WMI: Wireless Music Instruments
BSc Final Year Project
Final Report

Ilya Dmitrichenko

Department of Computing
London Metropolitan University

April 30, 2011

Abstract

This is the end of year report for a BSc degree project which had originally targeted a design of a complete wireless system for streaming of music control data collected from sensor devices and MIDI hardware to a host machine which would synthesise an audio signal to be delivered into loudspeakers or recorded on disk. Despite rather simple design idea, the research and development had taken extended time period and the emphasis of this report is on a variety of aspects related to wireless sensor network design and implementation. These aspects include: cross-platform operating system architecture, development hardware, application design and the abstraction layer requirements for interfacing hardware functions to the software as well as other miscellaneous subjects intersecting these areas. Two popular operating systems are presented in this report - *Contiki* and *TinyOS*. Two common microcontroller architectures had been used in the course of this project - 32-bit *ARM* and 8-bit *AVR*. *IEEE 802.15.4* devices where considered as the only suitable radio harware option, due to the wide use and availability of the chips which implement this protocol. Another key assumption has been that the complice to the Internet Protocol (IP) is neccessary. A simple prototype has been implemented and a set of solutions proposed for a complete product design. Several advanced subjects for further work are also addressed in this report.

Contents

List of Figures

Part I

Introduction & Research

Chapter 1

General Background

1.1 Motivation

Recent advances in low-power microcontroller and radio frequency data communications technology, together with enormous growth in the general purpose embedded computer market challenges several industries around the world. However, the music instrument industry haven't fully taken all the advantages of the most recent electronic devices and standards.

Wireless sensor networks will very soon appear in numerous application areas, which may include practically any market (consumer, industrial, medical and many other).

This project aims to implement a networked system for stage performance which uses wireless sensor devices as a creative user interface. Despite the title, this also concerns any theatrical or dance performance as well as various monitoring and control uses in entertainment systems.

1.1.1 System Specification & Requirements

This section shall outline the background for technical requirements for the system which is about to be discussed in this report. Some essential non-technical information will be introduced briefly. Below is a very general outline of what the requirements are.

- *Inexpensive hardware*
- *Low-power components*
- *Small form-factor*
- *Most recent radio technologies*
- *The system design also should:*
  - *be suitable for various similar applications*
  - *and avoid theoretical limits in scalability*

Observing contemporary arts and music scene it appears that that technology becomes increasingly popular among artists and to some surprise there are individuals who attempt to introduce non-trivial aspects of technology into visual and performed arts. Some are utilising commodity devices, such as mobile phones, while others desire to learn microcontroller programming and simple circuit design for physical interaction with sensors as a creativity instrument. As an example, Ryan Jordan in his MFA thesis [**?**] states that *"A fragile DIY hardware and software system has been created with various sensors which are attached directly to the performers body."*. The word "fragile" atracts the attention in this sentense, hence if a system was made such that an artist could apply for their performance (or display) in a flexible, robus and relieble manner, there is, probably, a niche market for it. Ryan Jordan has probably found certain intsperation in the actual wiring of his system, wireless provides a diffrent benefit to the artist.

1.2 Organisation

The very initial research and preparation phases of this project had taken place prior the start of the academic year *2010/11*. When the project has started in October, one of the first steps taken was a creation of website, which provides a number of important facilities for project management, file keeping and issue tacking. The website can be found at the URL below:

http://wmi.new-synth.info

It will be referred to throughout this report as *The WMI Website*. A current source code repository is also available on-line and linked to the website as well as work in progress notes and other supplementary information and data.

1.2.1 Report Structure

1.2.2 Typographic Conventions

Unfamiliar names and acronyms mentioned in the report are typeset in *italic*, as well as vendor brands when referred to a product from that vendor. Names of device models as well as commands and programming language keywords or statements are typed in ***bold-italic***. Filenames are `mono-spaced` and in the PDF version of the document are hyper-linked to the source code repository[1]. There will appear a special sub-section titled *"Tracked Issues"* in some sections, it shows reference to issue tickets on the website[2]. The tickets can be accessed via URL of the fallowing form (replace the *'#'* symbol with the given number):

http://wmi.new-synth.info/issues/#

[1] *The path is aways relative to the source code root directory, unless specified otherwise.*

[2] *The tickets are all enumerated in one sequence starting from 1 and classified by the type of issue each ticket relates to (i.e. "bug", "feature", "task"). Best effort was made to file these issue and there are very few remaining undocumented.*

Chapter 2

Field & Market Trends

This chapter intends to review the variety of available technologies and discuss why some desitions and preference had been made.

The terminology of the OSI 7-layer model will be used, therefore table 2.1 will illustrate the stacking of all layers in comparison to the Internet Protocol model. The reader is expected to be already familiar with this terminology, hence it is shown only for their reference.

OSI model	**IP model**
Application	Application
Presentation	
Session	TCP, UDP, ...
Transport	
Network	IP
Data Link	Data Link
Physical	Physical

Table 2.1: *Open System Interconnection Model and The Internet Protocol*

2.1 Music Control Protocols

One of additional motivations to work on low-power wireless network for music control, had been a desire to experiment in the area of high-level representation of control data, specific to live music performance and audio in general. Started by considering to extend, not so recently proposed, OSC ("Open Sound Control") protocol [30], with a set features which it appears to be missing. This is a subject to more extensive experimentation and therefore has not been included in the scope of the project itself.

OSC is effectively just an application layer data format and is mostly used with UDP. It had been proposed by a group of researchers at the Centrer for New Music & Audio Technologies (CNMAT), UC Berkley [11]. It was first presented in 1997 [31] and has received a rather limited adoption. Although, it was observed that there is a tendency towards a wider adoption of OSC - a

number of interesting hardware product had been released which use OSC as primary protocol. Some of these devices are listed below.

- *Livid Block64 [46]*
- *Jazz Mutant Lemur [38]*
- *Monome [50, 49]*

It has to be said that OSC seems to be intended as a candidate to replace the MIDI protocol (which had been define in 1983 and therefore considered in need of a replacement). The greatest limitation seen in MIDI today is the size of values it can represent, i.e. most control values a bound in 0-127 range. Nevertheless, wireless transmission of MIDI was chosen to be the initial target for this project. To avoid going outside of the scope of this report, it shall be defined that MIDI is quite likely to be most appropriate to implement at this stage, since OSC format is certainly not suitable. The reason for this is that microcontrollers do not have the capabilities to handle large amount of data represented as character strings. Therefore a new protocol needs to be designed, which would overcome most these limitations; though, that is already in the scope of another project.

2.2 Low-power Digital Radio

This section gives an overview of currently available low-power wireless communication technologies in general terms, then some of the important concepts are briefly introduced to support further discussion of these devices and software with appropriate terminology.

One of the main subjects of the initial research was wireless data communication standards for sensing and control applications and it should be noted that the transmission of audio signals has not been taken into consideration. Also as outlined in the requirenments, radio protocols such as UWB (e.g. WUSB, WiMAX) and *IEEE 802.11* (i.e. WiFi) are not low-power and therefore are not applicable for the purpose of this project. Although, short-range[1] UWB could be of great use for its potential throughput capabilities, the cost and availability of transceivers are yet unknown.

The main interest is in low-power radio of 2.4GHz range. The semiconductor market is currently flooded with a variety of inexpensive devices that implement either *IEEE 802.15.4* standard or patented protocols such as *ZigBee*, *RF4CE* or other that are based on *P802.15.4*.

A few more different low-power wireless networking technologies exist, such as *DASH7* and *ANT*. *DASH7* is an active RFID protocol for extended range and it is very specific for certain applications, it operates in 434MHz band [82]. *ANT* is a proprietary standard which uses 2.4GHz band [80]. Both of these technologies are support only by small groups of silicon chip vendors.

Multiple standards exist which are using the same hardware functions provided by *P802.14.5*-compliant devices[2], some of these are *ZigBee*, *RF4CE* and

[1] *Recent amendments in P802.15.4a specify alternative physical layer options that include sub-gigahertz UWB modes [84].*

[2] *This implies that all of these protocols are effectively defined only by software.*

6loWPAN [79]. *RF4CE* belongs to the *ZigBee* [91] family together with a number of other application area specific variations. *6loWPAN* [79] is most interesting patent-free protocol and it is transparent to existing software, since it is compliant to the Internet Protocol.

Nevertheless, all of these technologies are not yet widely available in the consumer market, where *BlueTooth* [85] and simplistic sub-gigahertz serial radio transceivers or infra-red are commonly found. Although, it is most likely that *P802.15.4* transceivers will soon dominate low-power wireless application markets.

Further in this report *P802.15.4* will be referred to as *LR-WPAN* (stands for Low-Rate[3] Wireless Personal Area Network). A number of amendments to *IEEE 802.15.4* had been published, the latest version is *P802.15.4-2006*. In this report it shall be looked at as a de-facto solution [84, 86, 33].

Some general concepts of the LR-WPAN *hardware described below are considered to be absolutely complete for understanding the system operation from the software design perspective.*

2.2.1 *IEEE 802.15.4* - Low-Rate WPAN

This standard was first proposed by the IEEE in 2003 and has evolved since. As far as the concepts essestial for application development are concerned, there is no major difference between the revisions.

