Einführung in das Thema Wireless Sensor Networks und Vergleich von Knotentechnologien

Benjamin Wiesmüller
Betreuer: Christoph Söllner, Corinna Schmitt
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2010
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: benny.w@mytum.de

KURZFASSUNG

Diese Arbeit soll eine Einführung in das weitläufige Themengebiet der Wireless Sensor Networks (WSN) bieten. Der Schwerpunkt liegt hierbei auf einem Überblick über den Aufbau von Sensorknoten, insbesondere einem Vergleich von Anschlusssystemen für die Peripherie. Es folgt auch eine kurze Einführung in das Betriebssystem TinyOS, das häufig auf Sensorknoten verwendet wird. Schließlich wird ein Projekt zur Überwachung eines Vulkans mit WSN vorgestellt, als Beispiel für eine Anwendung dieser.

Schlüsselworte

Wireless Sensor Networks, Sensornetzwerke, Sensorknoten

1. EINLEITUNG

Durch Weiterentwicklungen in der Elektronik wurden Prozessoren und Sensoren immer kleiner, billiger und energiesparender. Dadurch wurde es möglich, kleine Sensorknoten (Abbildung 1) zu fertigen. Diese vereinen auf wenig Platz die Möglichkeit, Messungen und Berechnungen selbstständig über längere Zeit durchzuführen und die Ergebnisse per Funk weiterzuleiten. Das führte zur Entwicklung von großen Sensornetzwerken bei denen mehrere solche Knoten zusammen vernetzt werden.
Ein solches Netz kann nahe um ein zu beobachtendes Phänomen verteilt werden, sich selbstständig vernetzen und vor Ort Messungen durchführen. Denkbar wäre eine großflächige Verteilung in schwer zugänglichem Terrain oder ein mobiler Einsatz des Netzes z.B. beim Krisenmanagement bei großen Katastrophen.
Wireless Sensor Networks sind verschiedenen einschränkenden Faktoren unterworfen. Dadurch ergeben sich besondere Herausforderungen für das Software- und Hardwaredesign. Sensorknoten, die zur Messung der Umwelt eingesetzt werden, müssen dauerhaft mit wenig oder keiner Wartung auskommen und den Umwelteinflüssen widerstehen. Ein sehr wichtiger Faktor ist dabei der Energieverbrauch. Zur Versorgung verfügt ein Sensorknoten meist nur über eine Batterie. Trotz dieses begrenzten Energievorrats soll das Netzwerk lange funktionstüchtig bleiben. Zu bemerken ist hier, dass das Senden von Daten um ein Vielfaches mehr Energie benötigt als die Datenverarbeitung im Prozessor [25].
Zu Beginn der Arbeit wird in Kapitel 2 auf den allgemeinen Aufbau von Sensorknoten eingegangen. Detaillierter wird dort auf die Verbindung der Sensoren mit dem Mikrocontroller eingegangen. Hier erfolgt ein Vergleich verschiedener gängiger Bussyteme.
Danach folgt in Kapitel 3 eine Einführung in das Betriebssystem TinyOS. Dieses wird immer öfter auf Sensorknoten eingesetzt und vereinfacht den Zugriff auf die heterogene Hardware der verschiedenen Sensorknoten und erleichtert allgemein die Softwareentwicklung.
In Kapitel 4 wird ein Projekt vorgestellt, bei dem ein drahtloses Sensornetzwerk eingesetzt wurde, um die Aktivität eines Vulkans zu überwachen. Dort werden die im Projekt verwendete Hardware und die erzielten Ergebnisse der Betreiber vorgestellt.

Abbildung 1: Beispiel für Sensorknoten: MICAz von Crossbow Technology [6].

2. AUFBAU VON SENSORKNOTEN

Sensorknoten bestehen meist aus einem Mikrocontroller, einem Sende- und Empfangsgerät, der Energieversorgung und den eigentlichen Sensoren.
Der Mikrocontroller vereinigt Prozessor, Speicher und Peripheriefunktionen auf kleinem Raum, innerhalb eines Chips. Ein wichtiges Designkriterium bei Mikrocontrollern ist der Energieverbrauch, so dass diese wesentlich weniger Energie verbrauchen, aber auch weniger Rechenleistung bieten, als Standard Desktop CPUs. Oft ist auch ein Sleep-Modus vorhanden, der die CPU-Clock und andere Teile des Controllers vorübergehend abschaltet um Energie zu sparen. Daten- und Programmspeicher sind meist schon auf dem Chip integriert. Dabei wird meist Flash-Speicher verwendet. Da dieser nicht flüchtig ist, bleibt das Programm auch nach Abschalten des Controllers erhalten.
Zur Stromversorgung verfügen die Sensorknoten meist über Batterien oder Akkus.
Kommunikation erfolgt üblicherweise über Funk in den ISM-

Mote Type Year	*WeC* 1998	*René* 1999	*René 2* 2000	*Dot* 2000	*Mica* 2001	*Mica2Dot* 2002	*Mica 2* 2002	*Telos* 2004
								
Microcontroller								
Type	AT90LS8535		ATmega163		ATmega128			TI MSP430
Program memory (KB)	8		16		128			60
RAM (KB)	0.5		1		4			2
Active Power (mW)	15		15		8		33	3
Sleep Power (μW)	45		45		75		75	6
Wakeup Time (μs)	1000		36		180		180	6
Nonvolatile storage								
Chip	24LC256				AT45DB041B			ST M24M01S
Connection type	I^2C				SPI			I^2C
Size (KB)	32				512			128
Communication								
Radio	TR1000				TR1000	CC1000		CC2420
Data rate (kbps)	10				40	38.4		250
Modulation type	OOK				ASK	FSK		O-QPSK
Receive Power (mW)	9				12	29		38
Transmit Power at 0dBm (mW)	36				36	42		35
Power Consumption								
Minimum Operation (V)	2.7		2.7		2.7			1.8
Total Active Power (mW)	24				27	44	89	41
Programming and Sensor Interface								
Expansion	none	51-pin	51-pin	none	51-pin	19-pin	51-pin	10-pin
Communication	IEEE 1284 (programming) and RS232 (requires additional hardware)							USB
Integrated Sensors	no	no	no	yes	no	no	no	yes

Abbildung 2: Überblick über einige Sensorknoten [13].

Bändern. Diese Frequenzbänder sind lizenzfrei zur Funkübertragung für industrielle, wissenschaftliche und medizinische Anwendungen freigegeben [25].
Einen Überblick über die Entwicklung der Technologien für Sensorknoten bietet Abbildung 2. Gezeigt wird eine kleine Auswahl einiger Sensorknoten von 1998 bis 2004 und die jeweils verwendete Hardware.
Im folgenden werden die seriellen Bussysteme 1-Wire, SPI, I^2C und die serielle Schnittstelle RS–232 mit UART vorgestellt. Diese Systeme werden auf Mikrocontrollern bzw. Sensorknoten häufig eingesetzt um die Sensoren und sonstige Peripherie anzuschließen.

