Energieoptimiertes Scheduling für Mikrocontroller mit SDL

Marc Krämer, Alexander Geraldy

{kraemer, geraldy}@informatik.uni-kl.de
Technische Universität Kaiserslautern, Deutschland

ZUSAMMENFASSUNG

Für die Entwicklung und Einsetzbarkeit von ubiquitären Rechnersystemen und Sensornetzen spielt die energiesparende Arbeitsweise der mobilen Hardware eine elementare Rolle. Denn die heute noch zu kurze Batterielebensdauer verhindert die sinnvolle Anwendung dieser neuen Techniken noch in vielen Anwendungsbereichen. Um Energie zu sparen, ist jedoch das Zusammenspiel zwischen energiesparender Hardware auf der einen Seite und der Berücksichtigung der Hardwareaspekte in der Protokollentwicklung auf Anwendungsebene andererseits fast unabdingbar.

Wir werden im Folgenden zeigen, wie mit Hilfe eines energieorientierten Schedulers Energie gespart werden kann, ohne dass der Protokollentwickler Detailkenntnis der Hardware besitzen muss. Wir werden diesen Scheduler anhand der in der Forschung weit verbreiteten MICAz-Plattform vorstellen und evaluieren.

1. EINLEITUNG

Die Einsparung von Energie ist besonders für batteriebetriebene mobile Knoten, wie sie im ubiquitären Netzen und Sensornetzen verwendet werden, ein wichtiges Thema. Die Netzwerkknoten sollen ohne einen Eingriff, insbesondere ohne Batteriewechsel, möglichst lange arbeiten. Gleichzeitig wird versucht, die Entwicklung mobiler Applikationen auf einer hohen Entwicklungs- und Abstraktionsebene voranzutreiben, um die Entwurfskomplexität zu reduzieren. Hierbei fallen zu Verfügung stehende Möglichkeiten zur Energieeinsparung aber oft der Abstraktion zum Opfer, weil die benötigten Hardwaredetails und -funktionen durch Gerätetreiber verdeckt werden.

Diese Abstraktion durch Schichten erschwert es jedoch sowohl auf Applikations- als auch auf Treiberebene, Energieentscheidungen für den gesamten Knoten zu treffen. Es ist also notwendig, eine Integration auf einer Zwischenschicht (Betriebssystem) zu erstellen, die sowohl Informationen von der Applikation als auch Zustandsinformationen der Hardware erhält. Diese Zwischenschicht besitzt so einerseits die Information, welche Energiesparmodi verfügbar sind, und andererseits die Kenntnis, wann die Applikation einen bestimmten Energiesparzustand toleriert. Auf dieser Schicht kann also zu jedem Zeitpunkt der aktuell optimale Energiezustand für das System gefunden werden.

Wir werden nun kurz auf die zugrunde liegende Hardware und die von uns verwendeten Entwicklungstools eingehen. Anschließend stellen wir unseren energie-orientierten Scheduler vor und zeigen, dass mit ihm wertvolle Energie eingespart werden kann.

2. GRUNDLAGEN

Als Plattform für unsere Protokollentwicklung dient der MICAz, der weite Verbreitung in der Forschung gefunden hat. Für diesen MICAz wurde von uns eine Entwicklungskette entwickelt, der uns ermöglicht, Protokolle abstrakt in der formalen Spezifikationssprache SDL zu spezifiziern und automatisch, d.h. ohne weitere Eingriffe des Entwicklers, auf dem MICAz umzusetzen. Der Implementierungsschritt verläuft dadurch unmittelbar und in einer vordefinierten Weise, so dass ein direkter Zusammenhang zwischen der Protokollspezifikation und dem Verhalten des MICAz vorliegt. Die gleiche Toolkette kann verwendet werden, um weitere Plattformen, wie beispielsweise auch eine Simulationsplattform, anzusprechen.

Wir werden nun den MICAz sowie die Spezifikationssprache SDL vorstellen.