It is important to understand at this point that the concept of low-power consumption applies to all layers, so the application layer indeed is required to co-operate in order to preserve the energy. However, the task of this project is to get maximum throughput on *LR-WPAN* network and attempt to reduce the latency and maximize quality of service, hence power preservation techniques are considered very briefly. However, the low-power requirement for the design is met, since the avarage power rating of the device remains relatively low without applying these techniques.

LR-WPAN defines two layers of the OSI model, the *Physical (PHY)* and *Media Access Control (MAC)* layers. Network and Applications layers are defined by other standards mentioned above.

Physical Layer (PHY)

Source: "ZigBee Wireless Networks and Transceivers"
by Shahin Farahani (2008) [29]

"The PHY layer is the closest layer to hardware and directly controls and communicates with the radio transceiver. The PHY layer is responsible for the following:

- Activating and deactivating the radio transceiver.
- Transmitting and receiving data.
- Selecting the channel frequency.

[3] *Medium-rate (MR-WPAN) are the BlueTooth (P802.15.1) networks. Also there is HR-WPAN (P802.15.3) standard defining high-rate (UWB) networks. All classes of WPAN together with WLAN are are referred to as short-range wireless networks.*

- Performing Energy Detection (ED).
  The ED is the task of estimating the signal energy within the frequency band of interest. This estimate is used to understand whether or not a channel is clear and can be used for transmission.

- Performing Clear Channel Assessment (CCA).

- Generating a Link Quality Indication (LQI).
  The LQI is an indication of the quality of the data packets received by the receiver. The signal strength can be used as an indication of signal quality."

The *LR-WPAN* standard defines use for a number of bands in different geographical regions, the 2.4GHz band can be used anywhere in the world. The details regarding RF modulation techniques and various regulations are outside of the subject of this report. There are 27 channels in different bands, 2.4GHz band has been assigned with channel numbers 11 to 26.

Power regulations apply depending on geographical region, the measures are transceiver output power and duty cycle. The global ISM band can be utilized at 100% duty on approximately 10mW level.

Two modulation techniques can be used in 2.4GHz band:

Offset-QPSK - Offset Quadrature Phase-Shift Keying [88]

DSSS - Direct-Sequence Spread Spectrum (was used in 2003 revision) [83]

Alternative modulations techniques are defined in amendment *P802.15.4a*, these include Ultra-Wide Band *(UWB)* and Chirp Spread Spectrum *(CSS)*.

Common basic data rate is 250kbps and distance coverage is from 10m to 100m, but higher limits can be achieved (up to 2Mbps).

Media Access Control (MAC)

Source: "ZigBee Wireless Networks and Transceivers" by Shahin Farahani (2008) [29]

"The MAC provides the interface between the PHY and the next higher layer above the MAC."

"The IEEE 802.15.4 defines four MAC frame structures:

- `Beacon Frame` — *used by a coordinator to transmit beacons. The beacons are used to synchronize the clock of all the devices within the same network.*

- `Data Frame` — *used to transmit data.*

- `Acknowledge Frame` — *used to acknowledge the successful reception of a packet.*

- `MAC Command Frame` — *are used for commands such as requesting the data and association or disassociation with a network.*

It is important to understand that frames from each network layer are encapsulated into each other, i.e. the MAC frame are encapsulated into PHY frames on transmission and on reception these data structure is being decoupled.

LR-WPAN Classification: Nodes and Topologies

The *LR-WPAN* device hierarchy is defined in terms of *full-* and *reduced-function* devices (FFD and RFD for short).

- Routers (FFD)
  - Network Coordinator
  - Branch Coordinator
  - Border Router
- Participant Clients (RFD or FFD)

LR-WPAN Topologies are:

- Point-to-Point
- Star
- Mesh

Feature Outline

Source: IEEE 802.15 Task Group 4 Website[34].

- Data rates of 250 kbps, 40 kbps, and 20 kbps.
- Two addressing modes: 16-bit short and 64-bit.
- Support for critical latency devices, such as joysticks.
- *Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA)* channel access mode [81].
- Automatic network establishment by the coordinator.
- Fully hand-shaked protocol for transfer reliability.
- Power management to ensure low power consumption.

2.2.2 Conclusions

It has been found that some of the concepts defined in the *IEEE* paper are of very little practical use. For example, the hierarchy specification is only used in *ZigBee* and does not apply to *6loWPAN* and the entire MAC layer specification is disregarded by the *6loWPAN* user community and many papers where presented which propose various improvements, for example [28] (One most highly regarded papers presented at the ACM conference on Embedded Networked Sensor Systems in November 2010).

It does appear that Media Access Control for *LR-WPAN* as well as IP Routing Over Low-power and Lossy networks (ROLL) [45] are subject to very extensive research at present, as well as time synchronisation [65, 40, 47, 61, 42, 43, 64, 63]. Many papers appeared on this subject and it is of greatest concert to the music instruments application, however it was found too complex to cover in the course of this project.

2.2.3 Higher Layers and the Application Layer

The greatest benefit that the OSI model gives, is that any layer can be re-implemented without any changes to other layers above or below. Also translation between different implementations would a trivial task. *6loWPAN* allows for connectivity of low-power wireless nodes to the Internet, enabling the future paradigm of *"the Internet of Things"*. This has been developed and popularised by an alliance of Internet Protocol for Smart Objects (IPSO) [35]. *6loWPAN* takes care of several aspects such as compute resources limits and address space requirements due to large amount of participant nodes by providing *IPv6* addressing with additional features such as *header compression*, reduced functionality as well as other miscellaneous enhancements [36, 37].

By enabling Internet connectivity for the sensor nodes, the application developers are presented with lesser challenge, because existing algorithms, packages and libraries can be used. Care still needs to be taken due to limitations such as bandwidth and data structure complexity.

For example, a Representational State Transfer Application Programming Interface (*RESTful API*) [90] would make a system much simpler to integrate into existing environments, there won't be a need to develop tools for any particular computer platform. *RESTful* approach brings a cross-platform compatibility out-of-the-box by utilising web-client software with very little of extra work to be done. *RESTful API* is largely used in modern web-application services (also known as "the cloud"), it uses human readable URL strings for addressing functions on remote servers via HTTP protocol (the handling of HTML or XML response is optional and not applicable here, due to above mentioned data structure complexity issue).

Another feature of *IPv6* that is very important for complex wireless networks, is *neighborhood discovery*, which eliminates any need for what DHCP served in *IPv4*. Regarding the scalability, it is true that there is no particular use in this project for 128-bit address space that *IPv6* provides. Nevertheless, this network may need to extend to a larger number of nodes if a orchestra of these is going to be built, also including various stage automation and safety sensors all in one area.

2.2.4 Alternatives and Concerns

It is very appropriate to discuss a few alternative approaches which could be taken to implement a radio network specific to the application of interactive objects for live performance.

An alternative to using *LR-WPAN* radio would be a different, rather simpler protocol and, perhaps, in a different frequency band. The reason for this is obviously because of application-specific concerns. The data transmission for musical performance is constrained to very strict real-time latency measures. In other words, when a musician is playing on stage the audience will perceive the music as very odd if the latency is too high. It would be even worse if several participants exhibit random latency, while not playing an improvised piece of music.

Using a radio protocol which is in its own *clear* frequency band and uses a minimal communication abstraction stack, should be more appropriate for latency optimisation. However, the cross-platform networking capabilities would

be lost. That might lead to incompatibilities with market trends as well as radio band regulations. Since *IEEE 802.15.4* is the current global technology trend and the 2.4GHz ISM band is accepted world-wide, the alternatives are not quite appropriate.

There are a few solutions for *non-P802.15.4* 2.4GHz ISM radio, but these would impose a vendor locking for the transceiver interfaces to such manufacturers as *Nordic Semiconductors* [52] or *Hope Microelectronics* [32]. There might be even model-specific incompatibilities, while it is quite certain that all *P802.15.4* transceiver chips will be compatible in the future.

2.3 Study of Earlier Work and Commercial Solutions

Earlier works in the field of wireless performance devices (i.e. music instruments and devices to assist dance and theater performance) were studied. A number of links to other sites and publication are listed in the WMI website sections "Links" [20] and "Bibliography" [18]. Below only the most relevant information is discussed.

2.3.1 Publications

One publication by Paulo Jorge Bartolomeu (Univ. of Aveiro, 2005) [15] which has also been submitted to the IEEE, evaluates the use of *BlueTooth* for a wireless MIDI network. Paulo Bartolomeu in his dissertation considered either *ZigBee* or *UWB* as an alternative, however in 2005 these technologies still were immature and expensive. A number of interesting results and figures are provided, proving that *BlueTooth* exhibits a definite byte latency of, on average, 20ms and a maximum of 100ms, these figures were reduced with improved network configuration to an average byte latency below 10ms. A quite interesting "command aggregation algorithm" had also been proposed in this paper.

A surprisingly small amount of white papers have been found on this subject, which indeed makes this study more interesting. Though, there are various articles available on-line, these are listed on the WMI website. There is no significant information to discuss in regards to those articles [41].

A few short reports demonstrating *BlueTooth* implementations were found on CCRMA (Centre for Research in Music and Acoustics at UC Stanford) website [9], these appear to be homework reports by CCRMA students [5, 92]. Other related articles are listed in the bibliography [4, 2] appeared in 2005 and only one recent publication describing another *BlueTooth* controller for *PureData* [89] written by a researcher from Plymouth University [60].