2.1 1-Wire

Hierbei handelt es sich um einen asynchronen seriellen Bus, entwickelt von Dallas Semiconductors (inzwischen Maxim Integrated Products) [7]. Dieses Bussystem kommt dabei mit nur einer Leitung aus, die sowohl zur Datenübertragung, als auch zur Stromversorgung der angeschlossenen Geräte verwendet wird. Die Kommunikation erfolgt dabei bidirektional im halbduplex Betrieb. Das heißt, Nachrichten können in beide Richtungen gesendet werden, aber nicht gleichzeitig. Ein Gerät übernimmt die Rolle eines Masters. Dieser kontrolliert den Bus und initiiert die Kommunikation mit den anderen Geräten, die dann Slaves genannt werden. Bei einem Sensorknoten wäre der Master üblicherweise der Mikrocontroller und die Sensoren die Slaves. Die 1-Wire Geräte verfügen über eine eindeutige unveränderbare Identifikationsnummer, die bei der Produktion festgelegt wird. Über diese werden die Slave-Geräte adressiert und auch der Gerätetyp identifiziert.

2.1.1 *Übertragung einer Nachricht*

Abbildung 3 zeigt den Aufbau einer Nachricht des Masters. Zum Start einer Übertragung sendet der Master ein Reset Signal. Danach befinden sich alle Geräte in einem wohl definiertem Zustand und sind bereit für die Übertragung. Anschließend folgt die *ROM Command Sequence*, die unter anderem zur Auswahl des Zielgerätes dient. Dazu mehr im Abschnitt Adressierung. Danach kommt der *Function Command*, der nun abhängig vom Zielgerät ist und die auszuführende Aktion bestimmt [7].

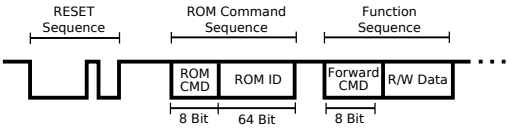

Abbildung 3: Aufbau einer 1-Wire Nachricht.

2.1.2 *Asynchrone Übermittlung der Bits*

Die Datenübertragung erfolgt asynchron. Zur Erkennung der Bits ist ein Zeitslot bestimmter Länge definiert. Erkennt ein Gerät eine abfallende Flanke, tastest es nach einer definierten Zeit das Signal ab. Für eine logische „1“ befindet sich der Pegel der Leitung nur für einen kurzen Zeitabschnitt auf Low und zum Abtastzeitpunkt wieder auf High. Für eine logische „0“ befindet sich der Pegel während des gesamten Zeitslots auf Low. Dadurch, dass jeder Zeitslot durch eine abfallende Flanke eingeleitet wird, synchronisieren sich die Geräte selbst auf jedes Bit. Die Zeitspanne zwischen dem definierten Abtastzeitpunkt nach der fallenden Flanke und dem Ende des Zeitslots, dient als Sicherheitsfenster für die Abtastung um unterschiedlich laufende Uhren in Master und Slave auszugleichen [7].

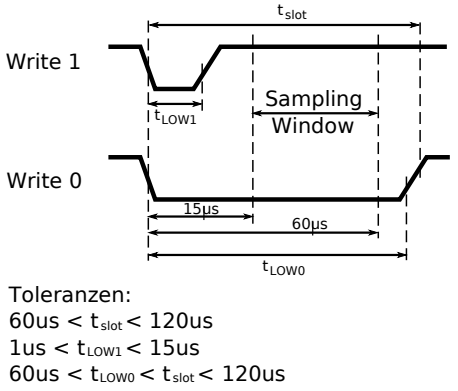

Abbildung 4: Schreiben eines Bits bei 1-wire.

2.1.3 *Adressierung*

Nach dem Reset sendet der Master eine *ROM Command Sequence*, bestehend aus einem 8 Bit *ROM Command* und der 64 Bit langen *ROM ID* des gewünschten Kommunikationspartners. Wie zu Beginn erwähnt wird diese ID schon bei der Fertigstellung des Gerätes fest eingebaut und ist einzigartig.
Die ID besteht aus drei Teilen. Der erste Teil ist der 8 Bit lange *Family Code* und ist spezifisch für den Gerätetyp. Anschließend folgt die 48 Bit lange Seriennummer. Der letzte Teil bildet ein 8 Bit langer CRC.
Von den *ROM Command*s seien hier nur zwei erwähnt. Ein *Match ROM* Befehl ermöglicht es ein bestimmtes Gerät anhand der übertragenen *ROM ID* als Kommunikationspartner auszuwählen. Der *Search ROM* Befehl startet ein Suchverfahren, bei dem alle angeschlossenen Geräte und deren IDs ermittelt werden [7].

2.1.4 *Stromversorgung*

Wenn nicht gesendet wird, befindet sich die Leitung auf dem High Pegel und lädt dabei einen Kondensator in den angeschlossenen Geräten auf. Befindet sich die Leitung auf Low versorgen sich die Geräte durch den im Kondensator gespeicherten Strom. Für Geräte mit geringem Energiebedarf, wie beispielsweise Temperatursensoren, kann dieses Verfahren zur Stromversorgung schon ausreichen [7].

2.1.5 *Erreichbare Übertragungsgeschwindigkeit*

Im regulären Modus sind Übertragungsraten von 15,4kbit/s möglich. Es wurde auch ein *Overdrive* Modus spezifiziert, bei dem Raten von bis zu 126kbit/s möglich sind. Im *Overdrive* Modus sind die Abtastfenster für die einzelnen Bits entsprechend kürzer [7].

2.2 SPI

SPI steht für Serial Peripheral Interface [2]. Ursprünglich wurde SPI von Motorla entwickelt. Eine ähnliche Spezifikation unter dem Namen MicroWire stammt von National Semiconductor [4]. Da es für SPI keine offizielle Spezifikation gibt, ist es nötig die Datenblätter der jeweiligen Geräte genau zu Rate zu ziehen. So sind auch verschiedene Modi für die Datenübertragung üblich und es werden verschieden breite Bitshift-Register verwendet. Bei SPI erfolgt die Datenübertragung seriell und wird durch ein Taktsignal synchronisiert. Dabei werden erneut zwischen Master- und Slave-Geräten unterschieden. Es werden vier Anschlüsse benötigt, zwei davon zur Datenübertragung und zwei zur Steuerung der Übertragung.

