WMI: Wireless Music Instruments
BSc Final Year Project
Final Report

Ilya Dmitrichenko

Department of Computing
London Metropolitan University

May 4, 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 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 the technology is becoming increasingly popular among the artists and, to some surprise, there are individuals who attempt to introduce a non-trivial aspect 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 [85] 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" attracts the attention in this sentence, hence if a system was made such that an artist could apply for their performance (or display) in a flexible, robust and reliable manner, there is, probably, a niche market for it. Although, Ryan Jordan has found an inspiration in the actual wiring of his system, nevertheless wireless would provide a different 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

The report had been divided into two parts. Part I introduces some general concepts and gives a summary of the initial phase of the research. Part II contains all the details related to development process, presents problems and discusses various related solutions.

There are several chapters in Part II, these were organised in the logical order. The best effort had been made to clearly describe all necessary details and progress towards the final implementation chapter, without any degree of obscurity. It is, however, up to the reader whether some references should be followed and analysed. A few assumptions had been made with the hope that the reader has certain background to follow the ideas which are being presented.

It is important to note, that most of the software which had been used and/or written throughout the project, is located in the on-line repository (see section 1.2.2). That also provides the logging facility, hence no handwritten log book will be submitted with the report. However, a supplementary disk is being submitted as a form of evidence. Some of the data on the disk is a subject to the terms and conditions of a respective copyright licenses, those components were included with the believe that the disk serves the purpose of a personal back-up copy which is not a distribution media.

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 following form (replace the ***'#'*** symbol with the given number):

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

The base URL to access any references to source code is:

https://github.com/errordeveloper/contiki-wmi/blob/wmi-work/

For example, to see the file `README` this URL should be used:

https://github.com/errordeveloper/contiki-wmi/blob/wmi-work/README

Similarly, to view the commit log for a certain file use:

https://github.com/errordeveloper/contiki-wmi/commits/wmi-work/

[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.

<table>
<tr><th>OSI model</th><th>IP model</th></tr>
<tr><td>Application</td><td rowspan="2">Application</td></tr>
<tr><td>Presentation</td></tr>
<tr><td>Session</td><td rowspan="2">TCP, UDP, ...</td></tr>
<tr><td>Transport</td></tr>
<tr><td>Network</td><td>IP</td></tr>
<tr><td>Data Link</td><td>Data Link</td></tr>
<tr><td>Physical</td><td>Physical</td></tr>
</table>

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 [65], 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 [26]. It was first presented in 1997 [66] 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 [98]*
- *Jazz Mutant Lemur [84]*
- *Monome [106, 105]*

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 [160]. *ANT* is a proprietary standard which uses 2.4GHz band [157]. 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 [162].*

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

6loWPAN [156]. *RF4CE* belongs to the *ZigBee* [173] family together with a number of other application area specific variations. *6loWPAN* [156] 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* [163] 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 [162, 164, 77].

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) [64]

"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 [168]

DSSS - Direct-Sequence Spread Spectrum (was used in 2003 revision) [161]

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) [64]

"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[78].

- 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 [158].
- 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 [63] (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) [95] are subject to very extensive research at present, as well as time synchronisation [137, 87, 99, 130, 92, 93, 133, 132]. 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 *Internet Protocol for Smart Objects* (IPSO) alliance [80]. *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 [81, 82].

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*) [171] 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 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* [113] or *Hope Microelectronics* [76]. 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" [47] and "Bibliography" [44]. Below only the most relevant information is discussed.

2.3.1 Publications

One publication by Paulo Jorge Bartolomeu (Univ. of Aveiro, 2005) [33] 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 [88].

A few short reports demonstrating *BlueTooth* implementations were found on CCRMA (Centre for Research in Music and Acoustics at UC Stanford) website [24], these appear to be homework reports by CCRMA students [9, 174]. Other related articles are listed in the bibliography [8, 6] appeared in 2005 and only one recent publication describing another *BlueTooth* controller for *Pure-Data* [169] written by a researcher from Plymouth University [129].

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 [86]*
- *M-Audio MidAir [19]*
- *CME WIDI [25]*
- *PatchMan MidiJet [115]*

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* [126], is of particular interest, it utilizes high-level frameworks such as *Pure Data*, *Max MSP* and *Super Collider* [128]. The *MiniBee* hardware [127] 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 [41].

In the area of sensor products, there is one interesting company called *Phidgets, Inc.* [116], 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* [117] 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* [148], 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* [125], 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* [147], 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 [170, 165].

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 [46] (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 [172, 22, 152]. 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 [150]. The development ecosystem of *TinyOS* also includes a network simulation package *TOSSIM* [149], 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 [109, 7, 10, 5, 123]. Some particular features to mention are an implementation of real-time wireless communication protocol for voice signal transmission [5] and a presence of an audio device driver within *Nano-RK* kernel.

Tracked Issues

Issue #21 : Test Nano-RK on *ATmega128RFA1*

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." [27]

Many publications by Adam Dunkels can be found on the SICS website [58, 153]. He is also an author of two recent textbooks on IP WSN subject [56, 154]. 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 [28].

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 [54].

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 [59, 60] describing how it had been achieved, and summarises it in his doctoral thesis [55]. 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 [60], 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 will be described in 4.1, including *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 [34] are to follow in 4.3, 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 (4.3).

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 [49], 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. *Atmel* offers two software stack for *6loWPAN* and two other packages for *ZigBee* implementation [16, 13]. Only *6loWPAN* was of interest to this project.

- *Atmel RUM* [14]
- *Atmel MAC* [15]
- *Contiki RF230 Driver*
- *Contiki RF230BB Driver*

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*.

The *MAC* stack from *Atmel* has been looked at and was considered not suitable for it's license terms. The license can be found in the source code available form the *Atmel* website [15]. Unlike the *RUM* license, the *MAC* license dictates a set of limitation on how the source code can be used, this package cannot be distributed as part of an open-source project. Nevertherless, it appears that the *Atmel MAC* implementation is generally more advanced (provides more functions) and has a clearer coding style and structure.

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.

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.

Issue Tickets

The following bug tracker issue tickets had been open and are considered completed now, since this part of project has been suspended. This list is included here for refrence only.

Issue #14 *: Compare Avaliable Driver Implementations*

Issue #15 *: AVR cross-compiler toolchain installation*

Issue #16 *: Porting Methodology*

Issue #20 *: Driver Code Licensing*

Issue #24 *: Early Compilation Errors*

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

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

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*** [131, 100]. Comparing to *Contiki*, *TinyOS* has rather superior abstraction layer that provides a seamless cross-platform integration methods [*TEP2* [144], *TEP108* [138], *TEP109* [139], *TEP131* [140]].

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 [94]). 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 [151] 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, *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 [166]. 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 *[143] 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* [155].

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 [92, 93, 133, 132] and the mainstream repository of *TinyOS* source code already contains implementations for some of the proposed protocols [*TEP132* [141],*TEP133* [142]].

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 (2.2.1), however the status of software support for these features of *P802.15.4* is currently uncertain. It is most likely that modification of the MAC layer is required to implement dual-mode transmission (real-time and standard non-priority). It appears however, that some reaserch in this area has been done by the *Nano-RK* developers [123][6], thought more work is needed to find the most appropriate way to implement it within *TinyOS* or *Contiki*.

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 Berkeley Lightweight IP), the stack is currently undergoing major development. Details on what is required are available on *TinyOS* wiki page [32], 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[7]. In the early days of WSN research *P802.15.4* and *ZigBee* where still emerging and the *6loWPAN* RFCs appeared at IETF more recently[8]. *TinyOS* has originally used its own protocol called *ActiveMessage*.

[6] *More details are also available at:*

`http://nano-rk.org/wiki/RT-Link`

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

[8] *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.*

Currently *TinyOS* incorporates *BLIP* stack, which initially was released by UC Berkeley Wireless Embedded Systems research group in the summer 2008, know as *bl6lowpan* at that time [21]. 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.