2.3.2 Commercial Wireless MIDI Products

Currently there are four commercial solution available from different manufacturers. It is most likely that the radio communication protocol used by these devices is unique to each particular manufacturer, since no information on compatibility is provided.

- *Kenton MidiStream [39]*
- *M-Audio MidAir [7]*
- *CME WIDI [10]*
- *PatchMan MidiJet [53]*

The *Kenton* devices operate in the two sub-gigahertz ranges, 869MHz and 433Mhz, while the other three manufactures use the 2.4GHz ISM band. It may be desired to evaluate these devices before proposing any solution of a commercial nature; there is no particular interest to obtain any of these product otherwise.

Other Related Hardware

The *MiniBee* project by the Canadian group of artists, *Sense/Stage* [57], is of particular interest, it utilizes high-level frameworks such as *Pure Data*, *Max MSP* and *Super Collider* [59]. The *MiniBee* hardware [58] is an *Atmel AVR* 8-bit MCU and *Digi Xbee* transceiver module, which is not the most optimal design solution. Again, *ZigBee* is a patented technology and is not appropriate for an open-source project. Also, the *Xbee* module itself is already using an MCU (`HCS08`) to run its software stack [17].

In the area of sensor products, there is one interesting company called *Phidgets, Inc.* [54], it manufactures rather wide variety of sensor devices, ranging from base single-board computer (SBC) to sensor and actuators sharing same type of 3-pin connector. The SBC board runs custom-built *Linux OS* on *ARM9* 266MHz CPU with 64MB of SDRAM and 64MB of flash memory. *Phidgets SBC* [55] has USB and 100MB Ethernet peripherals as well as a number of screw terminals and 3-pin connectors for sensors. *Phidgets, Inc.* supplies numerous easy-to-use sensor and actuator devices. The cost of their products is quite high, but the reason of this is the target market. Buyers of these easy-to-use devices are not professional electronics engineers and are ready to pay these prices. Their website also has source code for reference in a wide selection of programming languages and frameworks.

However the top product in the hobbyist marker remains *Arduino* [72], of course the low price factor is key to its popularity. A huge number of semi-clone products emerged, all featuring various additions that some users may find more useful. The most recent interesting semi-clone of *the Arduino* is *Seeeduino Film* [56], made by *Seeed Studio* in China. It appears to be the only product of its class that uses flexible film instead of regular circuit board. For example, this could be quite useful for integration into a music instrument or attached on performer's body in combination with capacitive flex sensors. Even the above mentioned *MiniBee* in its design is also a semi-clone of *the Arduino*, in order to support convenient *Arduino C++ Library* [71], which is in no doubt appreciated by the target audience.

Part II

Design & Development

Chapter 3

Operating Systems for Wireless Sensor Networks

3.1 Overview of Embedded Operating Systems

A subject of extensive research for this project has been into the area of Real-Time Operating Systems for embedded devices, RTOS for short. This family of operating systems is not like any other general purpose OS family. The distinct feature that any RTOS has is the task management in constrained time-scale, the term *real-time* is relatively abstract, it can be classified either as *soft-* or *hard-real-time*. The classification is also an abstract matter, some operating systems provide various approaches and it is up to the engineer to utilise it correctly for any given application. A modern RTOS should also provide a communication protocol stack and file storage drivers, support a set of platforms and a variety of peripheral devices.

It is commonly known that RTOS is a rather expensive element in embedded design. This is because until very recently most commonly used RTOS implementations were provided only by specialised commercial vendors, brand names like *Micrium*, *VxWorks* and perhaps *QNX* are popular in specific application fields and there are also more obscure RTOS products offered at premium prices. Another more recent commercial RTOS is *Nucleus*, currently owned by *Mentor Graphics*. This repor does not focus on commercial products, hence these will not be refer to, more information can be found in [**?**, **?**].

Currently the open sources community does present a number of very interesting RTOS projects, some of which are briefly introduced below. More details can be found on the WMI website [19] (all the general project URLs are listed in that section of the website, and therefore omitted from the text below). It may be important to note that many of the project which are listed here are originating from major universities, some of which previously gave birth to very well know UNIX software and general innovations in the computer science.

RTEMS

The *Real-Time Executive for Multiprocessor Systems* or *RTEMS* is a full featured RTOS that supports a variety of open API and interface standards. It originates from US military and space exploration organisations. By definition *RTEMS* is a highly scalable system, i.e. it is designed for multi-processor deployment and is an interesting system to use for various possible applications, *RTEMS* appears to be the oldest open-source RTOS. Unlike other systems below, the *RTEMS* doesn't belong to sensor network OS sub-family on which this project had been focused.

TinyOS

TinyOS originates from the University of California, Berkeley and Stanford and the Intel Research Laboratory at Berkeley [**?**, 8, **?**]. The approach of *TinyOS* is very different from all other systems, it uses a higher level abstraction language based on *C*, which is called *nesC*. It has a very modular semantics with degredd some similarity to hardware description languages (HDL), such as *VHDL* or *Verilog*. The name *nesC* stands for "Network Embedded Systems C", in analogy with HDL it describes application level connectivity for modules, protocols and drivers. Despite this high level of abstraction, compiled source code is targeted for devices with limited compute resources [74]. The development ecosystem of

TinyOS also includes a network simulation package *TOSSIM* [73], which is a rather trivial element to design when such level of abstraction is in place. At the start of the project *TinyOS* was of some interest, however it had been opted out in preference to *Contiki* for it's more tradition *"plain C"* language abstractions. However, *TinyOS* had been evaluated during the development phase and more details can be found under section 3.3.

Nano-RK

Nano-RK is a fully preemptive reservation-based RTOS, unlike any others in this section it already contained source code for `ATmega128RFA1` chip. The testing of *Nano-RK* had been added to the project agenda. This OS is developed by Carnegie Mellon University and there are various interesting research papers available from the website [51, 3, 6, 1]. Some particular features to mention are an implementation of real-time wireless communication protocol for voice signal transmission [1] and a presence of an audio device driver within *Nano-RK* kernel.

Tracked Issues

Issue #21 : Test Nano-RK

Contiki - "The OS of Things"

"Contiki is an open source, highly portable, multi-tasking operating system for memory-efficient networked embedded systems and wireless sensor networks. Contiki is designed for microcontrollers with small amounts of memory. A typical Contiki configuration is 2kB of RAM and 40kB of ROM.

"Contiki provides IP communication, both for IPv4 and IPv6. (...)

"Many key mechanisms and ideas from Contiki have been widely adopted in the industry. The uIP embedded IP stack, originally released in 2001, is today used by hundreds of companies in systems such as freighter ships, satellites and oil drilling equipment. (...) Contiki's protothreads, first released in 2005, have been used in many different embedded systems, ranging from digital TV decoders to wireless vibration sensors.

"Contiki introduced the idea of using IP communication in low-power sensor networks. This subsequently lead to an IETF standard and the IPSO Aliance (...)

"Contiki is developed by a group of developers from industry and academia lead by Adam Dunkels from the Swedish Institute of Computer Science." [12]

Many publications by Adam Dunkels can be found on the SICS website [25, 76]. He is also an author of two recent textbooks on IP WSN subject [24, 77]. In addition, there is a rich set of simple code examples illustrating basic applications. The source code has well-defined directory hierarchy, build system and programming style [13].

Contiki provides a very simple API for advanced functions, such as real-time task scheduling, *protothread* multitasking, filesystem access, event timers, TCP *protosockets* and other networking features.

More detailed description *Contiki* is provided the following section (3.2).

3.2 Introducing the Contiki Operating System

Contiki operating system can be seen as collection of processes. In the same way as the UNIX operating system is seen as collection of files and processes. However, files are not a necessary component, in fact *Coffee File System (CFS)* is only needed for some specific applications which require a non-volatile storage abstraction. The inter-process communication (IPC) in UNIX has not been implement in the very early releases, although Contiki processes wouldn't work without its event-driven IPC mechanism[1], which also drives the core scheduler. There is no notion of users in *Contiki* however, and at this point comparison with UNIX becomes apparently impossible. Nevertheless, such a high-contrast comparison will hopefully deliver a good picture of what *Contiki OS* is and what it is not.

The lead developer of *Contiki* Adam Dunkels of Swedish Institute of Computer Science (SICS), has revolutionised embedded system world by implementing *lwIP* in 2000 and later, in 2002, even further optimised *uIP* stack. It is known that until *lwIP*, no complete IP stack existed which could run on a microcontroller device [22].

The key technique utilised in implementing *Contiki* and the *uIP* stack is based on a very obscure behaviour which the C language's `switch/case` statement exhibits in certain context. Dunkels wrote a few papers [26, 27] describing how it had been achieved, and summarises it in his doctoral thesis [23]. The threading technique has been named *protothreads (PT)*. Processes in *Contiki* are designed as an extension to the *protothreads*. Another extension is called the *protosockets*, as the name suggest it provides the abstraction of network sockets.

Most of functionality of all three key elements is delivered by the use of C pre-processor macros. The *protothreads* are platform independent[2] and can be integrated into any C program by including only one header file. *Protothreads* can be used with virtually any C compiler and, as described in [27], the overhead is rather insignificant, considering the powerful functionality; the only limitation is due to the use of `switch/case` statements by the *PT* macros, these control structures will lead to undefined behaviour of the code in the *PT* context.