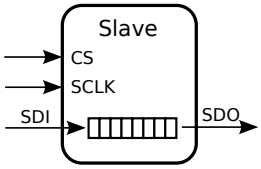

Abbildung 5: SPI-Slave.

2.2.1 *Datenübertragung und Geräteauswahl*

Der Master liefert den Takt für die synchrone Datenübertragung über die SCKL-Leitung (serial clock) [3]. Das Slave-Gerät mit dem er kommunizieren möchte wird über CS (Chip Select) ausgewählt. Über die SDI-Leitung werden die eigentlichen Daten von Master zu Slave übertragen (Input). Über SDO können Daten zurück an den Master gesendet werden werden (Output). Über SDO können auch mehrere Slave-Geräte hintereinander geschaltet werden. Dies wird im nächsten Abschnitt näher beschrieben. Dadurch dass für Input und Output extra Leitungen verwendet werden ist ein vollduplex Betrieb möglich. Das serielle Senden und Empfangen der Daten erfolgt dabei über ein Bitshift-Register, in dem die Daten gepuffert werden. Die einzelnen Bits werden entsprechend dem Takt aus/in dieses Register geschrieben und können dort gelesen/geschrieben werden. Anschließend können die Daten im Mikrocontroller wieder parallel verarbeitet werden.

2.2.2 *Kaskadierte Slaves*

Wie schon erwähnt können die Slave–Geräte kaskadiert werden oder auch direkt mit eigener CS-Leitung angesprochen werden [3]. Die kaskadierte Konfiguration ist in Abbildung 6 dargestellt. Der Output eines Slaves ist direkt mit dem Input des nächsten verbunden. Für eine sinnvolle Steuerung der Slaves müssen diese ein solches Vorgehen unterstützen und die gesendeten Daten teilweise oder ganz – je nach verwendetem Protokoll – zum nächsten Slave weiterleiten.

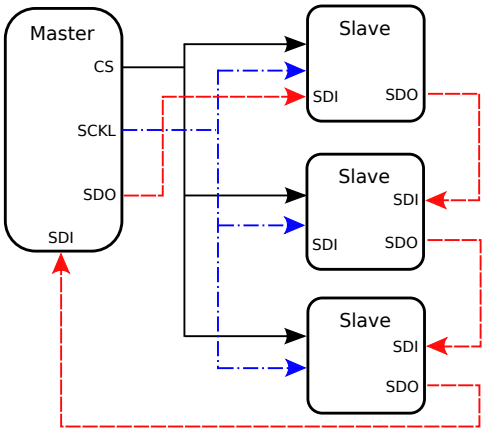

Abbildung 6: SPI mit kaskadierten Slaves.

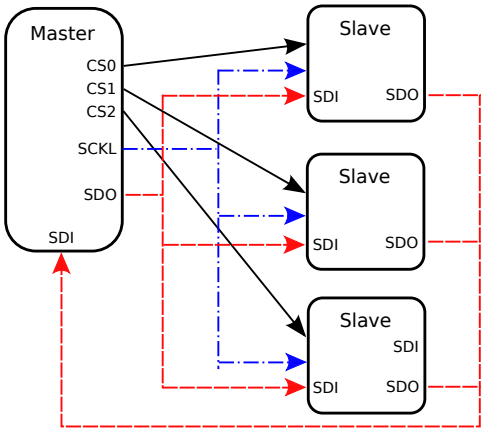

Abbildung 7: SPI mit seperaten CS-Leitungen.

2.2.3 *Einzeln angesprochene Slaves*

Bei dieser Konfiguration — gezeigt in Abbildung 7 — wird der Input durch separate CS–Leitungen auf alle Slave–Geräte gelegt. Ebenso werden die Output-Leitungen zusammengefasst und zurück geführt. Der Host kann hier den gewünschten Kommunikationspartner durch Aktivieren des zugehörigen CS auswählen. Natürlich sind auch Kombinationen aus kaskadierten und einzeln angesprochenen Slaves möglich.

2.2.4 *Erreichbare Übertragungsgeschwindigkeit*

Als erreichbare Übertragungsgeschwindigkeiten, für beispielsweise die M68HC11E Mikrocontrollerfamilie [2], wird zwischen Master und Slave unterschieden. Bei 3MHz Taktfrequenz werden für einen Master 1,5Mbit/s und für einen Slave 3Mbit/s angegeben.

2.3 I²C

Bei I²C (Inter-Integrated Circuit) handelt es sich um ein serielles Bussystem. Es wurde ursprünglich von Philips Semiconductors erfunden [5]. Es verwendet zwei Kabel, *Serial Data* (SDA) und *Serial Clock* (SCL). Jedes Gerät wird durch eine eindeutige Adresse identifiziert und kann als Empfänger oder Sender fungieren (oder beides). Gleichzeitiges Senden und Empfangen ist dabei aber nicht möglich. Die angeschlossenen Geräte werden in Master und Slave eingeteilt, wobei auch mehrere Master–Geräte an einem Bus direkt unterstützt werden. Abbildung 8 zeigt schematisch mehrere mit I²C zusammengeschlossene Geräte.

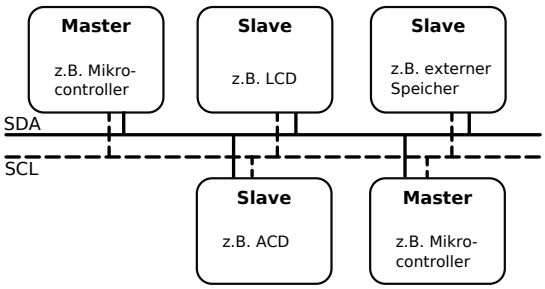

Abbildung 8: Schematischer Zusammenschluss von Master– und Slave–Geräten mit I²C.

2.3.1 *Start einer Übertragung*

Zur Kommunikation werden zwei besondere Signale definiert. Das Start- und das Stoppbit. Diese geben den Beginn und das Ende einer Übertragung bekannt und werden von einem Master erzeugt. Nach einem Startbit ist die Leitung belegt und nach einem Stoppbit wieder frei. Die Master beobachten den Bus und initiieren entsprechend keine Kommunikation bei belegtem Bus. Für den Fall, dass zwei Master fast gleichzeitig zu Senden beginnen, ist ein Arbitrierungsverfahren festgelegt, das im nächsten Abschnitt vorgestellt wird. Beginnt ein Master seine Übertragung, startet er auch mit der Generierung des Taktes zur synchronen Übertragung der folgenden gesendeten/empfangenen Bytes. Bei freiem Bus bleibt die SCL auf hohem Pegel. Durch die dominante Eigenschaft einer logischen Null, was einem niedrigen Pegel entspricht, wäre sonst kein Senden möglich. Diese Eigenschaft wird im nächsten Abschnitt näher erklärt [5].