MICAz. Der MICAz [6] basiert auf dem 8 Bit-Mikrocontroller ATMega128L von Atmel [3] und dem Transceiverchip CC2420 von Chipcon [5]. Der Mikrocontroller wird mit ∼8 MHz getaktet und verwendet als zusätzlichen Zeitgeber einen externen Quartz mit 32.768 kHz. Der ATMega128L stellt verschiedene Energiesparmodi zur Verfügung. Einige der Modi können benutzt werden, um über einen Timerinterrupt zu einer bestimmten Zeit wieder aufzuwachen. In anderen Modi lässt sich der Knoten nur durch externe Interrupts, wie z.B. einen Tastendruck, wieder aufwecken [8]. Als Speicherausbau besitzt der ATMega 4 kB RAM sowie 128 kB Flash. Der Transceiver des MICAz arbeitet auf dem 2.4 MHz-Band und ist weitgehend frei programmierbar, so dass nicht unbedingt das vorgesehene ZigBee-Protokoll[2] verwendet werden muss, sondern auch eigene Protokolle entwickelt werden können. Lediglich auf die maximale ZigBee-Rahmengröße von

127 Bytes bleibt man festgelegt. Zur Einsparung von Energie stellt der Transceiver neben dem *Shutdown*-Modus noch den *Idle*-Modus zur Verfügung. Der Shutdown-Modus schaltet dabei den Transceiver vollständig aus und benötigt eine sehr lange Wiedereinschaltzeit. Im Idle-Modus ist der Wechsel in den Sende-/Empfangsmodus sehr schnell möglich, er bringt jedoch einen höheren Energieverbrauch mit sich. In beiden Modi ist kein Empfang oder Senden möglich. Nach jedem Sendevorgang wechselt der Transceiver automatisch in den Empfangsmodus zurück und zeigt damit ein ähnliches Verhalten, wie es von kabelgebundenen Halbduplexverbindungen zu erwarten ist.

SDL. Die formale Spezifikationssprache Sprache SDL [1] ermöglicht die strukturierte Beschreibung von verteilten endlichen Automaten, die über Kanäle verbunden kommunizieren. Zusätzlich lassen sich SDL-Systeme auch als offene Systeme spezifizieren, welche Signale mit ihrer Umgebung austauschen. SDL-Timer ermöglichen das Auslösen bestimmter Ereignisse zu einer vorherbestimmten Zeit. Diese Eigenschaften machen SDL zu einer hervorragende Beschreibungssprache für typische Anwendungen mit Mikrocontrollern. Zusammen mit dem Kodegenerator CMicro von Telelogic Tau [9] und der SDL Umgebungsplattform SEnF (SDL Environment Framework) [7] lässt sich aus den SDL-Systemen automatisch C-Kode generieren, der mit Hilfe existierender C-Compiler dann für die entsprechende Plattform übersetzt und ausgeführt werden kann.

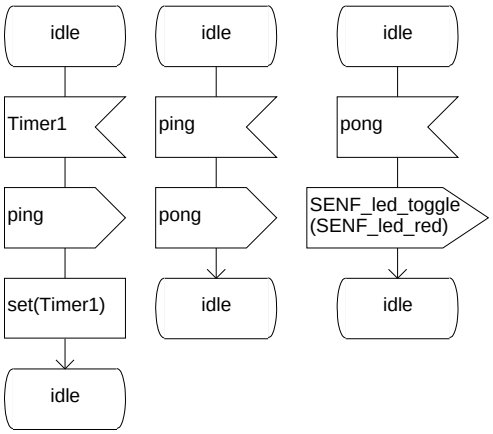

Abbildung 1: Einfaches SDL System

In Abbildung 1 ist ein einfaches (verkürztes) SDL-System gezeigt, das nach Ablauf eines Timers das Signal `ping` generiert und den Timer erneut aufzieht. Wird das Signal `ping` empfangen, wird daraus das Signal `pong` erzeugt. Bei Empfang dieses Signals wird auf dem MICAz die rote Leuchtdiode ein- bzw. ausgeschaltet.