Protosockets are designed specifically for *uIP* and therefore cannot be included with just one header. The only limitation of the *protosockets* is due to design decision, there is no UDP communication facility, only TCP is provided. The *protosockets* greatly simplify the application code, since all necessary functions are provided, including handling of strings. Thought no packet flushing is currently possible, which may be desired in some application.

3.2.1 Initial Development Phase

Contiki source code includes a large set of stable drivers for various RCBs which were mentioned previously, such as *Zolertia Z1*, *Tmote Sky*, *Atmel Raven* and `RF230`*-based* boards, *TI* `CC2430` and *MSP430* as well as *ARM Conrtex-MX* and many older devices, including *Comondore 64* and *Apple II* computers.

[1] *Though it is very primitive comparing to the usual meaning of IPC as an acronym.*

[2] *It is not completely true for Contiki, there are few additions which utilise facilities that some hardware platforms provide.*

Details regarding the *MC1322x* port [16] are to follow in **??**, since *MC1322x* was chosen as the target platform for this project. However, the *ATmega128RFA1* platform had been the first preference. Major work during the initial phase had been done to port the existing driver code for *RF230* transceiver to take the advantage of the single chip device.

A considerably long period of work during the first semester had been towards porting the *Contiki OS* to run on the *ATmega128RFA1* chip. However, if the porting task was to be completed, the time it would have taken may have exceeded the allocated period of one academic year. Therefore it was understood that some reconsideration was required and two major solutions were proposed: either changing the development hardware or trying a different OS with the same hardware.

Problems Encountered while Porting the Driver Code

It had been not a trivial task to compare a set of implementations of the driver code, nevertheless some common fragments of code had been identified. The interim report has included the details of what was achieved at the time [21], though most of work in this area was taken as a preparation to the next phase.

The log of commits made to the source code repository can be viewed at:

https://github.com/contiki/contiki-rf2xx/commits/porting?author=errordeveloper

There had been two main implementations which were compared, although another two components had been looked at while working on this part of the project. ...

After studying the source of these imlementations, it has become apparent that an earlier version of *Atmel RUM* has been already ported to *Contiki* and appears to be the base for the *AT86RF230* driver and the MCU core *ATmega1281*. The codename for that driver is *RF230* and it's modified version is name *RF230BB* (the suffix *"BB"* stand for *bare-bones*). *RF230BB* is generally a stripped version of the first driver, it has been designed to suite up-to-date internals of *Contiki OS*.

A major issue arose, when the upstream *Contiki* code tree has been modified by one of the developers[3], David Kopff, to provide an initial support for the *ATmega128RFA1* chip. Those change were contradicting with what has been done while working on this project. Also it became apparent that no major improvements are going to be made to provide simpler facilities for porting *Contiki* to new *AVR* devices. Nevertheless, during a discussion on the mailing list, that has been agreed upon as generally a good direction for the development of the *AVR* branch, thought had been seen as a remote target. Further study of the source code in the *AVR* branch lead to a decision that a complete refactoring is necessary to provide an appropriate level of abstraction for the new *AVR* device drivers to be designed. Also the separation of radio transceiver drivers, for use with other architecture branches, would be feasible. It is certainly a longer term task to leverage such changes and could not be achieved within the time allocated for this project.

[3] *The commit history of the upstream AVR sub-tree can be viewed at:*

https://github.com/contiki/contiki-mirror/commits/master/cpu/avr

Another reason for reconsidering the goals of the project had been a possibility of encountering bugs in the driver code at some point closer to the project deadline, since the application code was yet to be written. In such situation, it would be difficult to find whether a bug is in the driver code or it is in the application code, hence the best idea is to design the application using a tested platform, where a minimum of low-level code would need to be written. And then, if desired, migrate it to other platforms, once it's all been approved as working correctly.

3.3 Outlook Trial for the TinyOS

It has been discovered that a few members of *TinyOS* community have recently worked on porting *TinyOS* to run on `ATmega128RFA1` [62, 48]. Comparing to *Contiki*, *TinyOS* has rather superior abstraction layer that provides a seamless cross-platform integration methods [*TEP2* [70], *TEP108* [66], *TEP109* [67], *TEP131* [68]].

Various internals of *TinyOS* had been studied, however it is certainly a very broad area to be described here. It is best described in the *"TinyOS Programming"* book by David Gay and Philip Levis (it is available in print as well as a web-edition [44]). Two authors of this book are lead developer of the *TinyOS* project and are currently working at the University of California, Berkley and Standford.

TinyOS currently had been ported to a variety of 8-, 16- and 32-bit microcontrollers, most of the peripherals (such as I2C and SPI) have unified high-level access mechanisms, unlike in *Contiki*[4]. Most of development documentation is provided in the the format of *"TinyOS Enhancement Proposal"* (known as *TEP*[5]), there is also documentation generated from the source code comments (*nescdoc*) as well as various on-line resources [75] and the textbook mentioned above. It is noticeable that *TinyOS* documentation is more extensive then what is currently present for the *Contiki OS*.

3.3.1 Programming with TinyOS: Compilers and Abstractions

The greatest achievement of *TinyOS* is its specialised language, benefits of which had been overlooked at the earlier stage of research for this project. *"Network Embedded Systems C" (nesC)* is a C-based language, it provides abstractions for components, interfaces and configurations. It is not an object-oriented language, though it is rather component and interface oriented. As it was defined in the previous section that *Contiki* can be seen as a collection of processes and events are communicated between those processes, then *TinyOS* can be defined as a collection of components and interfaces which are wired together in configuration abstraction layer, tasks, events and the data are communicated between the components via the interfaces.

The control structures and all of C constructs are still valid in *nesC*. This can be seen as if *nesC*-specific statements are replaced by appropriate C source code that defines required behaviour. Nevertheless, this is only a brief description of the relation between *nesC* and its ancestor. Just as if C could be defined by stating a ratio of lines of code in C versus lines of assembly code.

It is important to note that current implementation of the compiler translates higher-level nesC code into C language. The resulting programs are specifically suited for embedded devices. Direct compilation would be possible and, if desired, there is an interesting platform to look into. *LLVM* ("Low-level Virtual Machine") is a new generation compiler technology [87]. It is know that using *LLVM* and its family member *clang* implementation of a new compiler

[4] *Most of peripheral drivers in Contiki OS are rather specific to a certain platform and only some particular drives share a similar interface.*

[5] *These documents are commonly referred to as TEP**n**, where **n** is a number.* TEP1 *[69] defines the format of the TEP documents.*

for a C-like language would be simplified (comparing to more traditional techniques). One good example of industrial grade compiler based on *LLVM* is the *XC* toolchain for *XS-1* devices from a UK semiconductor company *XMOS* [78].

Nevertheless, currently *nesC* compiler is known to be fully functional and it's task is not as complex as it may seem. Programming in any language is always done by applying code patterns, general to some degree, to accomplish desired behaviour of a program. The purpose of *nesC* abstraction layer can be seen as making the details of complex coding - with which high degree of modularity can be achieved - rather hidden away from the programmer.

3.3.2 Network Protocols in TinyOS

TinyOS has become widely adopted and there are many researchers who contributed significant work in different application areas, one such area of great interest to WMI project is sensor node timer synchronisation protocols. A number of papers are available and the mainstream repository of *TinyOS* source code already contains implementations for some of those protocols.

(A few details with references needed)

Further study and experimentation in this area are required to design a system which could cope with real-time constraints of stage control applications. It is important to note that *P802.15.4* has addressed some real-time application requirements, however the status of software support for these features of *P802.15.4* is currently uncertain. *(Perhaps mention NanoRX ?)*

After several aspects of *TinyOS* were studied, it was clear that its current implementation of *6loWPAN* is fully compatible with *Contiki* and it had been desired to prove this in practice, but this has not been achieved at the time of writing of this paper.

A problem exists however, there was no fully working IP-enabled driver for *ATmega128RFA1* transceiver. The IP layers are provided by *BLIP* stack (this stands for Berkley Lightweight IP), the stack is currently undergoing major development. Details on what is required are available on *TinyOS* wiki page [14], implementing it in *nesC* was not as trivial due to the learning curve. Therefore this had be postponed.

This problem requires further explanation. *TinyOS* has been developed since 2001, while *Contiki* first dates back to two years later - 2003[6]. In the early days of WSN research *P802.15.4* and *ZigBee* where still emerging, the *6loWPAN* RFCs appeared at IETF more recently[7]. *TinyOS* has originally used its own protocol called *ActiveMessage*.

Currently *TinyOS* incorporates *BLIP* stack, which initially was released by UC Berkley Wireless Embedded Systems research group in the summer 2008, know as *bl6lowpan* at that time [**?**]. As mentioned below, major code changes are currently still in progress. The driver code which fully implements all new features of the *BLIP* stack (including point-to-point tunnelling for simple UART wired connectivity) is only for the most popular `CC2420` radio chips.

[6] *No exact information has been found, these are the earliest dates which appear in the source code.*

[7] *The first revision of P802.15.4 was released in 2003 (current revision is from 2006) and first draft of 6loWPAN is dated April 2007.*

3.3.3 Hardware Resources

It is difficult to compare the two sets of hardware platforms which *TinyOS* and *Contiki* had been ported to, since the status of support for some platforms is uncertain. It is probably most appropriate to say that these two sets are almost equal, however, some platforms which appear in *Contiki* source code, had not been ported to *TinyOS* and vise versa. Two devices of interest to WMI project are `ATmega128RFA1` and `MC1322x`. The issue with the first chip is already described above. The second chip has not been ported to *TinyOS* yet. This should not be too difficult task to achieve considering that driver implementation could be derived from *Contiki* code. There also very noticeable progress towards *ARM Cortex* support[8], hence there is base for *ARM* devices (i.e. toolchain support and core architecture code).