2.3.2 *Arbitrierung*

Da bei I²C die Möglichkeit besteht, mehrere Master an einem Bus anzuschließen, könnte es dazu kommen, dass zwei solche Master Geräte gleichzeitig versuchen eine Kommunikation zu starten. Deshalb muss es ein Verfahren geben, das den Zugriff auf den Bus regelt. Dies wird allgemein Arbitrierung genannt. Bei I²C funktioniert die Arbitrierung wie im Folgenden beschrieben. Die Datenleitung SDA ist so konzipiert, dass eine logische „0“ dominiert. Das bedeutet wenn zwei Master Daten auf den Bus legen werden alle „1“en von eventuellen „0“en „überschrieben“. Ein Master beobachtet während dem Senden das entstehende Signal auf der Leitung. Stellt er fest, dass das Signal nicht seinem gesendeten entspricht, also eine gesendete „1“ einer „0“ entspricht, weiß er, dass ein anderer Master ebenfalls sendet und beendet seinen Versuch. Der Master, der die Arbitrierung „gewonnen“ hat, merkt davon nichts, da sein Signal unverfälscht übertragen wird und sendet ungestört weiter.
Zur Erklärung, wie die dominante Eigenschaft der logischen „0“ erreicht wird, dient Abbildung 9. Wie in der Abbildung gezeigt, ist der Bus über einen Pullup-Widerstand ständig mit der Versorgungsspannung +VDD verbunden. Um eine logische „0“ zu Senden, wird an Data Out eine Verbindung zur Masse hergestellt, die den ganzen Bus auf Masse „zieht“. Da der Widerstand zur Masse gegen Null geht und die Versorgungsspannung mit einem Widerstand am Bus angeschlossen ist, fließt der gesamte Strom zur Masse und der Bus befindet sich auf einem niedrigen Pegel – auch wenn mehrere Slaves versuchen eine „1“ zu senden.
Über die abgebildete Leitung Data In, kann das Slave–Gerät den Pegel auf dem Bus lesen und so, auch während eigener Übertragung, feststellen was aktuell gesendet wird.
Durch die Eigenschaft der dominanten „0“, kann auch das Taktsignal zweier Master, die gleichzeitig senden, angeglichen werden, so dass die Taktgenerierung nicht gestört wird [5].

2.3.3 *Datenübertragung und Adressierung*

Nach dem Start–Bit beginnt ein Master mit dem Senden der Adresse des Slaves, mit dem er kommunizieren möchte. Diese Adresse ist 7 Bit lang. Es gibt auch eine reservierte Adresse für einen Broadcast an alle angeschlossenen Geräte. Nach der Adresse folgt ein zusätzliches achtes Bit, dass die gewünschte Senderichtung bekannt gibt. Auf diese Weise kann der Master festlegen, ob er in dieser Übertragung Daten an

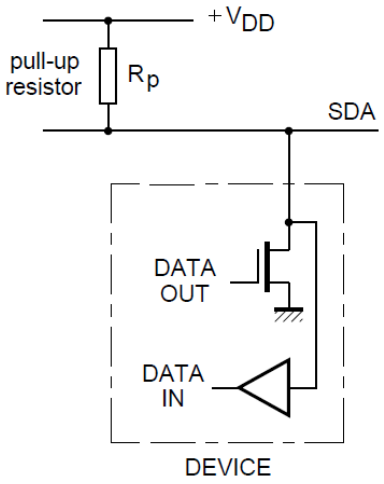

Abbildung 9: Verbindung eines Geräts an den I²C Bus [5].

den Slave senden will, oder von diesem Daten empfangen. Anschließend werden die Nutzdaten übermittelt. Es können beliebig viele Bytes übertragen werden, bis der Master mit einem Stop–Bit, das Ende der Übertragung bestimmt. Jedes Byte muss vom Empfänger mit einem Acknowledgment–Bit bestätigt werden. Durch ein Ausbleiben dieser ACKs können defekte Salves identifiziert werden.
Die Adressen der Geräte bestehen oft aus einem festen Teil und einem programmierbaren Teil. Wie viele Bits dabei programmierbar sind, hängt von der Anzahl der PINs ab, die das Gerät zur Programmierung zur Verfügung stellt. Wenn mehrere Geräte gleichen Typs an einem I²C-Bus angeschloßen sind teilen diese sich den festen Teil ihrer Adresse. Wenn dann drei PINs und damit Bits zur Programmierung zur Verfügung stehen, können maximal $2^3 = 8$ Geräte dieses Typs angeschlossen werden [5].

2.3.4 Clock Stretching

Eine Besonderheit von I²C ist das Clock Stretching. Dabei kann ein Gerät während einer Übertragung die SCL-Leitung auf niedrigem Pegel halten und so die weitere Übertragung von Daten verzögern. Dies ist z.B. nützlich, wenn ein Gerät die Daten nicht schnell genug verarbeitet. Ein Gerät, das beispielsweise gerade einen hoch priorisierten Interrupt bearbeitet und nicht antworten kann, könnte so die Kommunikation verzögern [5].

2.3.5 Spezifikation verschiedener Modi und mögliche Geschwindigkeiten

Im Standard Mode unterstützt I²C 7-Bit Adressen und Geschwindigkeiten von bis zu 100kbit/s. Es gibt verschiedene erweiterte Spezifikationen mit Modi für höhere Geschwindigkeiten oder größerem Adressraum. Beispielsweise den *Fast Mode* mit bis zu 400kbit/s, den *Fast Mode Plus (Fm+)* mit bis zu 1Mbit/s und dem *High Speed Mode* mit bis zu 3.4 Mbit/s. Diese Modi werden allerdings nicht von allen Geräten unterstützt [5].

2.4 UART

Die Bezeichnung UART steht für *Universal Asynchronous Receiver/Transmitter*. Sie dienen der Umwandlung von parallelen Daten in eine serielle Form zur Übertragung an ein anderes Gerät. UARTs werden meist zusammen mit dem Kommunikationsstandard RS–232 verwendet, der die nötigen Anschlüsse und die Signalkodierung festlegt. Durch RS–232 ist ein vollduplex Betrieb vorgesehen. Im Bereich der PCs wurde UART weitgehend von USB verdrängt. Allerdings verfügen Mikrocontroller meist immer noch über einen oder mehrere UART, der zur Kommunikation mit einem Host–PC, zur Programmierung oder zur Kommunikation mit anderen Mikrocontrollern dient. Oft werden auch USARTs verwendet die zusätzlich einen synchronen Übertragungsmodus unterstützen. Hier wird jedoch nur die asynchrone Variante vorgestellt.