Für die Prozessorzuteilung wird der Scheduler aus der CMicro-Implementierung von Telelogic verwendet. Dieser dient als Grundlage für die Entwicklung unseres energieorientierten Schedulers.

3. ENERGIE-SCHEDULING FÜR DEN MICAZ

Das Ziel des energie-orientierten Schedulings besteht darin, dass Protokolle energieeffizient ausgeführt werden können, ohne dass Hardwaredetails bei der Spezifikation der Protokolle einfließen müssen. Die Entscheidung über Energieressourcen soll viel mehr im Scheduler auf Betriebssystemebene getroffen werden können.

Hierzu muss die Anwendung dem Betriebssystem Details über ihr Verhalten mitteilen damit dieses Entscheidung zur Aktivierung eines Energiesparmodus geeignet treffen kann. Der Scheduler entscheidet über den nächsten zur Auswahl stehenden Prozess, kennt seine Ressourcenanforderung und weiß außerdem bei der Unterbrechung eines Prozesses, welche Ressourcen wieder frei werden. Die Verwaltung der Ressource Energie wird genau aus diesem Grund in den Scheduler integriert.

Wir werden nun am Beispiel der Hardwareplattform des MICAz unsere Erweiterung des energieoptimierten Scheduling vorstellen. Der MICAz ist hierbei nur als Stellvertreter für eine Vielzahl unterschiedlicher Hardwareplattformen zu sehen. Die an diesem Beispiel gezeigten Details lassen sich in ähnlicher Weise auch auf andere Plattformen übertragen. Zunächst werden wir auf die Hardwaredetails des MICAz eingehen, soweit sie Möglichkeiten zur Energieeinsparung betreffen. Schließlich stellen wir den SDL-Scheduler vor, sowie eine Evaluation des Schedulers.

3.1 Hardwareplattform

Systemzeit. Da SDL zeitbasierte Ereignisse (Timer) anbietet, muss von der Hardwareplattform eine Zeitbasis zur Verfügung gestellt werden. Diese Zeitbasis sollte möglichst gleichmäßig und kontinuierlich laufen. Wir verwenden für unsere Kommunikationssysteme einen interner Timer des Mikrocontrollers, der durch den externen Quartz bei 32,768 kHz getaktet wird. Die Genauigkeit der systeminternen Zeit ist damit auf $\frac{1}{32768}$ s $\approx 30\,\mu$s beschränkt. Im normalen Betriebsmodus wird der Mikrocontroller nach Ablauf der $30\,\mu$s per Interrupt unterbrochen und die Systemzeit inkrementiert.

Wegen dieses Mechanismus wäre die Energiesparphase auf $30\,\mu$s beschränkt. Atmel hat bereits vorgesehen, den Timer selbständig ein Register hochzählen zu lassen, ohne das gerade laufende Programm zu unterbrechen. Bei Erreichen einer definierten Schwelle des Zählers wird dann der Interrupt ausgelöst. Bei der 8-Bit Architektur des MICAz lässt sich die maximale Schlafzeit damit auf $\frac{254}{32768}$ s $\approx$ 7.8 ms erhöhen. Eine weitere Verlängerung der Schlafzeit wäre möglich, indem der Zeitgeber mittels Prescaler um einen Faktor verlangsamt wird. Da es während der Schlafphase zu jedem Zeitpunkt zu einer Unterbrechung durch einen externen Interrupt kommen kann und der Stand des Prescaler nicht auslesbar ist, führen diese längeren Schlafphasen jedoch zu einer Verschlechterung der Zeitgenauigkeit. Je nach Anwendungsfall ist dieses Verhalten störend oder kann im schlimmsten Fall sogar zur Fehlfunktion des Systems führen. Daher werden wir im Folgenden von einer maximalen Schlafdauer von 7,8 ms ausgehen.