There is a large repository of board design files featuring so called *Epic Mote* platform, that is base around `CC2420` and the `MSP430` [**?**]. The most interesting designed by Prabal Duta of UC Berkley [**?**] is shown in Figure **??**. The reason why it is so interesting is because it features *Digi Connect-ME* microprocessor system block that is of standard RJ45 form-factor [**?**]. This tiny device has 55MHz `ARM7TDMI` CPU, 4MB of flash and 8MB of SDRAM as well as hardware cryptographic unit and is capable to run Linux kernel and a small subset of standard software in userspace, hence the advantage of scripting languages can be utilised. Currently it is the smallest device available on the market featuring such capabilities and the unique form-factor. It is a very appropriate device to use for wireless network edge gateway system, for example a hardware crypto unit could accelerate secure tunneling of the network traffic. It could be considered as a better solution instead of using a large Ethernet chip, unless there would be a chip with a combination of Ethernet and RF hardware in a single package.

3.3.4 Conclusion

Several steps were made in attempt to bring up *TinyOS* on the hardware chosen for this project earlier, though it appeared rather unmanageable within the give time frame. This is still a subject of interest and shall be looked into at a later time. An evidence of work carried out with *TinyOS* code can be viewed on WMI website issue tracker.

Tracked Issues

Issue #30 : Porting Radio Interface to BLIP 2.0

[8] *The homepage of TinyOS Cortex project can be found at:*

`http://code.google.com/p/tinyos-cortex/`

The code structure was found to be well organised and most probably can be relied upon. However it has not been desired to take this challenge in the near future.

Chapter 4

DSP Host System

4.1 DSP Host Hardware

A set of hardware platforms suitable for DSP host were considered. Among those are the following:

- *Analog Devices SHARC*
- *ARM:*
  - *Marvell Sheeva*
  - *Texas Instruments OMAP*
  - *Freescale i.MX*

All of these processor architectures are fully supported and widely used in embedded systems. Various other architecture families were considered, but found rather inappropriate due to the price range dictated by the target of this project. Multi-core *MISP64* devices by *NetLogic* [**?**], for example, have great computational capacity, thought these are too expensive for this particular application.

4.1.1 x86-based Embedded Devices

The list above does not include *x86*-based CPUs due to the fact that it had been very difficult to find a target board using Intel or any *x86* CPU from other vendors. There are many boards from a large variety of manufacturers, hence the selection process becomes particularly time consuming. Quality evaluation would also be necessary[1], when looking at *x86* devices. Although, there are boards that would match the low-power and small form-factor requirements, most of these are with a handful of peripherals, for example low quality audio and video interfaces and, if these are not present, the board may have two or more RS-232 connectors. Another aspect of *x86* system design is that it comes with legacy *BIOS* technology, while most of *ARM* systems, for example, have rather more flexible facilities for booloader and set-up. Also in the class of *single board computers (SBC)* there are many boards which do not inlude physical connectors for above mentioned peripherals, however isuch boards are designed for industrial use require a specialised chasis [**?**].

Nevertheless, an *x86* machine has been used for most of the development work on this project, that is for a rather transparent reason.

4.1.2 OMAP

The *OMAP* application processors form *TI* has become popular in the open-source community, and in fact *TI* promotes Linux and Android as most suitable operating systems for this platform. The *OMAP* architecture is based around an *ARM* processor (there are various models with different versions of *ARM* core) and *TI C64x* DSP block. However, there is a little problem associated with how the DSP unit is integrated with the CPU. Very limited documentation is provided on how it can be used in the *Linux* environment [**?**].

One most outstanding development platform that uses an *OMAP* chips with dual-core *Cortex-A8* CPU clocked at 1GHz, is the *PandaBoard* [**?**], however it

[1] *Due to the scale of production in this market, there is a great chance to obtain a device with a failure.*

is available for back-order only, the maker is producing these boards on demand and the lead time would be at least one month. Otherwise this board would have been purchased despite the fact that DSP unit would be difficult to utilised.

4.1.3 Other ARM CPUs

ARM CPUs are produced by almost any major semiconductor firm, so do *Freescale* and *Marvell*. Not all of those companies make very high performance *ARM* chips, i.e. clocked near to 1GHz, and few make multi-core CPUs. And only some CPUs have an FPU. *ARM* floating point instruction set is know as *VFP* ("Vector Floating Point"). There are different versions of *VFP*, though in this context this wasn't a concern.

Marvell Sheeva

Marvell has originally started offering *ARM* processors since their purchase of Intel's *PXA* devision. Now *Marvell* produces a series of high-performance *ARM* chips, most of which are multi-core. *Sheeva* is the brand name for these SoC devices. The clock frequency ranges from 800 MHz to 1.6 GHz, gigabit Ethernet is one of the outstanding features, since *Marvell* specialises in the networking and storage IC market. As mentioned above, most important feature of *ARM* chips that is necessary for the design of DSP host for this project is *VFP*. Some of *ARMADA* SoCs include *VFP*, namely *ARMADA XP* series, *ARMADA 510* and *610* [**?**]. *Kirkwood* series are not featuring *VFP*, thought there are some outstanding embedded development platforms available.

Plug Commputer (also know as *Seeva Plug*) is small, yet very powerful computer in a form-factor of wall socket DC adaptor [**?**, **?**]. There many exciting application where these devices could be the best fit, thought due to above mentioned lack of *VFP*, this CPU is not well suited as the DSP host.

Freescale i.MX

4.1.4 SHARC

The DSP instruction can be used directly and a library is provided which . . . blah . . . blah.

300MHz - 600MHz .. there dual-core Blackfin chips ..

Analog Device SHARC development hardware is mostly available from the vendor and very few third-party board could be found. Since this hardware architecture is more specialised, the prices of the development boards are not as low as some of the *ARM* boards. Although the vendor offers a comprehensive free library of board design file [**?**].

SRARC processor are commonly used in professional audio equipment. One very outstanding product has been released by a new British company *Dark Matter Audio*, the device called *DMA1* [**?**] has been released this year. It is an advanced system for interactive music performance with very novel design features and powerful *SHARC* DSP engine. According to the product announcement it runs Linux OS and also uses an *ARM* chip for graphical user interface. It would be possible to integrate *DMA1* with current prototype system developed in the course of this project. The vendor has provided an SDK, thought

a request has been sent to obtain more specific details needed to design this add-on solution appropriately.

4.2 Building Embedded Linux OS

4.2.1 Background

Until quite recently, it had been rather more difficult to achieve the task of building (custom) embedded Linux system. Traditionally, the engineer who desired to do so, would need to follow instructions provided in the reference book *"Linux from Scratch"*, commonly known as *LFS* [**?**]. This text provides details on how to utilise various tools for building an embedded Linux kernel and the file system from source, it discusses how to tweak various features at build time and configure appropriate runtime services.

4.2.2 Build Utilities

More recently, a number of projects emerged, which do a great job of extending the flexibility by automating some of the simpler tasks, e.g. package dependency tracking and build version control (enabling the maintainer to revert to previous builds). One of the most outstanding and widely used projects in this areas is *OpenEmbedded* [**?**] it has recently been adopted by a major software company *Mentor Graphics*. *Mentor Embedded Linux* extends *OpenEmbedded* framework with a set of tools which are useful in a large-scale development project [**?**].

There are a few alternative approaches which lead to similar results, one may wish to use *Gentoo Linux* meta-distribution, there is only one source of documentation - *"Gentoo Linux Embedded Handbook"* [**?**]. Generally, *Gentoo* framework (named *Portage*) has all necessary components which a system designer may need and there is no big difference between *Gentoo* and *OpenEmbedded*. Many internals are known to be very similar, tough *Gentoo* doesn't provide some specialised tools which are provided in *OpenEmbedded*, hence *Gentoo* has been originally designed for custom desktop and server systems. Major aspect which is different is that *Gentoo* is rather bound to package releases, while *OpenEmbedded* is more flexible when a development revision number has to be specified for a certain source code package. The conclusion is that *OpenEmbedded* is most likely to be more convenient to use for an embedded target.

A little different approach can be taken with *BuildRoot* [**?**], which is a build system for embedded Linux. Based around the same framework that was design for configuring the Linux kernel builds. This tool is a set of scripts controlled via a console menu. Although, it appears to have a simple to use front-end, the underlying configuration system is certainly behind the competition with *OpenEmbedded*.

A project which have a slightly different orientation is *ScratchBox* [**?**]. It provides a cross-compilation toolkit for application developers. It can be used to provide an software development kit (SDK) for a 3rd-party developer with appropriate toolchain and hardware emulator. However, *OpenEmbedded* includes support for building SDK and *ScratchBox* is rather limiting in various ways, i.e. it is a static distribution, rather then a build system.

Another project in this area of interest is *Linaro Foundation* [**?**], the initiative has been originated by London-based *Canonical Limited* (the company behind now most popular Linux distribution - *Ubuntu*). The purpose of *Linaro* is to improve various issues related to embedded Linux and provide a higher grade platform for Android and other multimedia and consumer oriented distributions.