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[9], 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* [62]. The most interesting designed by Prabal Duta of UC Berkley [61] is shown in Figure 3.1. The reason why it is so interesting is because it features *Digi Connect-ME* microprocessor system block that is of standard RJ45 form-factor [39]. 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.

[9] *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.

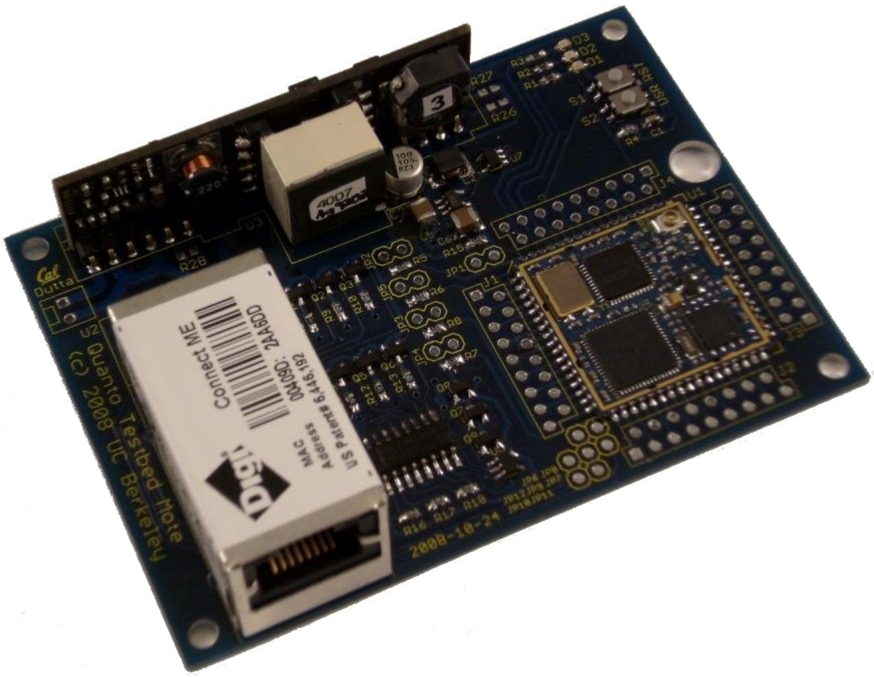


Figure 3.1: *Epic Quanto mote featuring Digi Connect-ME*

Tracked Issues

Issue #30 : Porting Radio Driver Interface to BLIP 2.0

Chapter 4

Wireless Node Hardware

4.1 Overview of Hardware Options: Platforms and Cores

This chapter discusses various alternatives, which had been considered for the target hardware platform. To narrow this selection, an MCU with an integrated radio was of primary interest. Currently, the semiconductor market presents a wide range of single-chip MCUs with *LR-WPAN* interfaces. There is a dedicated section on the WMI website for more information and references to product pages [42, 48].

32-bit ARM

Two different integrated *LR-WPAN* microcontrollers based on *ARM* core architecture were found. Most notable chips are *Freescale* `MC1322x`, which are classified as a *platform-in-package*. *Freescale* combined all transceiver components in one chip, providing a very simple design solution. Only one external RF component that `MC1322x` requires is the antenna. Alternative *Cortex-M3* product is *Ember* `EM300`*-series*, these don't achieve the same scale of integration on the transceiver side and still requires an external RF circuit.

The `MC1322x` gives a great benefit to a designer, as it is known that layout of a radio circuit for frequencies in gigahertz range is particularly challenging task and, indeed, this is a specialist area of board design.

More details on `MC1322x` will appear in section 4.3, including a block diagram of the architecture (Figure 4.4).

Other Microcontrollers

ST Microelectronics and *Ember* offer 16-bit single-chip *LR-WPAN* microcontrollers based on *XAP2* processor architecture.

NEC (now *Renesas*) has a few 16-bit RF chips available, but the selection of development tools and libraries for that platform is rather poor.

Freescale has a another set of 8-bit *LR-WPAN* microcontrollers. `MC13213` family currently consists of just three devices with 16k/32k/60 kB of flash memory and *Motorola 68HC08* core.

A UK company *Jennic* (now acquired by *NXP*) produces a range of wireless micorocontrollers and sub-assembly modules. These are utilising *OpenRISC* 32-bit processor core combined with a proprietary transceiver. The *OpenRISC* core had been developed as a community effort of *OpenCores.org*, however *Jennic* software source code is not available and a rather low quality binary library distribution is offered. This software had been analysed, but the results are irrelevant to this report.

Texas Instruments currently offer a `CC2530` based on *8051* MCU with 2.4 GHz radio, which is most like a product for designers who prefer to use *8051*. *Telit Communications* sells sub-assembly RF and USB dongles featuring the `CC2530` chip. Another interesting product from *Texas* is `CC8520` which allows low-power compressed audio transmission in the ISM band, however as it was mention earlier, transmission of audio is not of interest to this project.

At the time of writing of this report the product range of *TI* wireless ICs has expanded. It now has `CC2511` *SoC* that is based on `CC2530` with a USB interface added and yet another integrated family - *CC430*. These chips are using *TI's MSP430* 16-bit core and a sub-gigahertz *CC1101* transceivers.

8-bit Atmel AVR ATmega128RFA1

The *ATmega128RFA1* [18] chip has 16MHz clock and contains 128kB of flash, 4kB of EEPROM and 16kB of SRAM memories, 32 general purpose registers, 35 GPIO lines, 8 channels of 10-bit ADC and 6 configurable timers, counters and PWM as well as USART, SPI and JTAG interfaces. *ATmega1281* core of the *ATmega128RFA1* is a generic 8-bit microcontroller. Apart from integrated *AT86RFA231* [17] low-power 2.4GHz *P802.15.4* transceiver, there are no other distinct features, it is a stock-standard AVR chip, though the first Wireless MCU from *Atmel*.

A choice of software stacks is provided [14, 15, 16, 13], although the use of these packages is a subject to licensing and architectural issues [43].

4.1.1 Boards and Peripherals

Note: Product URLs are provided on dedicated page of the WMI website [45].

For a radio controller board (RCB) one of the major factors is the type of antenna. Type of sensor connectors and serial interface, as well as presence of JTAG header are also taken into consideration.

`"32-bit"`

Freescale offers a variety of evaluation boards featuring *MC1322x* chips and some interesting sensor ICs. There is a number of *MC1322x* RCBs from 3rd-party suppliers, an up-to-date list is available from *Contiki MC1322x* website [36]. More information on this platform will appear below 4.3.

`"16-bit"`

Two interesting devices which are using TI MSP430 16-bit MCU with a discrete transceiver chip is *Zolertia Z1* board and the *Epic Mote*. The *Zolertia* PCB also features a 3-axis accellerometer and two 3-pin sensor connectors (compatible with *Phidgits* sensors). USB serial interface chip and two anennas are other distinct features of this device. A chip antenna is soldered on to the *Z1* PCB and second micro-FL socket is provied for alternative antennae options. A version of *Z1* with extra peripherals (JTAG, battery holder and external antenna) as well as compact plastic enclosure will be soon available from *Zolertia* sore [175].

`"8-bit"`

Atmel offers a range of RCB evaluation kits [12] and there a few *ZigBit* modules which feature an external RF power amplifier, although those could be great boards to use, the 3rd-party solutions are more suitable for the budget of this project. Two development board options are briefly described below.

First *ATmega128RFA1* board that had been looked at was *Dresden Elektronik Radio Controller Board RCB128RFA1* [50]. In combination with *Sensor Terminal Board* [51], it is a very suitable development and prototyping solution.