2.4.1 UART Rahmen

Zur Umwandlung der parallelen Daten wird ein Bitshift-Register verwendet, dass die zu senden Bits einzeln weitergibt bzw. beim Empfang einzeln speichert und dann als Byte an den Host-Prozessor weiter gibt. Dabei sind auch zusätzliche Empfangs- oder Sendepuffer möglich. Bei UART werden die zu sendenden Daten in einen Rahmen mit Startbit, Stoppbit und Paritätsbit verpackt.Eine schematische Darstellung zeigt Abbildung 10.

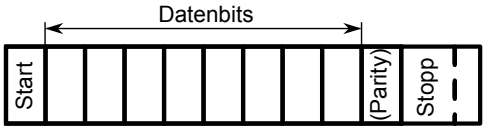

Abbildung 10: UART Rahmen mit 8 Datenbits.

Das Startbit ist dabei ein spezielles Signal um den Beginn einer Übertragung zu kennzeichnen und dem Empfänger die Synchronisation zu ermöglichen. Entsprechend kennzeichnet das Stoppbit das Ende des Rahmens. Das Paritätsbit ist entweder „gerade“, „ungerade“ oder kann ganz weggelassen werden. Durch ein gerades Paritätsbit wird die Gesamtanzahl der „1“en in den Datenbits gerade, durch das ungerade Paritätsbit entsprechend ungerade. Dies dient einer einfachen Fehlererkennung. Erkennt ein Empfänger das Startbit, beginnt er das Signal regelmäßig abzutasten um die einzelnen Bitwerte festzustellen. Die Abtastung erfolgt dabei jeweils ungefähr in der Mitte des Zeitraums, der für ein Bit festgelegt wurde. Dazu müssen Sende- und Abtastrate gleich eingestellt sein und dürfen im Laufe eines Rahmen nicht zu weit auseinander laufen. Zur Übertragung müssen zwei Geräte somit auf die selbe Übertragungsrate und Anzahl von übertragenen Datenbits eingestellt sein. Zusätzlich müssen beide auch auf die Verwendung des selben Typs von Paritätsbit und auf die selbe Art des Stoppbits eingestellt werden. Beim Stoppbit wird zwischen der Länge unterschieden, die ein Bit, ein–einhalb Bit oder zwei Bit betragen kann.

2.4.3 UART mit Adressierung: RS–485

Es existiert auch eine adressierte Version des UART. Auf Mikrocontrollern von Atmel [19] befinden sich beispielsweise USARTs, die RS–485 unterstützen. Dort können mehrere Geräte an den selben Kabeln angeschlossen werden und so

Tabelle 1: Vergleich serieller Busse

Bus	Typ	Anzahl Kabel (ohne Masse)	Adressierung	*2.4.2 Datenrate*	Sonstiges
1-Wire	asynchron, halbduplex	1	Adresse (Gerätetyp/Seriennummer in Paket-Header)	15,4kbit/s bis 126kbit/s	Stromversorgung über Datenleitung
I^2C	synchron halbduplex	2	7 Bit Adresse Nur wenige Geräte gleichen Typs	100kbit/s bis 4Mbit/s	Schnelle Betriebsmodi selten unterstützt Direkte Multi–Master Unterstützung
SPI	synchron, vollduplex	Slave: 4 Master: 4–X	Extra Kabel zur Auswahl	Master: 1,5Mbit/s Slave: 3Mbit/s	
U(S)ART RS–232	a-/synchron, vollduplex	spezieller Stecker (RS–232)	Nur zwei Geräte	bis 500kbit/s	
U(S)ART RS–485	a-/synchron, vollduplex	spezieller Stecker (RS–485)	Extra Protokoll	bis 2 MBit/s	Protokoll regelt Adressierung und Arbitrierung

ein serieller Bus geschaffen werden [20]. Üblicherweise wird dabei ein Gerät als Master und die restlichen als Slave konfiguriert. RS–485 selbst definiert nicht das zu verwendende Protokoll. Dieses muss den Zugriff auf den Bus und die Adressierung regeln.

2.5 Vergleich der Technologien

Bei allen hier vorgestellten Bussystemen handelte es sich um serielle Busse. Dort ist die Verkabelung im Vergleich zu parallelen Bussystemen einfacher. Dank hoher Frequenzen bei der Datenübertragung spielt auch der Geschwindigkeitsvorteil von parallelen Systemen keine so große Rolle und ist für die Kommunikation mit Sensoren meist nicht erforderlich. Einen Vergleich der verschiedenen Busse zeigt Tabelle 1. SPI kommt mit einem Kabel weniger als angegeben aus, wenn nur ein Master und Slave verwendet wird und der CS des Slaves permanent auf High gesetzt wird.

3. EINFÜHRUNG IN TINYOS

TinyOS [12] ist ein Betriebssystem, das speziell für Sensorknoten entworfen wurde. Es bietet dazu ein ereignisgesteuertes Ausführungsmodell, das Nebenläufigkeit unterstützt. Eine Anwendung wird erstellt, in dem verschiedene Komponenten miteinander verbunden werden. Die Komponenten sind dabei entweder vom Benutzer erstellt oder stammen aus dem von TinyOS bereitgestellten Katalog. Hardware Ressourcen werden ebenfalls als Komponenten dargestellt. TinyOS wurde in der Programmiersprache NesC geschrieben, die einen Dialekt von C darstellt. NesC biete Sprachkonstrukte, die direkt den modularen Aufbau aus Komponenten und die Kommunikation zwischen diesen unterstützt.

TinyOS ist dabei kein Betriebssystem im traditionellen Sinne. Es ist ein Framework zur Programmierung eingebetteter Systeme und eine Sammlung von Komponenten, die es ermöglichen ein anwendungsspezifisches Betriebssystem in einer Anwendung zu integrieren.

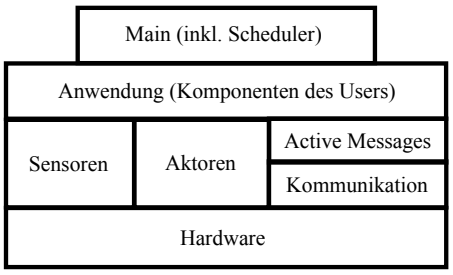

Abbildung 11: Stark vereinfachte Darstellung einer TinyOS Anwendung.