Linaro is still an emerging project and has not produced any significant output [**?**]. It has specific orientation, in terms of hardware, being currently involved only with *ARM* platforms, and in terms software, it is bound rather strictly to *Ubuntu* methodology.

4.2.3 Conclusion

It had been desired to build an embedded DSP host OS as one of additional project targets, however due to various[2] limitations, most importantly - the time frame, the project has not achieved this at the time of writing of the final report. The above section was included to provide the evidence of research in this area.

[2] *Another aspect was due to hardware selection and purchasing issues.*

Chapter 5

Application Development

5.1 Working with Contiki

After an appropriate compiler toolchain and debugger packages for *MC1322x* had been installed on the development host, a few steps were taken to simplify the work-flow in *Contiki* application development environment. It may be noted here, that an integrated development environment (IDE) could be used and some programmers do prefer to use an IDE, such as *Eclipse* [**?**], nevertheless the command line tools are known to be the most efficient approach. It should be noted that this section is rather brief description of what has been done and was not intended to provide a detailed guidance on how to reproduce the results.

Apart from the revision control tools[1] and the GCC toolchain for ARM [**?**], there are four key command-line tools which were utilised during the development process.

gdb [**?**] - GNU Debugger (source-level) [**?**]

OpenOCD - On-chip Debugger [**?**]

Vim - a text editor

make [**?**] - GNU make program [**?**]

To enhance the work-flow *"Makefiles"*[2] were amended throughout the development process. Generally there is one *Makefile* in each subdirectory of the source tree, thought most of these inherit rules specified in the main *Makefile* (in *Contiki* there are two of these - one at in the root directory and one for each processor architecture).

Most of the changes were made in `cpu/mc1322x/Makefile.mc1322x` to provide a variety of short command for connecting the debugger and sending the program to run on a development board[3]

5.2 Application Prerequisites

The first step in application development was to add a driver for the second UART (UART2) port. This has been done by copying an existing driver code for UART1, though enhancements were required at a later stage. The history of changes to the code can be viewed in at the repository by utilising the commit log filter. The files shown here had been modified.

- `cpu/mc1322x/lib/include/uart1.h`
  normal and weak prototypes, register pointers and macros
- `cpu/mc1322x/lib/uart2.c`
  driver interrupt handlers

[1] *This project used git system, however the details on how that has been done are considered to be irrelevant to the subject of this report*

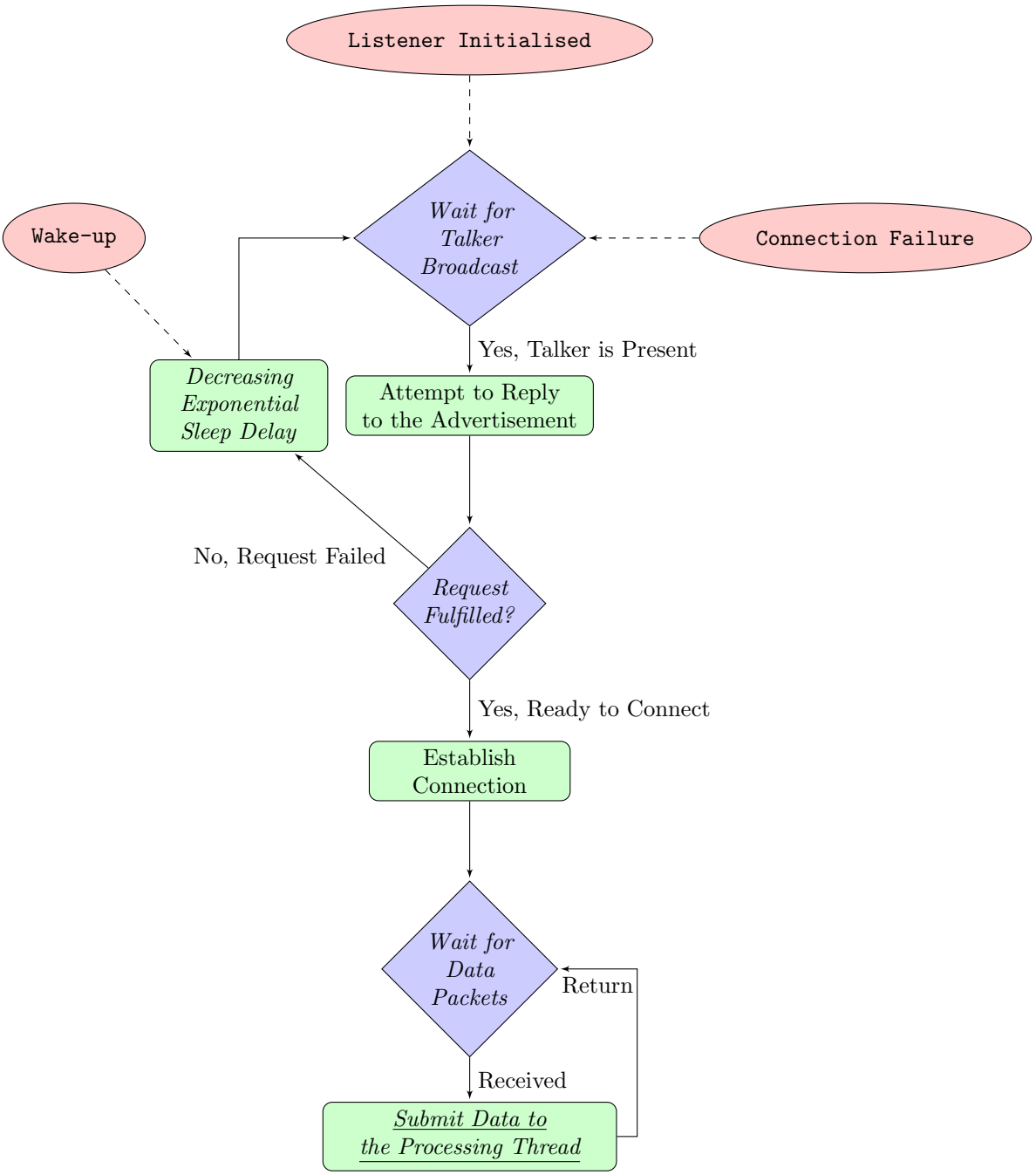
[2] *These are the file which specify a set of rules for the make program on how to compile the source code and also perform administrative tasks and run debugger or other tools.*

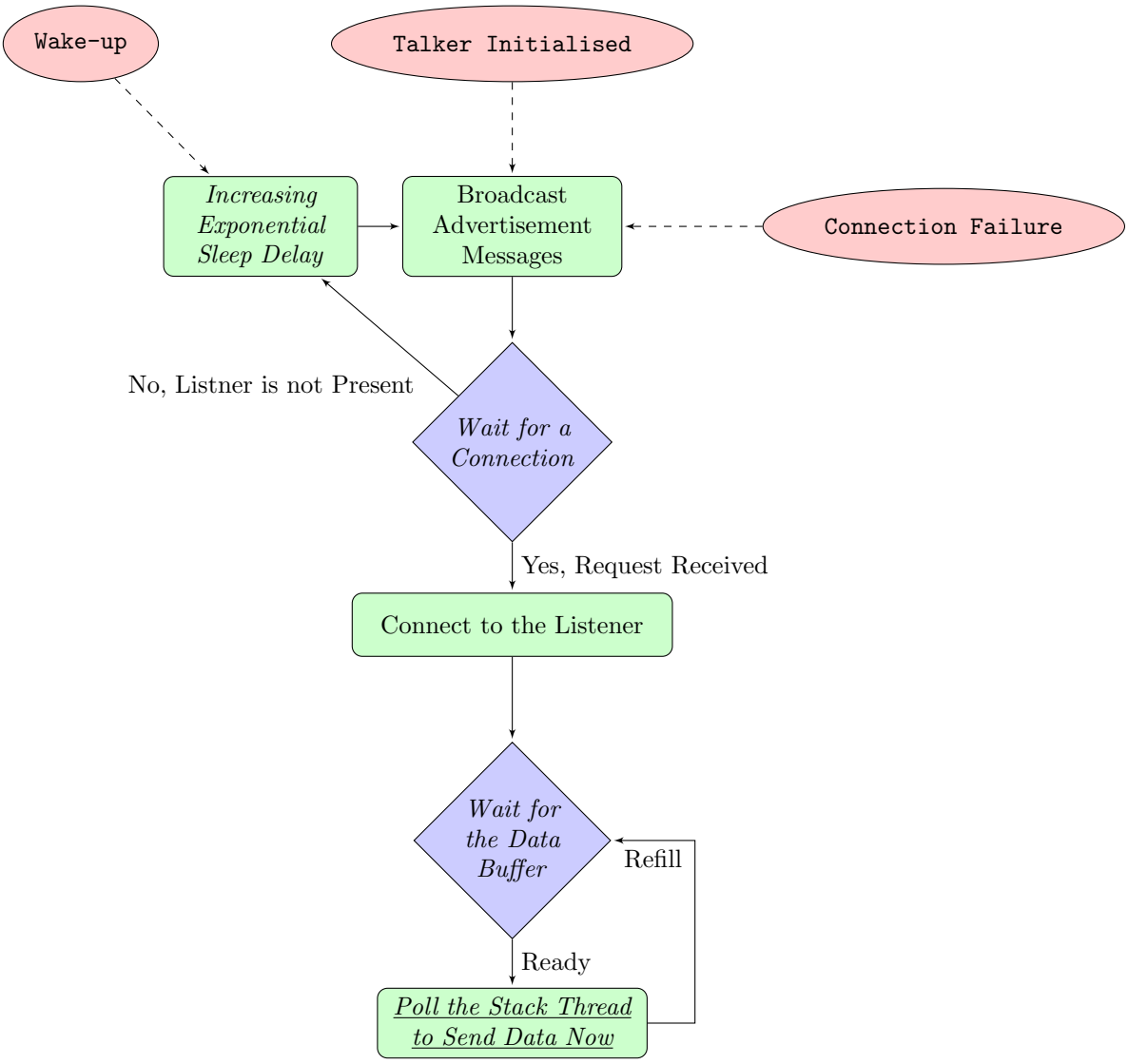
[3] *For example, to set-up the WPAN router on the Freescale board - run 'make.f1 router' and to compile 'example.c', load and print serial output on the console for the Econotag - 'make.e1 example.load-print'; in case if 'example.c' does not behave as expected - run 'make.e1 example.ocd-screen'.*