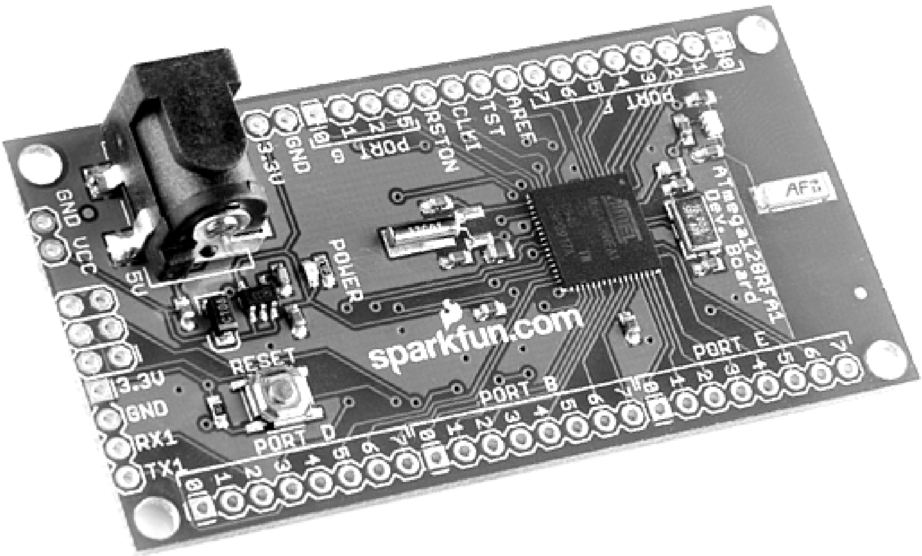


Figure 4.1: *SparkFun* `ATmega128RFA1` *Development Board*

The *RCB* has external antenna connector and the MCU chip is enclosed in a metal shielding and a battery holder is also included. The *STB* boards has screw terminals for sensors and JTAG, ISP, USB and DC power connectors, as well as an external 32kB SRAM chip.

The *SparFun* board pictured in Figure 4.1 has a much more compact design, therefore it could fit into a slim enclosure with *I/O* breakout connections or, otherwise, just a few sensors placed inside of the enclosure for a simpler application. The circuit diagram for this device is shown in Figure 4.3. A pair of these boards has been purchased and used during the first phase of development (4.2). This preference was due to the buget concerns.

4.1.2 Popular Solutions

The *Xbee* [40] modules from *Digi* are very popular among hobbyists, however it is rather a drop-in solution, and is not appropriate for a new design. *Xbee* modules can also be used without external MCU [88], but the functionality is fixed to what is implemented in the firmware. Programable modules are also available [41], however very limited information has been found. One recently published book [112] does suggest to use the *Xbee* modules, althogh author mentions no alternatives. *IcludeTech WiMi* [79] is a development board which uses an *Xbee* module with a whip antenna and a *Microchip PIC* MCU. It would be appropriate as a drop-in module for a MIDI device which is being produced already and wireless connectivity needs to be offered as an expansion option, engineers who are familiar with *PIC* microcontrollers may find these devices suitable for their designs.

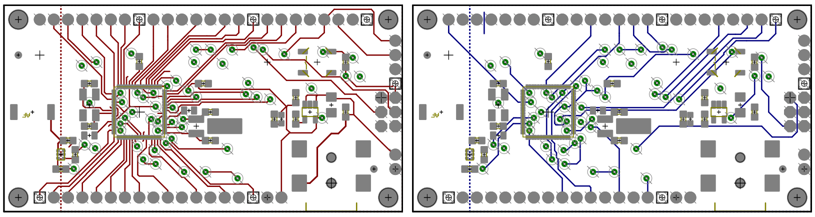


Figure 4.2: *SparkFun development board layout (top and bottom)*

4.2 Initial Hardware Consideration

At the initial stage of this project, the hardware platform was chosen based on previous experience of using development tools for *Atmel AVR* microcontrollers. The original orientation was towards using a chip with integrated radio transceiver, *Atmel* advertises the ***ATmega128RFA1*** chip as having the lowest power consumption and most appropriate link budget.

Development Boards for ATmega128RFA1

As it was already mentioned, two choises of development boards were considered:

- *Dresden Elektronik [50, 51]*
- *SparkFun (Figure 4.1)*

DE hardware features a useful set of components, including a spare 32 kB of memory and robust screew terminals for I/O connections. However, the board from *SparkFun* has been chosen for it's low price and the GPIO pins of all ports are made available on the PCB edges.

It has been found later that this board was inconvenient to use in various ways, for example the layout of serial port pins on the side of the board could had been designed to fit standard serial USB cables[1]. As it can be seen in figure 4.2, the layout of transciever side of the PCB was found to be quite primitive and, most importantly, has not included a suitable ground plane nor it has a shield. Nevertheless, this board is of a rather small footprint and could be used in a prototype product.

Certainly some rework can be done to improve the layout, since *SparkFun* published this design under the *Creative Commons License*, which allows freedom of sharing and adoption of the circuit, provided that the original copyright notice is attributed [30, 159].

[1] *SparkFun sells few different serial USB adaptors all of which have the same layout. These are know to be very popular among most of hobbists and professional engineers and therefore are considered to be a de-facto standard.*