Modus	Aufwach-zeit	Leistung	I/O-Clock	Inter-rupt
active	-	28 mW	X	X
idle	kurz	15 mW	X	X
ext. standby	lang	3 mW	-	X

Tabelle 1: Energieverbrauch des ATMega128L

Auswahl des Energiesparmodus. Jeder der Energiesparmodi besitzt unterschiedliche Eigenschaften. Generell gilt jedoch: Je weniger Energie der Modus verbraucht, desto weniger Funktionen stehen auf dem Sensorknoten zur Verfügung. Dadurch ist es nicht möglich, zu jedem Zeitpunkt in den Modus mit dem geringsten Verbrauch zu wechseln. In [8] wurden bereits Messergebnisse zum Energieverbrauch des MICAz veröffentlicht die wir als Grundlage verwenden. Für die Auswahl des Modus müssen zu jedem Zeitpunkt die (Anwendungs-)Anforderungen an das System bekannt sein. Wird beispielsweise die UART-Schnittstelle benötigt, kann nur noch der Idle-Modus verwendet werden. Jeder andere Energiesparmodus deaktiviert die I/O-Clock, die für die Kommunikation über die UART-Schnittstelle benötigt wird. Sobald jedoch auf dem Knoten bekannt ist, wann eine Kominikation über UART durchgeführt wird, kann der Knoten in der Zwischenzeit Energie sparen, indem er in einen Modus wechselt, der die I/O-Clock deaktiviert. Die Auswahl eines optimalen Energiesparmodus erfordert also eine sehr genaue Kenntnis der zugrundeliegenden Hardware und der Anforderungen der Anwendung. Tabelle 1 zeigt für den Prozessor einen Auszug der Funktionalität, den Energieverbauch sowie die Aufwachzeit.

Transceiver. In [8] wurde gezeigt, dass der Transceiver einer der größten Verbraucher auf dem MICAz ist. Das Hauptproblem ist hierbei, dass er im Empfangsmodus genausoviel Energie benötigt, wie im Sendemodus. Daraus folgt, dass beispielsweise eine Minimierung der Übertragungszeit keine Energieeinsparung bewirkt. Ein Energiesparmodus darf nur dann aktiviert werden, wenn keine Nachrichten von Nachbarknoten zu erwarten sind. Eine Anwendung, welche die notwendige genaue Synchronisation mit den Nachbarknoten anbietet (hier wird u.a. die bereits erwähnte genaue Zeitbasis benötigt!), und dem Scheduler die notwendige Information über mögliche Schlafzeiten anbietet, erlaubt es, drastische Energieeinsparungen vorzunehmen. Hier wird ganz deutlich, dass die Hardware oder Treiberschicht nicht ohne Anwendungs-/Protokollwissen entscheiden kann, wann welche Energiesparmodi erlaubt sind.

3.2 Energieoptimiertes Scheduling

SDL. SDL erlaubt eine Vielzahl an Auslösern zur Ausführung einer Transition. Zu diesen gehören neben den nachrichten- und timergesteuerten Transitionen u.a. auch die spontane Transition und eine bedingte ereignislose Ausführung (continuous signal). Diese beiden speziellen SDL-Konstrukte werden von CMicro aus Effizienzgründen nicht unterstützt und von uns auch bei der Optimierung des Schedulings ausgeklammert. Der Aufgabenbereich des so reduzierten SDL-Schedulers gliedert sich damit in drei Teile:

- Überprüfen, ob Timer abgelaufen sind und in Folge dessen zugehörige Prozesse aktivieren.
- Befinden sich Signale in einer Warteschlange, so werden die entsprechenden Prozesse abgearbeitet.
- Außerdem wird die Warteschlange der externen Signale auf neue Einträge überprüft.