3.1 Komponentenmodell

Wie zu Beginn erwähnt besteht ein TinyOS Programm aus einer Verknüpfung von Komponenten [9]. Jede dieser Komponenten stellt eine eigene Berechnungseinheit, mit einer oder mehreren Schnittstellen, dar. Die Anwendung wird dabei durch eine Wiring Specification beschrieben, die angibt wie die einzelnen verwendeten Komponenten miteinander verbunden sind.
TinyOS abstrahiert alle Hardware Ressourcen als Komponenten. Dabei bietet TinyOS eine große Sammlung an fertigen Komponenten, die einem Anwendungsentwickler zur Verfügung stehen. Darunter befinden sich Abstraktionen für verschiedene Sensoren, Single-Hop Networking, Ad-Hoc Routing, Power Management, Timer und nicht flüchtige Speicher. Der Nachteil vieler kleiner Software Komponenten wird durch den Compiler ausgeglichen, der das entstehende Programm als ganzes analysiert und großen Gebrauch von Inlining macht.

Bei den Interfaces der Komponenten werden solchen unterschieden, die die Komponente anbietet und solche, die von ihr benützt werden. Abbildung 12 zeigt eine vereinfachte Darstellung der TimerM Komponente aus TinyOS, die die Interfaces StdControl und Timer anbietet und HWClock verwendet. Pfeile nach unten repräsentieren dabei Comman-

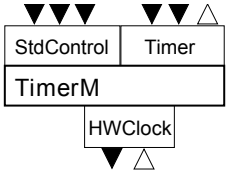

Abbildung 12: Spezifikation und Abbildung der TimerM Komponente [9].

ds, Pfeile nach oben Events.
Im Zusammenhang mit den Komponenten sind drei wichtige Abstraktionen wichtig: Commands, Events und Tasks. Commands und Events dienen dabei der Kommunikation zwischen den Komponenten, während Tasks nebenläufige Berechnungen innerhalb einer Komponente darstellen.
Durch Commands und Events wird der Aufruf eines angebotenen Services einer Komponente in zwei Phasen geteilt. Der Command wird dazu verwendet den Service anzustoßen, beispielsweise das Auslesen eines Sensorwertes, während ein Event die Fertigstellung eines solchen Services signalisiert. Der Command terminiert dabei sofort. Die Events können auch asynchron auftreten, beispielsweise durch Interrupts.
Commands und Event Handler können dabei anfallende teure Berechnungen als Tasks an den TinyOS Scheduler abgeben. Komponenten bleiben so reaktionsfähig, da Commands und Event Handler sofort terminieren. Auf den Scheduler und das Ausführungsmodell wird später näher eingegangen.

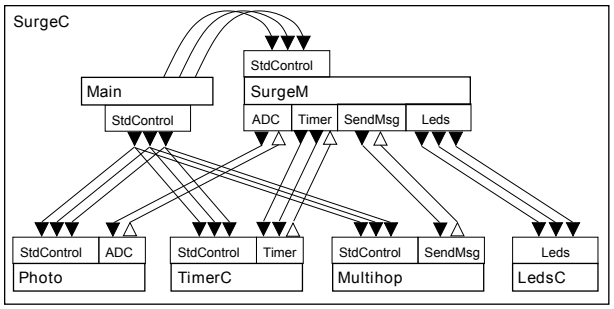

Abbildung 13: Top-level Konfiguration für Surge Anwendung [9].

Eine beispielhafte Vernetzung für eine komplette TinyOS Anwendung ist dabei in Abbildung 13 gezeigt. SurgeC ist ein einfaches Programm, das periodisch (TimerC) Werte eines Lichtsensors (Photo) ausliest und diese zurück an eine Basisstation schickt, wobei Multi-Hop Routing (Multihop) verwendet wird.

3.2 Ausführungsmodell

Sensorknoten reagieren typischerweise auf in ihrer Umgebung auftretende Ereignisse. Diese ereigniszentrierte Architektur erfordert einen feinen Grad an Nebenläufigkeit. Um dies zu unterstützen werden in TinyOS die sogenannten Tasks verwendet [9]. Diese repräsentieren fortlaufende Berechnungen und Interrupt-Handler. Dabei laufen sie einmalig, bis zu ihrer Terminierung. Tasks werden dabei an den Scheduler übergeben, der die Ausführung der Tasks regelt. Der Scheduler verwendet eine nicht preemptive FIFO Strategie. Da die Tasks nicht preemptiv sind, sind sie zueinander atomar. Allerdings gilt das nicht für Interrupt-Handler und von den Tasks angestoßene Commands und Events.
Es ist auch möglich, den Standard Scheduler durch eine eigene Implementierung zu ersetzen.
Um Race Conditions zu vermeiden, kann man kritischen Abschnitte in Tasks umwandeln oder Atomic Sections verwenden. Letztere werden in der aktuellen Implementierung von TinyOS umgesetzt, indem einfach die Interrupts für diesen Abschnitt deaktiviert werden. Außerdem wird sicher gestellt, dass ein solcher Abschnitt keine Schleifen enthält. Der NesC Compiler überprüft dabei zur Compile-Zeit ob Race Conditions auftreten. Dadurch muss man sich bei der Erstellung der Anwendungskonfiguration wenig Gedanken über Nebenläufigkeit machen und muss nicht beachten welche Komponenten Nebenläufigkeit einführen und wie diese untereinander verbunden sind. Dabei sei allerdings erwähnt, dass der Compiler leider nicht alle Race Conditions entdecken kann [9].

3.3 Netzwerkmodell

Ein wichtiger Teil von TinyOS und jedem WSN ist die Netzwerkarchitektur. Die Kommunikation wird von TinyOS durch *Active Messages*[10] abstrahiert [9]. Dabei handelt es sich um kleine 36 Byte Pakete, die mit einer 1 Byte ID versehen sind. Bei Erhalt einer Nachricht wird die Nachricht mit Hilfe eines Events an einen oder mehrere Handler weitergereicht. Diese Handler melden sich zuvor dafür an, Nachrichten diesen Typs zu erhalten. Diese Registrierung erfolgt durch die zuvor genannte Verknüpfung der Schnittstellen der Komponenten. Durch dieses Vorgehen ähnelt die Kommunikation im Netzwerk dem von TinyOS verwendeten Mechanismus der Commands und Events.
Active Messages ist ein unzuverlässiges, Single-Hop Datagram Protokoll und bietet ein einheitliches Kommunikationsinterface, sowohl für das Radio, als auch die eingebauten Seriellen Ports eines Sensorknotens. Höhere Protokolle, die Multi-Hop Kommunikation oder weitere Features anbieten, können auf den Active Messages aufsetzen. Wie erwähnt sind auch hier schon einige Komponenten in TinyOS verfügbar.