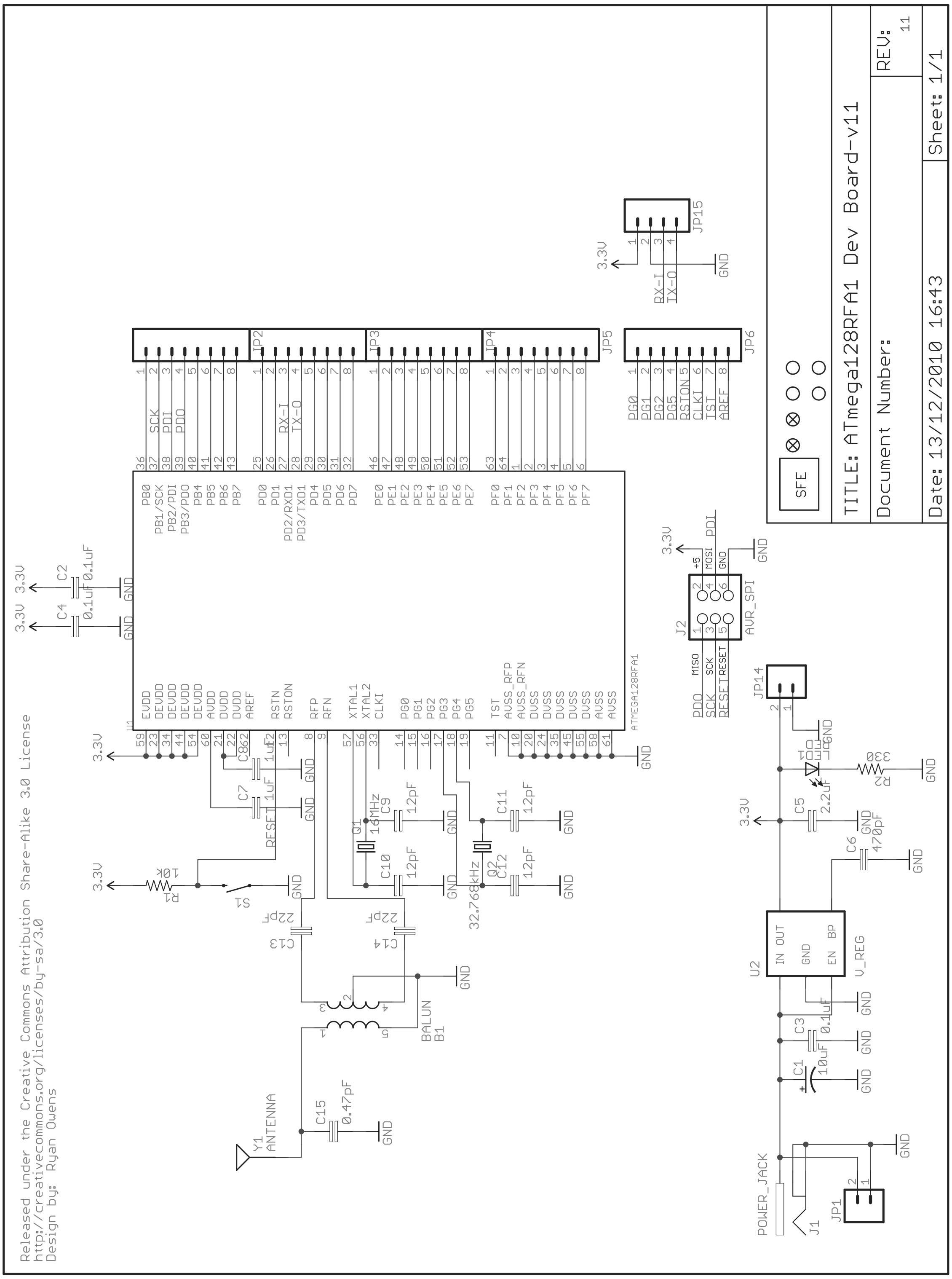


Figure 4.3: *SparkFun development board schematics*

4.3 Selecting the Development Platform for Contiki OS

Examining the code for various hardware platforms in the source tree of *Contiki*, it was understood that source code for `MC1322x` devices from *Freescale* is organised in a much clearer way. Current implementation appears to support most of important features of these chips and, in fact, these are very robust devices. As mentioned earlier, `MC1322x` is an `ARM7TDMI` microcontroller with fully integrated radio and the only component external to the chip itself is the antenna. Apart from this, the *Freescale* device has an outstanding set of peripheral and rather large amount of memory[2] (128KB of flash, 96kB of RAM and 80kB of ROM) the processor clock frequency is 24MHz. Comparing to the minimum amount of memory required to run *Contiki* (2kB of RAM and 40kB of ROM) there is a very large headroom available for the application and some additional drivers. This also makes it a great development platform, during debugging stages compiler optimisation often needs to be disabled and additional memory may be also populated with debug data.

4.3.1 MC1322x Architecture Overview

In addition to standard peripherals (UART, 12-bit A/D converters, SPI and I2C) this device has *Synchronous Serial Interface (SSI)* that would allow communication with I2S devices as well as other synchronous serial peripherals. Being a 32-bit microcontroller it can be used for some basic audio signal processing. For example, with *Analog Devices* `ADMP441` omnidirectional microphone with 24-bit I2S digital output [3] an acoustic measurement sensor node can be designed. According to the datasheet, `ADMP441` has a very linear frequency response in the band between 100Hz to 15kHz. Such wireless node would be suitable for field noise measurement at popular music festivals or construction sites and industrial areas. Another sensor that could be included on such board would be an accelerometer, according to *Freescale Application Note AN3751* [69] by utilising standard DSP techniques on accelerometer signal data, various vibration frequency analysis results can be produced. Another feature of the `MC1322x` is it's accelerated MAC unit, called *MACA*.

4.3.2 MC1322x Development Hardware

The homepage of `MC1322x` port of *Contiki* has a detailed overview of what development boards are available [36]. Two boards which were chosen for this project are *Freescale 1322xUSB* and *Redbee Econotag*.

A minimum of two devices is required to implement a *LR-WPAN* network. At the early stage, the *SparkFun* boards were supposed to be used as the sensor node and the gateway. When the decision had been made to use `MC1322x` platform, the *Freescale* dongle was found to be the most appropriate device to run the *Contiki Border Router* program, which implements end-to-end IP communication[3].

[2] *An average size of compiled code for the Talker program was reported to be 79 kB.*

[3] *The source code is located in:* `examples/ipv6/rpl-border-router/`

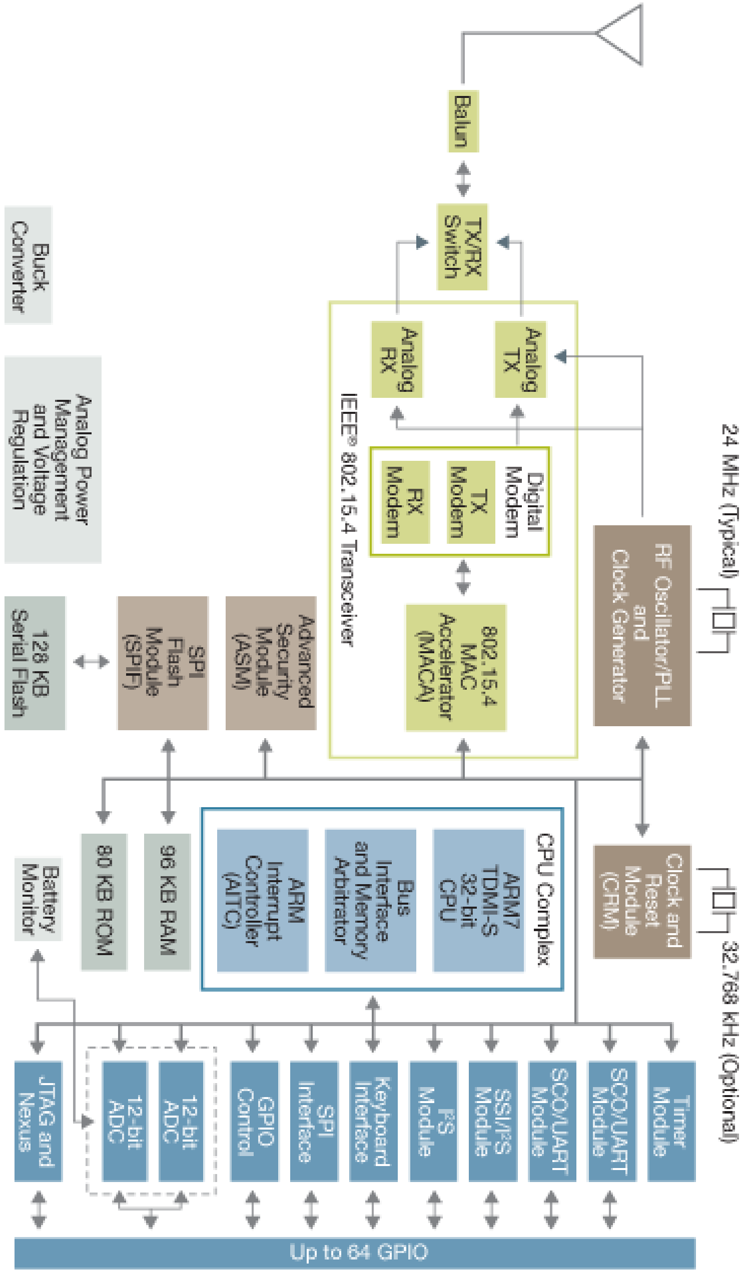


Figure 4.4: *The block diagram of Freescale MC1322x [72]*

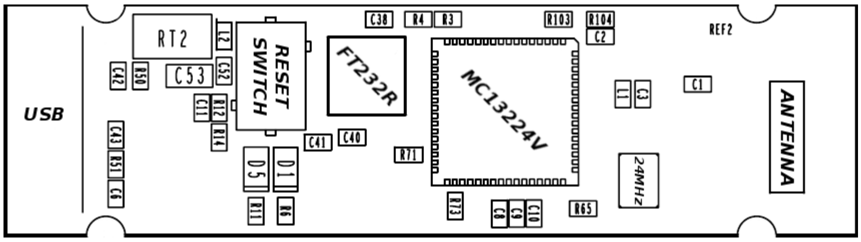


Figure 4.5: *The circuit layout of Freescale 1322xUSB*

Freescale USB Dongle

1322xUSB is a very small USB device that doesn't provide the access to any of the pin and therefore is well suited as a network gateway. The circuit diagram is shown in figure 4.5 to demonstrate the minimalistic layout of this board. One the bottom side of the board, there is a header for a JTAG connector and a few test points. It has appeared that the board has already been flashed in the factory, and the firmware had to be erased when the board was used for the first time to run the *Contiki* code. The information on how to erase the firmware was obtained from the datasheet [71]. This device was found rather primitive, its only benefit is the size, though it also serves as the minimal reference design.

