasterkennung im System. Diese ist wichtig, da bei allen Schlafmodi von Mikrocontrollern der Rechenkern deak-

tiviert ist und daher keine Berechnungen durchgeführt werden können. Daher kann nur in Nullastsituationen in Schlafmodi gewechselt werden. In ereignisgetriebenen Systemen ist Nullast gleichzusetzen mit einer leeren ready-list im Scheduler. Der Mikrocontroller kann dann in den gewünschten Schlafmodus geschalten werden. Standardmäßig wird der maschinespezifische halt-Befehl aufgerufen, da dies dort meist der tiefstmögliche Schlafmodus ist. Dies geschieht für den Anwendungsprogrammierer vollkommen implizit.

4 Zusammenfassung

Es wurde ein Energiemanagementsystem für tief eingebettete, ereignisgetriebene Systeme präsentiert, welches die technischen Merkmale dieser Systeme, wie z.B. Verfügbarkeit von Schlafmodi, berücksichtigt. Weiterhin bietet das System eine sehr einfache Nutzerschnittstelle und nutzt durch die ereignissflussbasierte Programmierung ohnehin vorhandenes Wissen zur internen Ablaufsteuerung. Somit ist das System sehr leichtgewichtig aber verspricht dennoch eine hohe Effizienz. In den laufenden Arbeiten soll dieses durch Messungen in realen Systemen bestätigt werden.

References

[1] A.Burg. The eventflow model - a concept for real-time control of intelligent autonomous systems. Techreport, TU Passau, 1997.

[2] Y. Cho, Y. Kim, and N. Chang. Pvs: passive voltage scaling for wireless sensor networks. In *ISLPED '07: Proceedings of the 2007 international symposium on Low power electronics and design*, pages 135–140, New York, NY, USA, 2007. ACM Press.

[3] V. Ekanayake, I. Clinton Kelly, and R. Manohar. An ultra low-power processor for sensor networks. In *ASPLOS-XI: Proceedings of the 11th international conference on Architectural support for programming languages and operating systems*, pages 27–36, New York, NY, USA, 2004. ACM Press.

[4] R. Ghattas and A. G. Dean. Energy management for commodity short-bit-width microcontrollers. In *CASES '05: Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*, pages 32–42, New York, NY, USA, 2005. ACM Press.

[5] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava. Power management in energy harvesting sensor networks. *Trans. on Embedded Computing Sys.*, 6(4):32, 2007.

[6] D. B. Stewart, R. A. Volpe, and P. K. Khosla. Design of dynamically reconfigurable real-time software using port-based objects. *IEEE Trans. Softw. Eng.*, 23(12):759–776, 1997.

[7] K. Walther and J. Nolte. A flexible scheduling framework for deeply embedded systems. In *In Proc. of 4th IEEE International Symposium on Embedded Computing*, 2007.