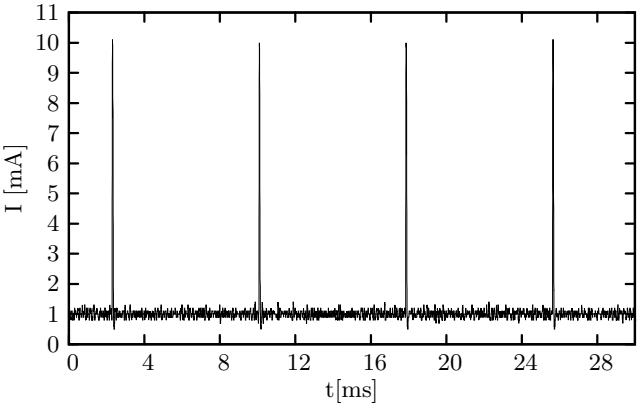

Abbildung 2: Energieverbrauch des MICAz

Der Knoten kann also bei einer leeren Signalwarteschlange bis zum Auftreten des nächsten Timers schlafen. Externe Ereignisse erfordern eine Interaktion des Prozessors, lösen also einen Interrupt aus. Die daraus resultierenden SDL-Signale werden in die externe Signalwarteschlange eingefügt. Sobald der Prozessor aus der Interrupt-Routine zurückkehrt muss er diese immer überprüfen.

Im besten Fall lässt sich die Schlafzeit auf dem verwendeten Knoten auf 7.8 ms ausdehnen. Nach dieser Zeit muss die Systemzeit weitergezählt werden und der Knoten kann ggf. weiterschlafen. In Abbildung 2 ist dieser Zyklus für ein leeres SDL-System gezeigt. Wie deutlich zu erkennen ist, beträgt der Stromverbrauch des MICAz im Betrieb $\sim$10 mA und im Schlafmodus $\sim$1 mA. Nur durch das Hinzufügen von Kommunikationskomponenten (UART, Transceiver) steigt der Energieverbrauch, ohne das sie im System verwendet werden. Deshalb wird die Modellierung dieser Komponenten im weiteren genauer betrachtet.

Paketbasierte Kommunikation (Transceiver). Statt nun die Energiemodi des Transceivers per Treiberaufruf zu steuern, was im Falle des UART so nicht umzusetzen wäre, gehen wir beim Transceiver, wie auch beim UART, den Weg über den expliziten Empfang. Dies bedeutet, dass nicht nur Sendewünsche an den Treiber mit den zu versendenden Daten weitergereicht werden, sondern auch Empfangswünsche an den Treiber geschickt werden. Der Treiber wartet ab diesem Zeitpunkt so lange, bis Daten eingehen und reicht diese dann an die Anwendung weiter. Die Zeit bis zum eigentlichen Empfang der Daten sollte dann so kurz wie möglich sein, um hier nicht unnötig Energie zu verbrauchen. Danach schaltet sich der Treiber wieder ab bis der nächste Empfangswunsch gesendet wird. Diese zwar sehr einfache, aber effektive Änderung ermöglicht es dem Treiber direkt zu entscheiden, in

welchem Modus er sich befindet und der Scheduler kann aufgrund dieser Informationen einen günstigeren Energiesparmodus auswählen. Zusätzlich sind bereits in der Spezifikation der Protokolle Übertragungszeitpunkte gekennzeichnet und können dort auch dokumentiert werden. Statische Analysen oder Simulationen des Systems können so sehr einfach Protokollfehler, die durch den Wechsel in Energiesparmodi entstehen können, aufdecken.