Redbee Econotag

Econotag by *Redbee LLC* is highly regarded open-source development board. It is specifically design to meet all important requirenements of a development board with very constrained bill of materials. The board provides access to all of the I/O pins and and is very convenient for programming and debugging. Both of these functions are achived with the help of dual-port USB UART chip, *FTDI* ***FT22232HL*** [73]. One of the two ports is connected to the primary UART interface and the second port provides access to the JTAG. The ***FT2232HL*** IC also facilitates bit-bang software reset, the *Freescale* dongle cannot be reset in this way.

4.3.3 Hardware Programming Facilities

To program the ***MC1322x***, the first UART port is used. It also used for standard serial port output. The program is localed into RAM and doesn't wear-off the flash memory each time. Thought, for flashing the ***MC1322x*** devices, a two-stage procedure is needed. First, a self-flasher code has to be loaded, which in turns loads the main program code into the flash memory.

Contiki source code contains the self-flasher implementations as well a set of utilities for programming the ***MC1322x*** devices[4] However, this is not the only way to update the software. Dunkels describes the design of *ContikiExecutable and Linkable Format* [57] loader. There are a few implication of using ELF loader, e.g. there has to be a certain amount of spare memory for the code to be downloaded into. However, it gives a great advantage of reprogramming over-the-air, without a need for physical access to the node.

[4] *These are located in* `cpu/mc1322x/tools/`.