- `cpu/mc1322x/src/default_lowlevel.h`
  prototypes
- `cpu/mc1322x/src/default_lowlevel.c`
  initialisation functions

Chapter 6

State Machines


Listener Initialised
Wait for Talker Broadcast
Wake-up
Connection Failure
Yes, Talker is Present
Decreasing Exponential Sleep Delay
Attempt to Reply to the Advertisement
No, Request Failed
Request Fulfilled?
Yes, Ready to Connect
Establish Connection
Wait for Data Packets
Return
Received
Submit Data to the Processing Thread


Wake-up
Talker Initialised
Increasing Exponential Sleep Delay
Broadcast Advertisement Messages
Connection Failure
No, Listner is not Present
Wait for a Connection
Yes, Request Received
Connect to the Listener
Wait for the Data Buffer
Refill
Ready
Poll the Stack Thread to Send Data Now

Chapter 7

System View

7.1 Key System Components: Specification & Requirements

This section outlines the specification which had been proposed after initial research phase has been accomplished. Diagram shown in figure 7.1 illustrates the function blocks of the complete system that meets this specification, however it is now considered suitable only for the development system and not the final product design.

Sensor Node

Sensor Node is a device featuring a microcontroller chip that contains following function blocks and peripherals:

- *internal:*
  - 2.4Ghz RF transceiver,
  - A/D converters,
  - GPIO,
  - SPI, and
  - USART
- *external:*
  - digital or analogue sensors,
  - external connectors for MIDI,
  - wired remote sensors, and
  - serial port header

The *Sensor Node* runs software which consists of:

- *operating system:*
  - communication protocol stack,
  - devices drivers, and
  - application task management,
  - service tasks, and
  - the main program
- *bootloader:*
  - loads new software

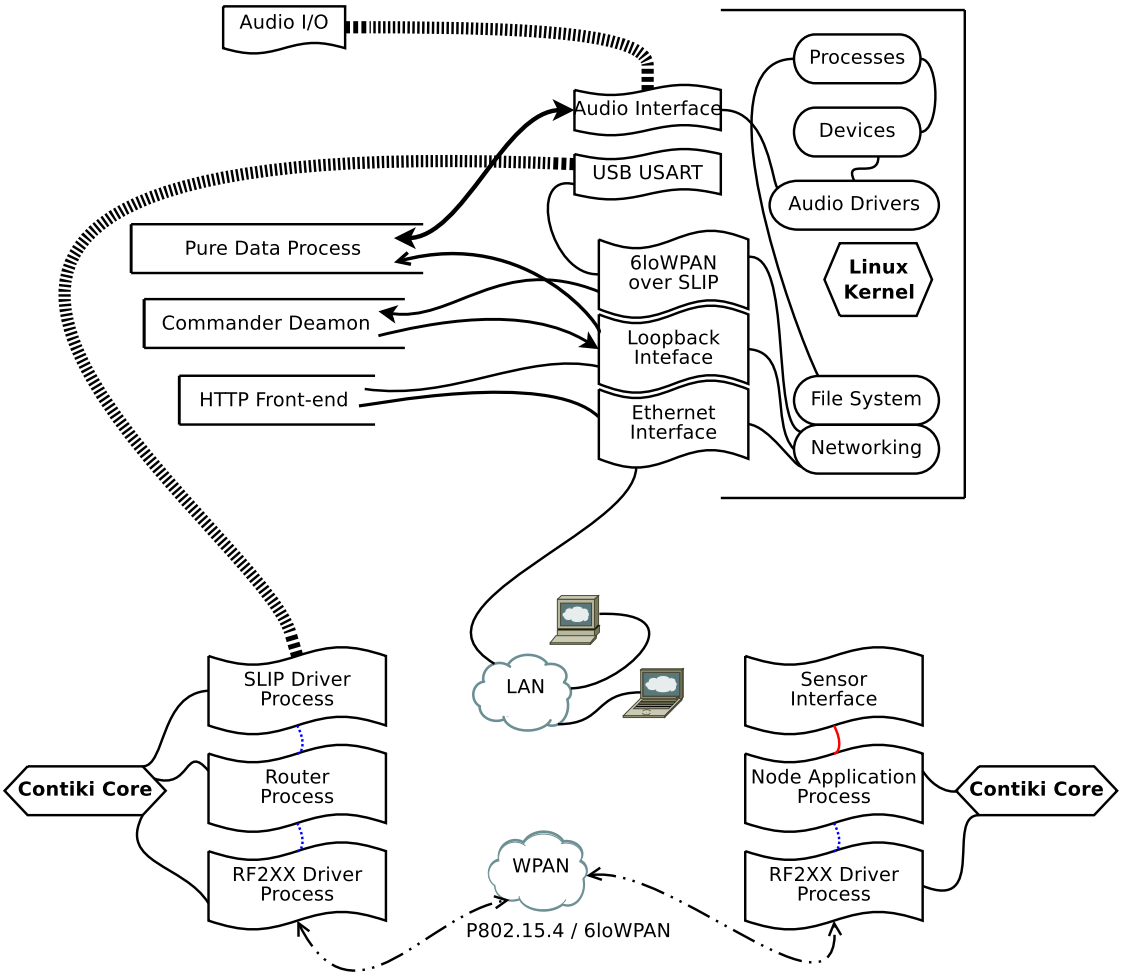
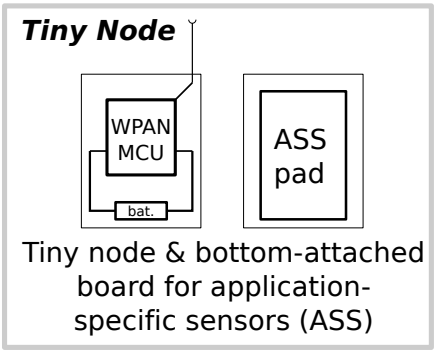
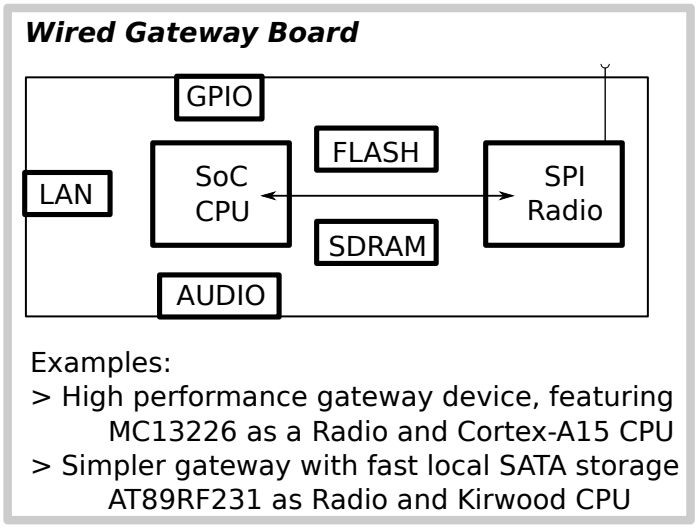


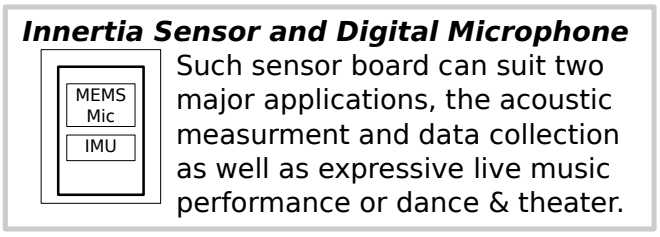
Figure 7.1: *Abstract Sketch of the Development System*

7.2 Final Product Design

The block diagrams in figure 7.2 briefly illustrate a subset of a product line suitable for a commercial solution. Some of the device proposed here can be used in a variety of applications and others are rather specific for the applications mention in the first part of this report.







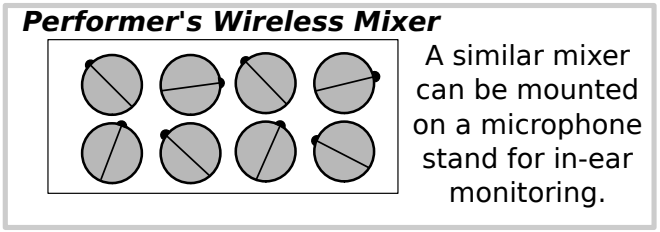


Figure 7.2: *TEST*

Bibliography

[1] Anthony Rowe Raj Rajkumar Ryohei Suzuki, R. M. Voice over Sensor Networks, 2006.