Serielle Kommunikation (UART). Die UART-Schnittstelle zeigt im Vergleich zum Transceiver einige Unterschiede. Es handelt sich um eine bidirektionale serielle Schnittstelle. Die Bidirektionalität der Schnittstelle drückt sich durch den weiteren Zustande „Senden + Empfangen“ aus. Die Tatsache, dass es sich um serielle Daten handelt, die kontinuierlich gesendet werden, macht den Einsatz des expliziten Empfangens zunächst unmöglich. Meist werden über dieses Schnittstelle auch Informations-Blöcke ausgetauscht. Zur Erkennung müssen entweder die Protokolle, die für die Schnittstelle eingesetzt werden, derart abgeändert werden, dass sich Blockgrenzen erkennen lassen und danach der UART auf nicht empfangen umgeschaltet. Hierfür wäre der Zugriff auf diese Protokolle nötig und die Implementierung eines Teilprotokolls würde im Hardware-Treiber erledigt. Eine andere Möglichkeit Blockgrenzen zu erkennen besteht in der Annahme, dass zwischen zwei Blöcken immer eine Pause entsteht, die länger ist, als die Zeit die sich rechnerisch aus der Übertragung zwischen zwei Bytes ergibt. Bei einer Übertragungsrate von 9600 Baud/s ist die Dauer bis das nächste Zeichen eintrifft $\sim$1 ms. Ist also nach 2 ms kein weiteres Byte eingetroffen kann man von einer Blockgrenze ausgehen. Durch diese Annahme verhält sich die serielle Kommunikation wie eine paketbasierte Vollduplexverbindung. Es kann damit aus SDL der Empfang und das Senden eines Paketes zu einem bestimmten Zeitpunkt spezifiziert werden.

3.3 Evaluation

Im Rahmen einer studentischen Arbeit [4] wurden die beschriebenen Änderungen am SDL-Scheduler von CMicro implementiert und getestet. Dabei wurden die zusätzlichen Signale der Anwendung an die Treiber weitergegeben und für den Scheduler zugänglich gemacht. Dadurch bleibt die Kapselung der Aufgaben und der Zustände weiterhin in der jeweiligen Komponente erhalten. Zusätzlich kann der Scheduler den aktuellen Zustand eines Treibers auswerten und aufgrund dieser Informationen eine Entscheidung zur Abschaltung der Komponente treffen.

Zur weiteren Evalution wurde als Anwendungsszenario ein Alarmierungssystem angenommen, wobei an den MICAz zwei Tasten angeschlossen wurden, über die sich ein Alarm an einer Basis-Station auslösen bzw. wieder ausschalten lässt. Die Basis-Station war hierbei mit einer permanenten Stromversorgung ausgestattet, wodurch die Synchronisation mit dem MICAz entfällt. Der MICAz muss sich jedoch alle 5 Minuten bei der Basis melden, um anzuzeigen, dass er sich noch innerhalb der Sendereichweite befindet. Die Basis-Station kann nach diesem Signal ihrerseits innerhalb eines Zeitfensters von zwei Sekunden Daten an dem MICAz senden, um bspw. eine der Leuchtdioden einzuschalten.

Für den Ruhezustand, also ohne Alarmauslösung, lässt sich der Energieverbrauch rechnerisch relativ leicht für den MICAz bestimmen. Bei einer Laufzeit von 11 Minuten ergeben sich 3 Synchronisationszeitpunkte zu denen der Transceiver jeweils maximal 3 Sekunden aktiv sein muss. Während der Laufzeit muss der Knoten $\sim$ 85000 mal geweckt werden, um die Systemzeit weiterzuzählen. Wenn pro Inkrement die Zeit mit 1000 Prozessorinstruktionen abgeschätzt wird, ergibt sich bei der Laufzeit die Gesamtzeit zu 11.5 Sekunden. Der MICAz verbraucht mit eingeschaltetem Transceiver 29 mA, ohne aktivierten Transceiver 9 mA und im Standby-Modus 2 mA. Damit lässt sich die gesamt verbrauchte Energie rechnerisch bestimmen:

$$\begin{aligned} 9\,\mathrm{s}\cdot 29\,\mathrm{mA} + 11.5\,\mathrm{s}\cdot 9\,\mathrm{mA} + & \\ (11\cdot 60\,\mathrm{s} - 9\,\mathrm{s} - 11.5\,\mathrm{s})\cdot 2\,\mathrm{mA} & = 1643.5\,\mathrm{mAs} \\ 1643.5\,\mathrm{mAs}\cdot 3.1\,\mathrm{V} & = 5095\,\mathrm{mJ} \end{aligned}$$