3.4 TOSSIM

TOSSIM ist ein Simulator für TinyOS Sensornetze. Mit Hilfe eines Simulators stehen weitere Debugging Möglichkeiten zur Verfügung, um das Design eines Sensornetzwerkes weiter zu untersuchen. So können Fehler gefunden werden, die in der realen Welt schwer nachzuvollziehen sind oder alternative Algorithmen ausprobiert werden.
TOSSIM versucht dabei möglichst einfach und effizient zu arbeiten und trotzdem eine Vielfalt an Interaktionen im Netzwerk abbilden zu können. Prinzipiell wird dabei versucht möglichst treu das Verhalten eines TinyOS Sensorknotens wiederzugeben. So wird die Netzwerkkommunikation auf dem Bit-Level simuliert [17].
TOSSIM simuliert diskrete Ereignisse [16]. Interrupts, Tasks, etc. werden als Ereignisse in einer Warteschlange dargestellt, die nacheinander abgearbeitet werden. Hier sieht man auch schon eine der Vereinfachungen von TOSSIM gegenüber der Realität. Alle Ereignisse werden sofort bearbeitet, Tasks besitzen keine Ausführungszeit und Interrupts treten nie während laufenden Tasks auf [17].
Um die Simulationsergebnisse übersichtlich darzustellen exis-

tieren verschiedene GUIs für TOSSIM [17].
Als Simulator werden in TOSSIM natürlich einige Vereinfachungen vorgenommen, so dass es keinen Ersatz für Tests in der realen Welt darstellt.

3.5 Fazit zu TinyOS

Inzwischen wird TinyOS in vielen WSN Projekten eingesetzt. Es selbst wird ständig weiter entwickelt – die zum Zeitpunkt dieser Arbeit neueste Version 2.1.1 wurde am 6. April 2010 veröffentlicht.
Zum Release der Version im November 2006 unterstützte TinyOS 2.0 die Plattformen EyesIFXv2, Intelmote2, Mica2, Mica2dot, MicaZ, Telosb, Tinynode und Btnode3 [11].

Durch die fertigen Komponenten für den Zugriff auf die Hardware und allgemein der Möglichkeit auf einer höheren Abstraktionsebene zu Programmieren wird durch TinyOS die Entwicklung von WSNs beschleunigt.

4. VORSTELLUNG EINES WSN PROJEKTS

Im Folgenden wird ein Forschungsprojekt vorgestellt, in dem ein drahtloses Sensornetzwerk praktisch angewendet wurde. Dabei wird auf die Ziele des Projekts, die verwendete Hardware und Sensoren und das Fazit der Betreiber eingegangen.

4.1 Projektbeschreibung

Bei diesem Projekt des Harvard Sensor Networks Lab [1], wurde 2004 (und später erneut im Jahre 2007) ein WSN eingesetzt um vulkanische Aktivität am Tungurahua in Ecuador zu messen. Bei der Vulkanforschung werden bisher meist verdrahtete Anordnungen mit mehreren Sensoren, wie akustischen Mikrophonen oder Seismographen, verwendet. In einer typische Anordnung sind mehrere Stationen um den zu beobachtenden Vulkan verteilt. Mit jeder Station sind einige wenige Sensoren in engerem Umkreis verdrahtet – z.B. fünf Stück innerhalb von 100m^2. Die dabei anfallenden großen Datenmengen werden oft in den Stationen digitalisiert und auf Festplatten gespeichert, die manuell abgeholt werden müssen. Die Anzahl und Platzierung der Sensoren an einer Station sind durch Stromverbrauch, Kabellänge und Aufzeichnungsmöglichkeiten der Daten limitiert.
Kleine günstige Sensorknoten könnten über eine größere Fläche verteilt werden. Das würde zu einer besseren Auflösung der Beobachtung führen. Mit einer zusätzlichen leistungsstarken Antenne könnten die Daten an eine entfernte Basisstation übermittelt werden. Ein weitere Vorteil eines WSN ist, dass es leichter neu programmierbar ist, und Wissenschaftler so mit verschiedenen Signalverarbeitungs- und Kompressionsverfahren experimentieren können, um die Qualität der aufgezeichneten Daten zu verbessern.

4.2 Systemarchitektur

Die verteilten Sensorknoten zeichnen mit akustischen Mikrofonen, die bei einem Ausbruch entstehenden niederfrequenten akustischen Wellen (bis zu 50Hz), auf. Die gesammelten Daten werden an einen Aggregationsknoten geschickt, der die Daten über eine drahtlose Verbindung zu einem 9km entfernten Laptop im Vulkanobservatorium weiter leitet. Zur Zeitsynchronisation der Netzwerks ist zusätzlich ein GPS-Empfangsknoten vorhanden. Der Test lief kontinuierlich über 54 Stunden [1].

4.3 Verwendete Hardware

Die Sensorknoten basieren auf einer Mica2-Plattform, bestehend aus einem 7,4MHz ATmega128L Prozessor, 128KB Speicher für den Code, 4KB Speicher für die Daten und einem Chipcon CC1000 Radiosender, der mit einer Frequenz von 433MHz sendet und eine Datenrate von ca. 34Kbps erreicht. Auf den Knoten läuft TinyOS. Als Wetterschutz diente ein einfacher Plastikbehälter in dem die Hardware untergebracht war. Zur Stromversorgung verfügt jeder Knoten über zwei AA Batterien. Der Aggregationsknoten verfügt über eine 12V Autobatterie zum Betreiben der Sendeantenne [1].

4.4 Fazit des Projekts

Die Anordnung lieferte zufriedenstellende Ergebnisse. Die Aufzeichnungen stimmten weitgehend mit denen einer naheliegenden verdrahteten Station überein. Die Verlustraten der Pakete lagen zwischen 1,5% und 5%. Ein dauerhafter Betrieb über längere Zeiträume wäre auf diese Art nicht möglich gewesen. Das kontinuierliche Senden der Daten benötigte zu viel Energie. Deshalb gibt es bei diesem Projekt auch Bestrebungen den Energieverbrauch zu verbessern, in dem ein Teil der Rechenarbeit und Interpretation der Daten auf die Knoten verlagert wird. Da Rechenzeit im Prozessor wesentlich weniger Strom verbraucht als das Senden von Daten, kann durch eine intelligente Auswahl der zu sendenden Daten Leistung gespart werden. Dazu wurde ein verbessertes System vorgeschlagen. Dort würden ein Knoten, sobald er ein „interessantes“ Ereignis beobachten, dies an seine Nachbarn mitteilen. Stimmen mehrere Knoten in dieser Meinung überein, senden sie ihre Daten weiter zum Aggregationsknoten. Insbesondere in einem größeren Netzwerk, bei dem nicht jeder Knoten direkte Verbindung zum Aggregationsknoten hat, soll ein solches Vorgehen für Einsparungen sorgen [1].