[2] Damondrick Jack, R. L. Beat Boxing: Expressive Control for Electronic Music Performance and Musical Applications.

[3] Dhiraj Goel Raj Rajkumar, A. R. FireFly Mosaic: A Vision-Enabled Wireless Sensor Networking System, December 2007. Real-Time Systems Symposium (RTSS).

[4] Jennifer Carlile, J. F. SoniMime: Movement Sonifcation for Real-Time Timbre Shaping.

[5] John R. McCarty Jeffrey Traer Bernstein, P. S. Muggling! A Wireless Spherical Multi-Axis Musical Controller.

[6] Rahul Mangharam Raj Rajkumar, A. R. FireFly: A Time Synchronized Real-Time Sensor Networking Platform, November 2006. Wireless Ad Hoc Networking: Personal-Area, Local-Area, and the Sensory-Area Networks, CRC Press (Book Chapter).

[7] AVID/M-Audio. Products: MidAir.

[8] Berkeley Wireless Embedded Systems (WEBS). Project Homepage.

[9] CCRMA. Homepage.

[10] CME. Products: WiDi.

[11] CNMAT. Homepage.

[12] Contiki. Homepage at the Swedish Institute of Computer Science (SICS).

[13] Contiki. Conding Conventions.

[14] Dawson-Haggerty, S. BLIP 2.0 Platform Support Guide.

[15] de Campos Bartolomeu, P. J. Evaluating Bluetooth for the wireless transmission of MIDI. Master's thesis, Universidade de Aveiro, Departamento de Electrnica e Telecomunicaes, 2005. *IEEE paper reference 10.1109/ETFA.2005.1612507.*

[16] devl.org. Contiki `MC1322x` Port (Homepage).

[17] Digi International, Inc. Getting Started Guide to XBee Programming.

[18] Dmitrichenko, I. WMI Project Wiki: Bibliography.

[19] Dmitrichenko, I. WMI Project Wiki: Embedded Operating Systems.

[20] Dmitrichenko, I. WMI Project Wiki: On-Line Refences.

[21] Dmitrichenko, I. Wireles music instruments project (interim report), 2010.

[22] Dunkels, A. Full TCP/IP for 8 Bit Architectures. In *Proceedings of the First ACM/Usenix International Conference on Mobile Systems, Applications and Services (MobiSys 2003)* (San Francisco, May 2003).

[23] Dunkels, A. *Programming Memory-Constrained Networked Embedded Systems*. PhD thesis, Swedish Institute of Computer Science, Feb. 2007.

[24] Dunkels, A. *The Wiley Encyclopedia of Computer Science and Engineering*, vol. 4. Hoboken, NJ, USA, Jan. 2009, ch. Operating Systems for Wireless Embedded Devices, pp. 2039–2045.

[25] Dunkels, A., Grnvall, B., and Voigt, T. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)* (Tampa, Florida, USA, Nov. 2004).

[26] Dunkels, A., and Schmidt, O. Protothreads - lightweight stackless threads in c. Tech. Rep. T2005:05, SICS – Swedish Institute of Computer Science, Mar. 2005.

[27] Dunkels, A., Schmidt, O., Voigt, T., and Ali, M. Protothreads: Simplifying event-driven programming of memory-constrained embedded systems. In *Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys 2006)* (Boulder, Colorado, USA, Nov. 2006).

[28] Dutta, P., Dawson-Haggerty, S., Chen, Y., Liang, C.-J., and Terzis, A. Design and Evaluation of a Versatile and Efficient Receiver-Initiated Link Layer for Low-Power Wireless. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys'10)*.

[29] Farahani, S. *ZigBee Wireless Networks and Transceivers*. Newnes, 2008.

[30] Freed, A., and Schmeder, A. Features and Future of Open Sound Control version 1.1.

[31] Freed, A., and Wright, M. Open Sound Control: A New Protocol for Communicating with Sound Synthesizers.

[32] Hope Microelectronics. Product Range: 2.4GHz Transceivers.

[33] IEEE 802.15 Wireless Personal Area Networks. Standard Homepage.

[34] IEEE 802.15 WPAN Task Group 4. TG4 Homepage.

[35] ISPO Alliance. Homepage.

[36] ISPO Alliance. Incorporating IEEE 802.15.4 into the IP architecture.

[37] ISPO Alliance. IP for Smart Objects. *Original version September 2008.*

[38] JazzMutant. Products: Lemur.

[39] Kenton Electronics, UK. Products: MidiStream.

[40] Keunsol, K., Seung-Woo, L., Dae-geun, P., and Bhum Cheol, L. Ptp interworking 802.15.4 using 6lowpan. In *11th International Conference on Advanced Communication Technology* (February 2009).

[41] Ladyada.net. Tutorial: Xbee Midi Link.

[42] Lenzen, C., Locher, T., Sommer, P., and Wattenhofer, R. Clock Synchronization: Open Problems in Theory and Practice. In *36th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM), pindlerv Mln, Czech Republic* (January 2010).

[43] Lenzen, C., Sommer, P., and Wattenhofer, R. Optimal Clock Synchronization in Networks. In *7th ACM Conference on Embedded Networked Sensor Systems (SenSys), Berkeley, California, USA* (November 2009).

[44] Levis, P., and Gay, D. *TinyOS Programming.* Cambridge University Press, 2009.

[45] Levis, P., Tavakoli, A., and Dawson-Haggerty, S. Overview of Existing Routing Protocols for Low Power and Lossy Networks. In *Internet Drafts* (October 2009), IETF.

[46] Livid. Products: Block64.

[47] Ma, Y., and Wobschall, D. Synchronization of Wireless Sensor Networks Using a Modified IEEE 1588 Protocol.

[48] Maroti, M., and Biro, A. Source Code Repository of ATmega128RFA1 port for TinyOS.

[49] MONOME. Details of OSC implementation in MONOME hardware.

[50] MONOME. Homepage.

[51] Nano-RK. Research Publication.

[52] Nordic Semiconductors. Product Range: 2.4GHz Transcievers.

[53] PatchMan Music. Products: MidiJet.

[54] Phidgets, Inc. Homepage.

[55] Phidgets, Inc. Products: SBC.

[56] Seeed Studio. Products: Seeeduino Film.

[57] Sense/Stage. Homepage.

[58] Sense/Stage. MiniBee Hardware.

[59] Sense/Stage. Software Clients.

[60] Sjuve, E. Prototype GO: Wireless Controller for Pure Data.

[61] Solis, R., Borkar, V., and Kumar, P. A New Distributed Time Synchronization Protocol for Multihop Wireless Networks. In *45th IEEE Conference on Decision and Control* (December 2006).

[62] Sommer, P. Source Code Repository of ATmega128RFA1 port for TinyOS.

[63] Sommer, P., and Wattenhofer, R. Symmetric Clock Synchronization in Sensor Networks. In *ACM Workshop on Real-World Wireless Sensor Networks (REALWSN), Glasgow, Scotland* (April 2008).

[64] Sommer, P., and Wattenhofer, R. Gradient Clock Synchronization in Wireless Sensor Networks. In *8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), San Francisco, USA* (April 2009).

[65] Sundararaman, B., Buy, U., and Kshemkalyani, A. D. Clock synchronization for wireless sensor networks: a survey.

[66] TEP Document. Resource Arbitration .

[67] TEP Document. Sensors and Sensor Boards .

[68] TEP Document. Creating a New Platform for TinyOS 2.x .

[69] TEP Document. TEP Structure and Keywords .

[70] TEP Document. Hardware Abstraction Architecture .

[71] Tinker.it. Arduino Library.

[72] Tinker.it. Arduino Homepage.

[73] TinyOS. Network Simulator TOSSIM.

[74] TinyOS. nesC Language Project Homepage.

[75] TinyOS Wiki. Documentation Resources.

[76] Tsiftes, N., Dunkels, A., He, Z., and Voigt, T. Enabling Large-Scale Storage in Sensor Networks with the Coffee File System. In *Proceedings of the 8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2009)* (San Francisco, USA, Apr. 2009).

[77] Vasseur, J.-P., and Dunkels, A. *Interconnecting Smart Objects with IP - The Next Internet*. Morgan Kaufmann, 2010.

[78] Watt, D., and Osborne, R. XMOS Tools Developer Guide.

[79] Wikipedia. 6loWPAN.

[80] Wikipedia. ANT Network.

[81] Wikipedia. Carrier Sense Multiple Access with Collision Avoidance.

[82] Wikipedia. DASH7.

[83] Wikipedia. Direct-Sequence Spread Spectrum.

[84] Wikipedia. IEEE 802.15 - Wireless PAN (Personal Area Network).

[85] Wikipedia. IEEE 805.15.1 (Medium-Rate WPAN) / BlueTooth.

[86] Wikipedia. IEEE 802.15.4 (Low-Rate WPAN).

[87] Wikipedia. Low Level Virtual Machine.

[88] Wikipedia. Quadrature Phase-Shift Keying.

[89] Wikipedia. Pure Data.

[90] Wikipedia. Representational State Transfer.

[91] Wikipedia. ZigBee.

[92] Yeo, W. S. The Bluetooth Radio Ball Interface (BRBI): A Wireless Interface for Music/Sound Control And Motion Sonification.