In diesem Szenario würde sich der maximal mögliche Energieverbrauch zu

$$11\cdot 60\,\mathrm{s}\cdot 29\,\mathrm{mA}\cdot 3.1\,\mathrm{V} = 59334\,\mathrm{mJ}$$

ergeben. Der minimale Energieverbauch für obiges Szenario kann um $11.5\mathrm{s}\cdot 9\,\mathrm{mA}\cdot 3.1\,\mathrm{V} = 320\,\mathrm{mJ}$ geringer ausfallen als das Ergebnis des vorgestellten Schedulers. Es kann hierbei also lediglich das Inkrementieren des Timers entfallen, da in diesem Szenario die Genauigkeit des Timers nicht nötig wäre.

4. ZUSAMMENFASSUNG UND AUSBLICK

In dieser Arbeit wurde die Funktionsweise eines energieoptimierenden Schedulers für SDL am Beispiel der MICAz-Plattform vorgestellt. Durch die Einführung von Signalen, die dem Betriebssystem Kommunikationszeitpunkte anzeigen, ist es nun möglich, auf Betriebssystemebene Entscheidungen für die Einsparung von Energie zu treffen, welche die Anforderungen der Anwendung einbeziehen. Die automatisierte Einsparung von Energie ist besonders für dynamische und große Systeme eine unabdingbare Hilfe. Während die Spezifikation der Anwendung weiterhin hardwareunabhängig erfolgen kann, bietet der energieorientierte Scheduler eine automatische Ausnutzung möglicher Energiesparmodi und verhilft damit der Anwendung zu einer drastisch verlängerten Laufzeit.
In Zukunft soll der hier vorgestellte energieorientierte Scheduler um weitere Aspekte erweitert werden, die zunächst weitere Energieeinsparungen zur Folge haben. In Zunkft soll es aber auch möglich sein Abschätzungen über den Energieverbrauch des laufenden Systems anzugeben und damit die Laufzeit sehr exakt vorrauszusagen. Diese Voraussage könnte genutzt werden, um kostbare Rechenleistung sinnvoll auf Prozesse verschiedener Priorität zu verteilen oder Arbeiten auf energiestärkere Rechnerknoten im Netzwerk zu verteilen.

5. REFERENZEN

[1] CCITT Specification and Description Language (SDL), 1995. ITU-T Recommendation Z.100 (03/93) + (10/96) + (1998).

[2] IEEE standard 802.15.4-2006. `http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf`, 2006.

[3] Atmel. Datasheet ATMega128L. http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf, 2006.

[4] A. Becker. Entwicklung eines Energie-Schedulers für SDL-Systeme. Projektarbeit, Technische Universität Kaiserslautern, 2006.

[5] Chipcon. Chipcon CC2420 datasheet. http://www.chipcon.com/files/CC2420_Data_Sheet_1_3.pdf, 2005.

[6] Crossbow. Crossbow datasheet on MicaZ. http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf, 2006.

[7] I. Fliege, A. Geraldy, S. Jung, T. Kuhn, C. Webel, and C. Weber. Konzept und Struktur des SDL Environment Frameworks (SEnF). Technischer Bericht 341/05, Fachbereich Informatik, TU Kaiserslautern, 2005.

[8] M. Krämer and A. Geraldy. Energy Measurements for MicaZ Node. In *5. GI/ITG KuVS Fachgespräch „Drahtlose Sensornetze“*, page 61ff. Pedro José Marrón, 2006.

[9] Telelogic AB. Telelogic Tau SDL Suite Homepage. http://www.telelogic.com/products/tau/sdl/index.cfm, 2006.