5. FAZIT & AUSBLICK

Diese Arbeit konnte nur eine kleine Einführung in das Gebiet der Wireless Sensor Networks bieten. Es wurden verwendete Hardware Technologien der Sensorknoten vorgestellt. Dabei wurde ein Überblick über die gängigsten seriellen Bussysteme geboten um einzuschätzen wann welches am einfachste und am besten geeignet ist und zu zeigen wie diese funktionieren. Wichtig bei der Auswahl ist hier auch vor allem welche Bussysteme und Modi der gewählte Mikrocontroller und die gewünschten Sensoren überhaupt unterstützen. Der Abschnitt über TinyOS zeigte wie mit Hilfe dieses Betriebssystems bzw. Frameworks Sensorknoten auf höherer Abstraktionsebene programmiert werden können. So muss bei der Entwicklung nicht mehr auf die Details der Hardware eingegangen werden und es fällt leichter das WSN als ganzes zu designen und debuggen. Zum Schluss wurde ein Projekt zur Vulkanüberwachung mit WSN gezeigt. Dieses Projekt diente als Beispiel, wie die Möglichkeiten eines WSN genutzt werden können und wo Vorteile gegenüber vorhandener Systeme liegen können.
Im Gebiet der Wireless Sensor Networks hat sich noch viel weiterer Forschungsbedarf gezeigt, um das Potential dieser auszuschöpfen. Im folgenden werden zu den genannten Punkten weiterführende Arbeiten referenziert. Ein wichtiger und interessanter Punkt, der hier nicht näher gezeigt wurde, ist z.B. die Thematik des Power Managements. Der Energieverbrauch der Sensorknoten spielt für den Wartungsaufwand und die Langlebigkeit eines WSN eine große Rolle, so dass

spezielle *Energy Aware* Algorithmen entwickelt werden, die auf geringen Energieverbrauch optimiert sind. Sehr wichtig ist hier die Verteilung der Datenlast im Netzwerk und die Fähigkeit zum ad-hoc Routing [21]. Eine weitere Optimierungsmöglichkeit, ist die *Datenaggregation* [23]. Dabei wird versucht, durch möglichst intelligentes Zusammenfassen und Verarbeiten der Daten auf den Sensorknoten die zu übertragene Datenlast zu senken. Eine andere Methode die Lebenszeit eines Sensorknoten zu erhöhen ist das *Energy Harvesting* [22], bei dem aus verschiedenen Quellen Energie gewonnen wird, um die Batterien zu laden. Beispielsweise können Sonnenlicht, Wind und Vibrationen in Energie umgewandelt werden. Als letztes sei hier die Sicherheitsthematik im Netzwerk erwähnt. Diese spielt für viele Anwendungen eine wichtige Rolle und sorgt für Forschungsbedarf [24].

6. LITERATUR

[1] Geoffrey Werner-Allen, Jeff Johnson, Mario Ruiz, Jonathan Lees, and Matt Welsh: *Monitoring Volcanic Eruptions with a Wireless Sensor Network*, In Proceedings of the Second European Workshop on Wireless Sensor Networks (EWSN 05), 2005

[2] freescale semiconductor, *M68HC11E Microcontroller Family - Data Sheet*, Chapter 8: Serial Peripheral Interface, Rev. 5.1, 07/2005, `http://www.freescale.com/files/microcontrollers/doc/data_sheet/M68HC11E.pdf`

[3] MCT Paul & Scherer Mikrocomputertechnik GmbH, *SPI - Serial Peripheral Interface*, `http://www.mct.de/faq/spi.html`

[4] National Semiconductor, *MICROWIRETM Serial Interface*, Application Note 452, January 1992, `http://www.national.com/an/AN/AN-452.pdf`

[5] Philips Semiconductors, *THE I^2C-BUS SPECIFICATION*, Version 2.1, January 2000, `http://www.nxp.com/acrobat_download2/literature/9398/39340011.pdf`

[6] *Crossbow Technology*, `http://www.xbow.com/`

[7] Maxim Integrated Products, *1-Wire Devices*, `http://www.maxim-ic.com/products/1-wire/`

[8] *Atendo GmbH & Co KG*, `http://www.atendo.de/`

[9] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer and David Culler: *TinyOS: An Operating System for Sensor Networks*, In Ambient Intelligence (2005), pp. 115-148.

[10] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser: *a mechanism for integrating communication and computation*, In Proceedings of the 19th Annual International Symposium on Computer Architecture, pages 256-266, May 1992.

[11] *TinyOS 2.0 Overview*, `http://www.tinyos.net/tinyos-2.x/doc/html/overview.html`

[12] *TinyOS Community Forum*, `http://www.tinyos.net/`

[13] Joseph Polastre, Robert Szewczyk, Cory Sharp, David Culler: *The Mote Revolution: Low Power Wireless Sensor Network Devices*, In HotChips 2004, `http://www.hotchips.org/archives/hc16/3_Tue/7_HC16_Sess7_Pres2_bw.pdf`

[14] *Willow Technologies Limited*, `http://www.willow.co.uk/TelosB_Datasheet.pdf`

[15] *SEAMONSTER: A terrestrial and marine geoscience research program at the University of Alaska Southeast*, `http://www.robfatland.net/seamonster/`

[16] Philip Levis, Nelson Lee, Matt Welsh, David Culler: *TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications*, `http://www.cs.berkeley.edu/~pal/pubs/tossim-sensys03.pdf`

[17] Philip Levis, Nelson Lee: *TOSSIM: A Simulator for TinyOS Networks*, `http://www.cs.berkeley.edu/~pal/pubs/nido.pdf`

[18] *TinyOS Tutorial – Lesson 5: Simulating TinyOS Applications in TOSSIM*, `http://www.tinyos.net/tinyos-1.x/doc/tutorial/lesson5.html`

[19] Atmel Corporation, `http://www.atmel.com/`

[20] Atmel 32-bit AVR Mictrocontrollers Application Note: *AVR32100: Using the AVR32 USART*, `www.atmel.com/dyn/resources/prod_documents/doc32006.pdf`, Rev. 32006A-AVR-04/06

[21] Mike Woo, Suresh Singh, C. S. Raghavendra: *Power-Aware Routing in Mobile Ad Hoc Networks*, 1998

[22] M Rahimi, H Shah, G Sukhatme, J Heidemann, D Estrin: *Studying the Feasibility of Energy Harvesting in a Mobile Sensor Network*, in ICRA´03 (2003)

[23] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong: *TAG: a Tiny AGgregation service for ad-hoc sensor networks*, In Proceedings of the 5th Symposium on Operating Systems Design and Implementation (2002)

[24] Adrian Perrig, John Stankovic, David Wagner, Caren Rosenblatt: *Security In Wireless Sensor Networks*, 2004

[25] O.K. Boyinbode, K.G. Akintola: *Transforming Nigeria Health Care System through Wireless Sensor Networks* In Pacific Journal of Science and Technology, Volume 10. Number 1. May 2009 (Spring), pages 312–313