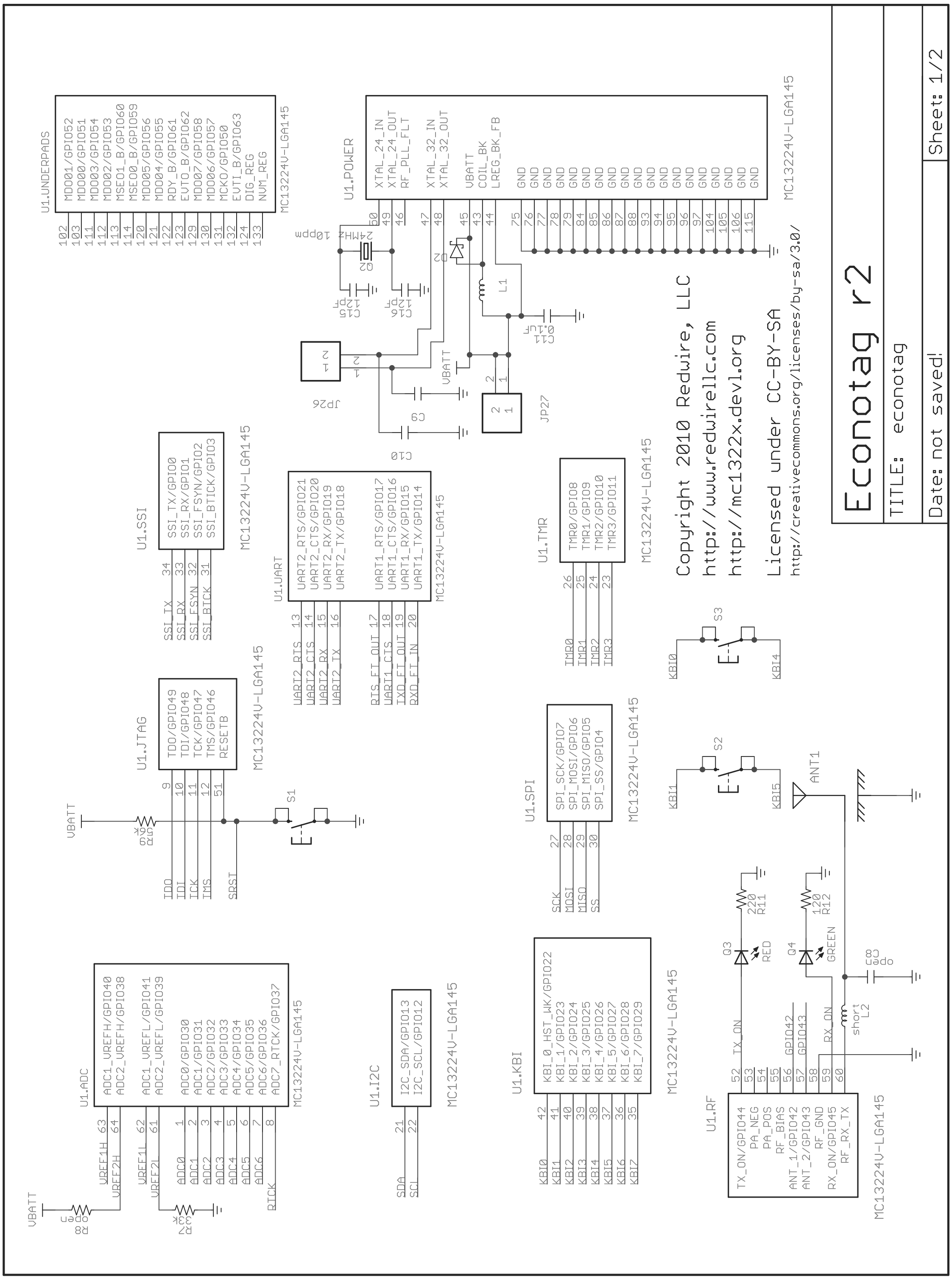


Figure 4.6: *Econotag schematics (sheet 1)*

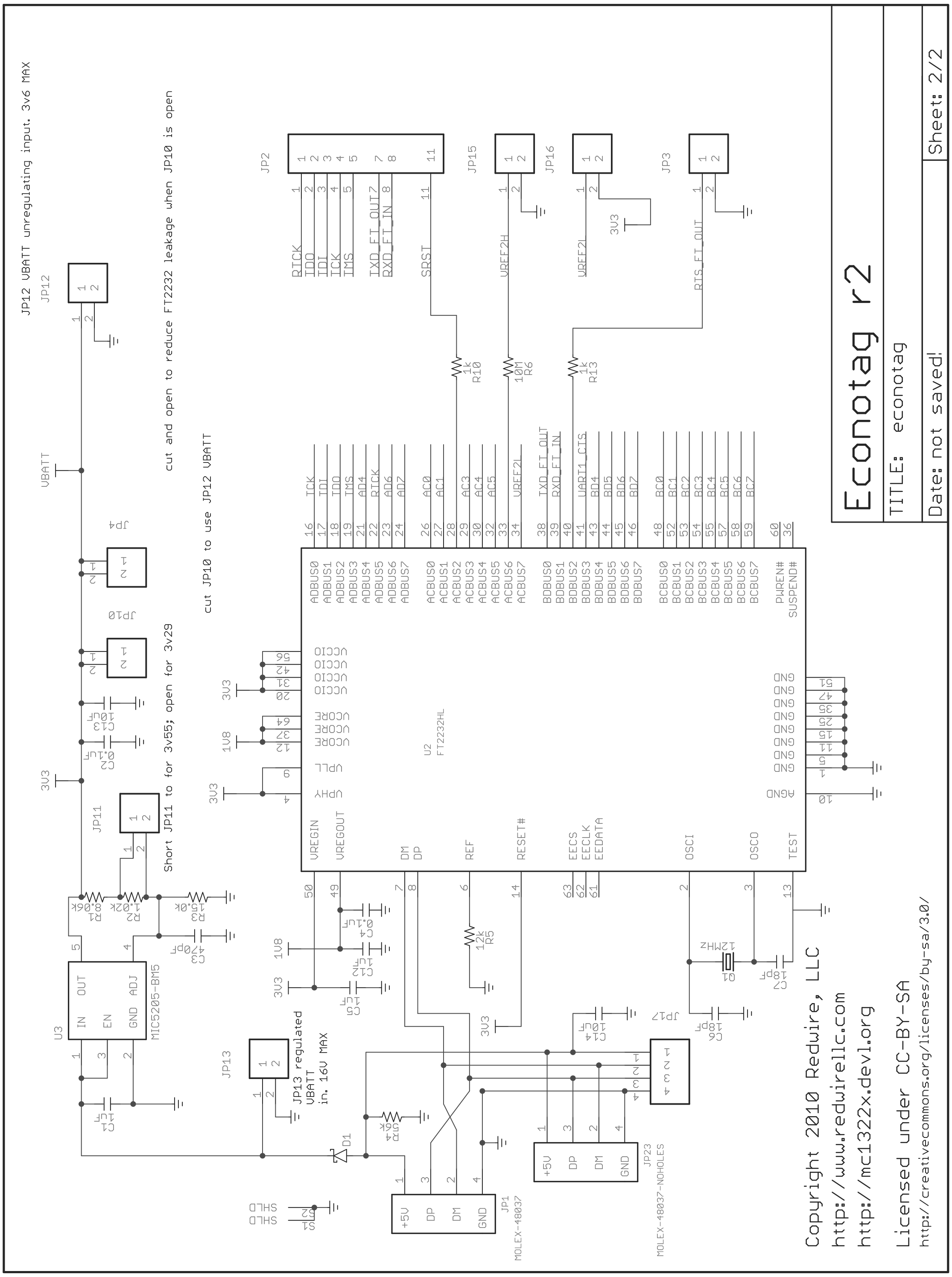


Figure 4.7: *Econotag schematics (sheet 2)*

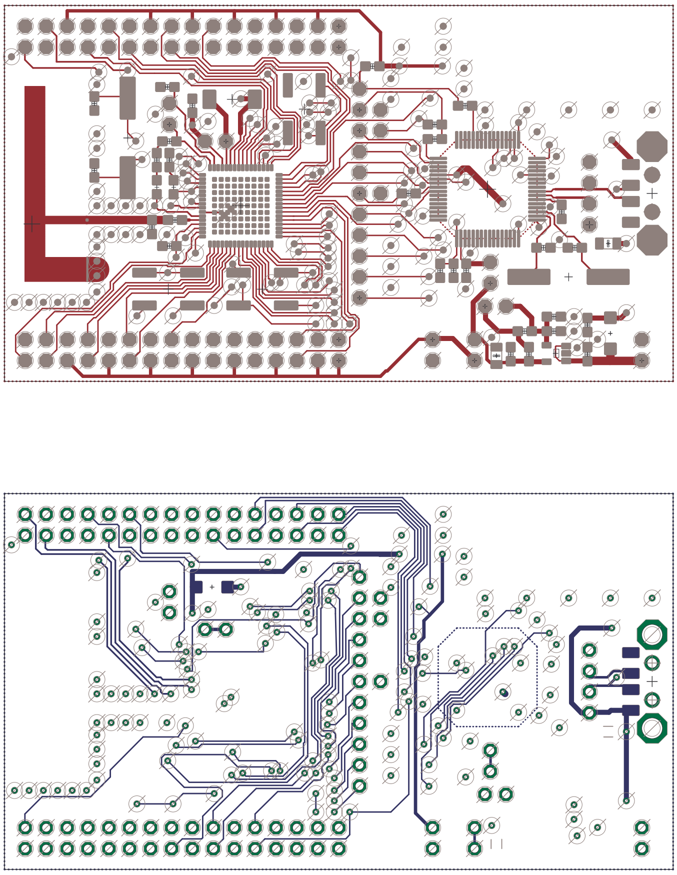


Figure 4.8: *Econotag development board layout (top and bottom)*

4.3.4 Econotag Circuit

The schematics show in figures 4.6, 4.7 and the board layout in 4.8 had been taken from the design package [122] that is distribute by *RedWire* under *Creative Commons* license. These are included here to approve the simplicity of circuit layout with highly integrated ***MC1322x*** chip.

Comparing to the *SparkFun* board, the *Econotag* demonstrates much superior development solution with the inclusion of dual USB chip, that provides software reset and JTAG facilitates. Also many other minor aspects of this board provide a greater benefit then the earlier board. Althought, the layout shown above does not include the ground plane, by by visual comparison, the *Econotag* has a very solid ground plane, unlike the *SparkFun* board.

The area where more experiments are still to be done, is the power consumption measurement. It is know however, that the *Econtag* will run of two cell batteries for up to 48 hours without utilising any of the sleep modes, however this information still has to be proven.

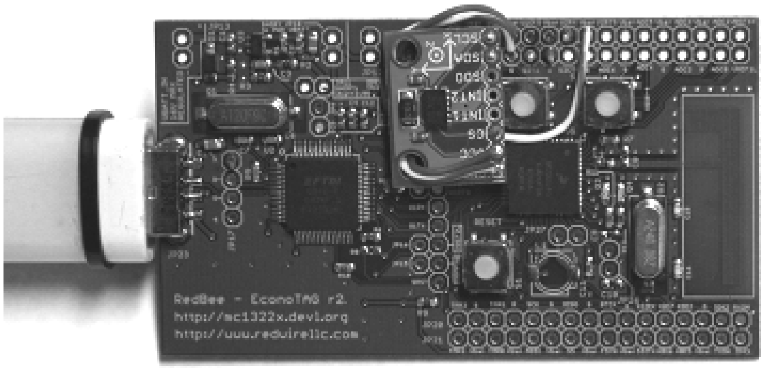


Figure 4.9: *Econotag (top view)*

4.3.5 Photographs of the Hardware in Use

The photograph in figure 4.9 shows the top view of the *Econotag* with an `ADXL345`[5]. break-out board soldered to the SPI pins. The second image (4.10) is the bottom view of the Econotag with MIDI cable soldered to the UART2 pins, the DIN5 MIDI connector and the *Freescale* dongle. It can observed the size of the dongle is very close to the size of DIN5 connector, therefore a device can be designed which would be of a size little bigger then the DIN5 connector body, though the power would need to be supplied. The standard midi connector has 5 pins and only two are used by most of the devices, hence, theoretically, this could be implemented. However, in practice, a battery will be required, since the modification of MIDI device to provide a power supply is not an option. The lack of bus power is another reason to consider MIDI being a legacy protocol.

[5] *More details on the use of the sensor are to follow.*

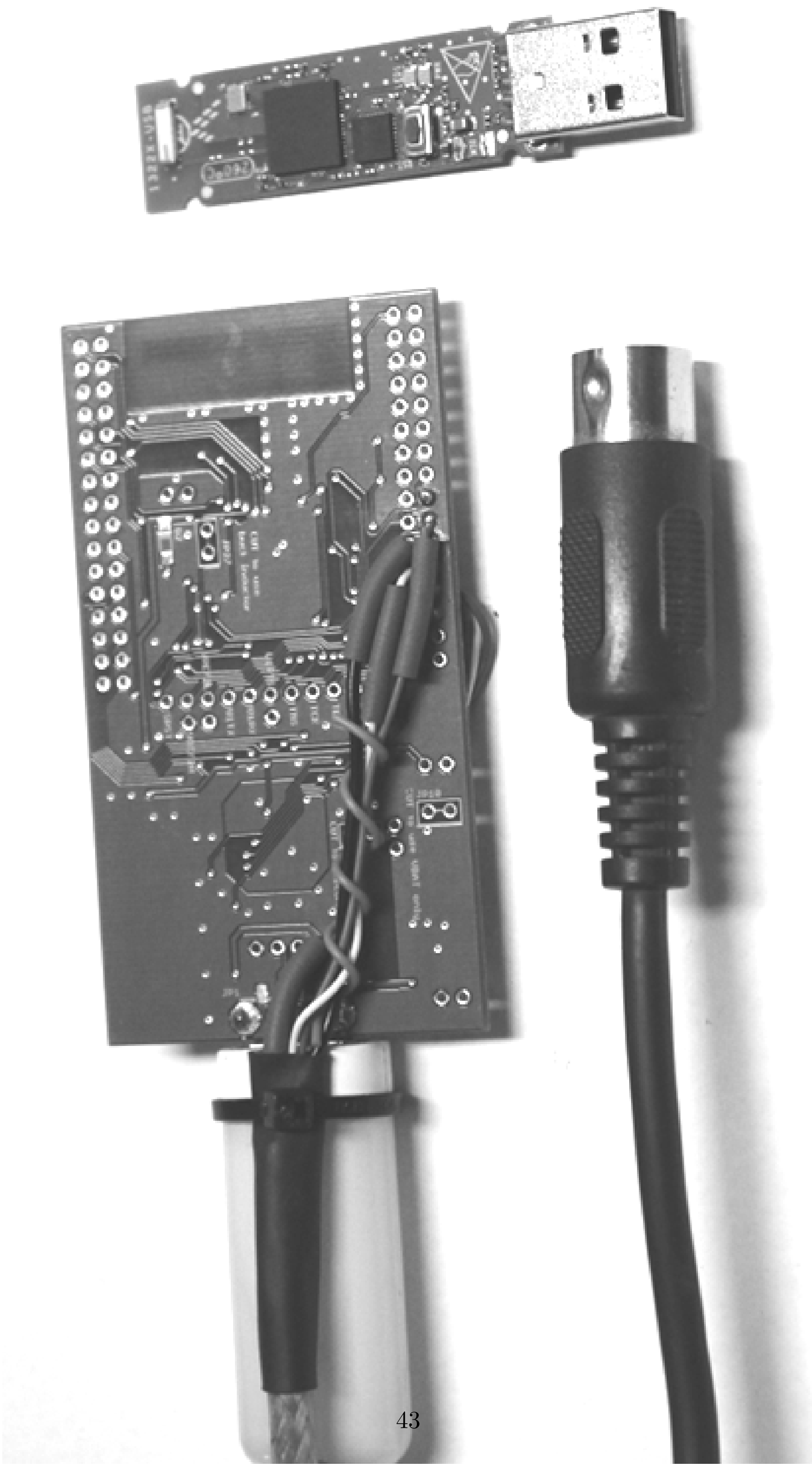


Figure 4.10: *Freescale USB dongle and the Econotag (bottom view)*

Chapter 5

DSP Host System

5.1 DSP Host Hardware

A set of hardware platforms suitable for the 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 *MIPS64* devices by *NetLogic* [111], for example, have great computational capacity, thought these are too expensive for this particular application.

5.1.1 x86-based Embedded Devices

The list above does not include the *x86*-based CPUs due to a 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 is also a necessity[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 instance, have rather more flexible facilities for bootloader and set-up. Although, in the class of *single board computers (SBC)*, there are many boards which do not include the physical connectors for above mentioned peripherals, however such boards are designed for industrial use and require a specialised chassis [97].

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

5.1.2 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 within the SoC. Very limited documentation is provided on how it can be used in the *Linux* environment [145].

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

One most outstanding development platform that uses an *OMAP* chips with a dual-core *Cortex-A8* CPU clocked at 1GHz - is the *PandaBoard* [146]. However, it 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 utilise.

5.1.3 Other ARM CPUs

ARM CPUs are produced by almost any major semiconductor firm, including *Freescale* and *Marvell*. Not all fabricators make very high-performance *ARM* chips, wich are clocked near to 1 GHz, and just a few make multi-core SoCs. And only some chips 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 that wasn't a concern [11]. It is needless to mention that both vendors fully support Linux OS.

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* [102]. *Kirkwood* series [103] are not featuring a *VFP*, thought there are some outstanding embedded development platforms available.

Plug Computer (also know as *Sheeva Plug*) is small, yet very powerful computer in a form-factor of wall socket DC adaptor [101, 118]. 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

Freescale Semiconductors offers a range of *ARM* processors branded *i.MX* [67]. The datasheets had been examined to find out which chips include a floating-point unit. Among the mid-range, ***i.MX31*** devices (532 MHz ARM1136JF-S) appear to have support for vector floating-point instructions, however the clock rate of the ***i.MX31*** SoC would suite only a limited number of applications [70]. In the upper-range of *Freescale i.MX* solutions, there are single-core *Conrtex-A8* devices with clock rates up to 1 GHz [68], namely the ***i.MX535*** and ***i.MX538***. The second SoC has better video coding and storage capabilities.

It has been noted that a simple development board for ***i.MX535*** can be purchased for a relatively low price[2].

[2] *There two basic options: one for 150 US dollars, and another for 200 with a touch-screen.*

5.1.4 SHARC

Analog Device is a well-known manufacturer of DSP chips, the *SHARC* architecture is different from *TI OMAP* and the signal processing instructions are handled directly on core[3]. The clock frequency is not being the main performance factor of the *SHARC* devices and range between 300 and 600 MHz, instead FLOPS (Floating-Point Operations per Second) are used to define the speed of these processors [4]. A *SHARC* chip suitable for audio typically is rated at 2700 MFLOPS. Linux has been a popular platform for advanced *SHARC*-based systems appearing in various application markets.

SRARC processors are commonly used in professional audio equipment. One very outstanding product has been released by a new British company *Dark Matter Audio* this year, it is called *DMA1* [31]. It is a portable system for interactive music performance with very novel design features and powerful *SHARC* DSP engine. According to the product announcement it runs Linux 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.

SHARC development hardware is mostly available from the silicon vendor itself and very few third-party boards could be found. Since this 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 files [2].

[3] *As it was mentioned above, TI OMAP is a combination of ARM and C64x DSP, while SHARC is a monolithic SoC. Nevertheless, some designer chose to use SHARC with an ARM control CPU, e.g. the DMA1.*

5.2 Building Embedded Linux OS

5.2.1 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* [20]. 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.

5.2.2 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* [114] 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 [104].

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"* [74]. 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* [23], 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* [124]. 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* [96], 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 [96]. 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].

5.3 Conclusion

It had been desired to build an embedded DSP host OS as one of additional project targets, however due to various[5] limitations, most importantly - the time frame, the project has not achieved this at the time of writing of the final report. The above chapter was included to provide the evidence of research in this area.

5.4 Synthesis Software

Another target which has not been achieved, since it was dependant on the design of the DSP host, was to use visual DSP modeling package *Pure Data* [169, 120]. It would be a simple task and there is no particular concern of how to integrate the current implementation with *Pure Data*[6]. Another reason for disregarding this target, was that the emphasis of the entire project has changed towards various aspects of wireless sensor network implementation and audio synthesis is outside the general scope of this report.

[4] *That implies it is being directed by the founder company, Canonical.*
[5] *Another aspect was due to hardware selection and purchasing issues.*
[6] *The author has extensive experience of using this package*

Chapter 6

Application Development

6.1 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* [29, 35], 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], a text editor and the GCC toolchain for ARM [38], there are three essential command-line tools which were utilised during the development process:

gdb **[136]** - GNU Debugger (source-level) [135]

make **[107]** - GNU make program [134]

OpenOCD - On-chip Debugger [37]

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 in the root directory and one for each processor architecture).

Most of the changes were made in `cpu/mc1322x/Makefile.mc1322x` to provide a few of shorter commands for setting-up the debugger and sending the program to run on a development board[3]

6.2 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

- `cpu/mc1322x/src/default_lowlevel.h`
  prototypes

[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.c`
  initialisation functions

A set of macros was defined to aid the application code, though most of those macros were used only for initial debugging.

Soon, it has been understood that the original implementation was missing handler functions for the UART interrupts. With the help of communication on the *Contiki* mailing list, the appropriate methods were realised. One of the key techniques was to use *"weak"* function linking attribute. Two macros named `U2_RXI_POLLHANDLER()` and `U2_TXI_CALL()` are appearing in the program listing 6.1. These macros define the function which is called by the interrupt routine in `cpu/mc1322x/lib/uart2.c`[4].

Tracked Issues

Issue #27 *: Compiling various example programs*

Issue #31 *: Wireless transmission of MIDI is proven working*

Issue #32 *: Requirenments for the MIDI UART driver*

6.3 Designing the MIDI Talker Program

A variety of constructs had been tried for handling the FIFO buffer into the *uIP* packet buffer. This was rather challenging and many complicated bugs appeared throughout out the process. The JTAG debugger had been utilised a number of times to find these bugs.

The macro named `PSOCK_GENERATOR_SEND()` had been found as the most direct way to transfer the data from FIFO into the packet buffer. This macro is defined and documented in `core/net/psock.h`, it is a part of *Contiki protosockets* library.

6.3.1 Protosockets

This section gives more detail on *protosockets*, which were introduced in 3.2.

The *protosockets* is a set of C macros (and just a few C functions) defined and documented in:

- `core/net/psock.h`
- `core/net/psock.c`

These are indeed simple methods for a TCP application to utilise. Nevertheless, it is commonly known that UDP communication is most appropriate for a type of program that is discussed here, the results included included in this chapter will show that TCP can be used for a simple program like this. If UDP was to be utilised, additional algorithms for retransmission and stream

[4] *For the definition of the macros, see:*

- *`cpu/mc1322x/lib/include/uart1.h`*
- *`projects/wmi/mc1322x/midi/uart2-midi.h`*

data ordering would need to be developed. As discussed in section 6.5, it should be rather more appropriate to develop a library for *RTP-MIDI* instead of a simple UDP application. There is a great benefit to use TCP at this stage, as it provides the guarantees of data being transmitted correctly and in a sequential order on the socket level.

For detailed documentation of the function which will appear in the program listing, see the *Contiki* [52] and the *uIP* [53] manuals. It would be misleading to included extracts of documentation here, since lower-level functions and algorithms would need to be listed.

6.4 The Source Code

The Talker program is considered to be complete and most of the possible bugs had been eliminated from this code, therefore it shall be presented here. Some lines had been omitted for clarity and complete version can be viewed in the repository (`projects/wmi/mc1322x/midi/psock-taker.c`).

The Listener program, however, was not finished at the time of submission of this report. The current version is missing an algorithm which would process the packet buffer before it had been filled-up. This is considered a major design problem, which should be solved within a short period of time after the submission of this report (see `projects/wmi/mc1322x/midi/psock-listener.c` for the current implementation).

6.4.1 Further Enhancements

The flow diagram of the Talker program is show in figure 6.2 and the Listener flow is shown in 6.1. There is problem with this approach - the Listener has to know the address of the Talker, which doesn't make this network very flexible. It would be rather appropriate to implement a self-advertising device. That can be achieved by adding message broadcast thread, as shown in figures 6.4 and 6.3. These diagrams also include an additional sleep state, which may be considered an optional element. The sleep delay calculation would need to be defined in such way that there is possibility of nodes not detecting each other. It should be also noted that these flow charts demonstrate the behaviour of wireless nodes, thought a similar flow can be used for the host program.

Listing 6.1: *The Talker program*

```
#include "contiki.h"
#include "contiki-net.h"
#include "sys/ctimer.h"

#include <string.h>

#include "uart2-midi.h"

typedef char data_buffer_t

/*
 * To be able to handle more than one connection
 * at a time, each parallell connection needs its
 * own protosocket, so TCP_thread would need to
 * be an array of type psock.
*/
static struct psock TCP_thread;
static struct pt    URX_thread;

/* The datastructure needed for this program */
struct signal {
        struct timer    time;
        unsigned short  size;
        unsigned short  flag;
};

enum { // Flag values:
        SKIP = 0xff,
        SEND = 1,
        SENT = 0,
};

// Buffer Length (has to be defined)
#define BL 32
// Queue Length (set to zero to disable)
#define QL (BL/2)

/* This function is only needed for
 * PSOCK_GENERATOR_SEND() to work */
static unsigned short
URX_proc(void *x)
{
  return ((struct signal *)x)->size;
}

/* Process declarations */
PROCESS(Talker, "MIDI Talker");
AUTOSTART_PROCESSES(&Talker);
```

```
/* RX Interrupt handlers */
U2_RXI_POLL_PROCESS(&Talker);
/* TX Interrupt is disabled */
U2_TXI_CALL(U2_RX_ONLY());

/*-------------------------------------------------------------------*/

volatile struct signal RX;
/* Protosockets require it to be a pointer */
static struct signal *urx = (struct signal *) &RX;

static data_buffer_t tcpbuf[BL];

/*-------------------------------------------------------------------*/
/*= This thread handles the data buffering =*/
static
PT_THREAD(URX_fill(struct pt *p))
{

  PT_BEGIN(p);

  if (urx->flag != SKIP) {
    PT_WAIT_UNTIL(p, (urx->flag == SENT));
    /* It should be okay to proceed now */
  }

  if (urx->size < QL) {
    /* Bytes not sent, increment the size */
    urx->size += urx->size;
  } else {
    /* The bytes have to be skipped */
    urx->size = 0;
  }

  while(*UART2_URXCON > 0) {
    /* Copy the buffer bytes and set the size */
    bcopy(UART2_UDATA, &uip_appdata[urx->size++], 1);
  }

  if(uip_conn != NULL) {

    /* Set the flag and poll the control thread */

    urx->flag = SEND; tcpip_poll_tcp(uip_conn);

  } /* Skipping if there is no connection */

  PT_END(p);
}
```

```
/*-----------------------------------------------------*/
/*= This thread handles the data transfers =*/
static
PT_THREAD(TCP_send(struct psock *p))
{
  PSOCK_BEGIN(p);

  while(1) {

    /* Wait for the data in the buffer */
    PSOCK_WAIT_UNTIL(p, (urx->flag == SEND));

    /* Push the buffer of the given size */
    PSOCK_GENERATOR_SEND(p, URX_proc, urx);

    /* It should had been sent */
    urx->size = SENT;
    urx->flag = SENT;

  }

  PSOCK_CLOSE(p);

  PSOCK_END(p);
}


/*-----------------------------------------------------*/
/*=// This process controls both threads //=*/
PROCESS_THREAD(Talker, ev, data)
{

  /* When this process is polled,
   * there is data ready to collect
  */
  PROCESS_POLLHANDLER(URX_fill(&URX_thread));

  PROCESS_BEGIN();

  /* Set-up UART2 as a MIDI port:
   * Baud Rate: 31250;
   * Parity: Off;
   * Stop Bits: 1;
   * Start Bits: 1;
  */
  midi_uart_init();

  /* Initialise the flag */
  urx->flag = SKIP;

```

```
  /* Set-up the timer (unused) */
  timer_set(&urx->time, CLOCK_SECOND * 120);

  /* Create the protothered
   * for the interrupt handler
  */
  PT_INIT(&URX_thread);

  /* Open the TCP port 2020 */
  tcp_listen(UIP_HTONS(2020));

  while(1) {

    PROCESS_WAIT_EVENT();

    if(ev == tcpip_event) {

      if(uip_connected()) {

        /* Once connection has been requested,
         * create the TCP thread and, if it is
         * failing, re-initialise the flag */
        PSOCK_INIT(&TCP_thread, tcpbuf, sizeof(tcpbuf));

        while(!(uip_aborted() || uip_closed() ||
            uip_timedout())) {

          PROCESS_WAIT_EVENT();

          /* There should be a connection set-up now. */
          if (ev == tcpip_event) TCP_send(&TCP_thread);

        }
      }
    } else { urx->flag = SKIP; timer_reset(&urx->time); }

  PROCESS_END();
}
/*---------------------------------------------------------------------------*/
```

Listing 6.1: *The Talker program*

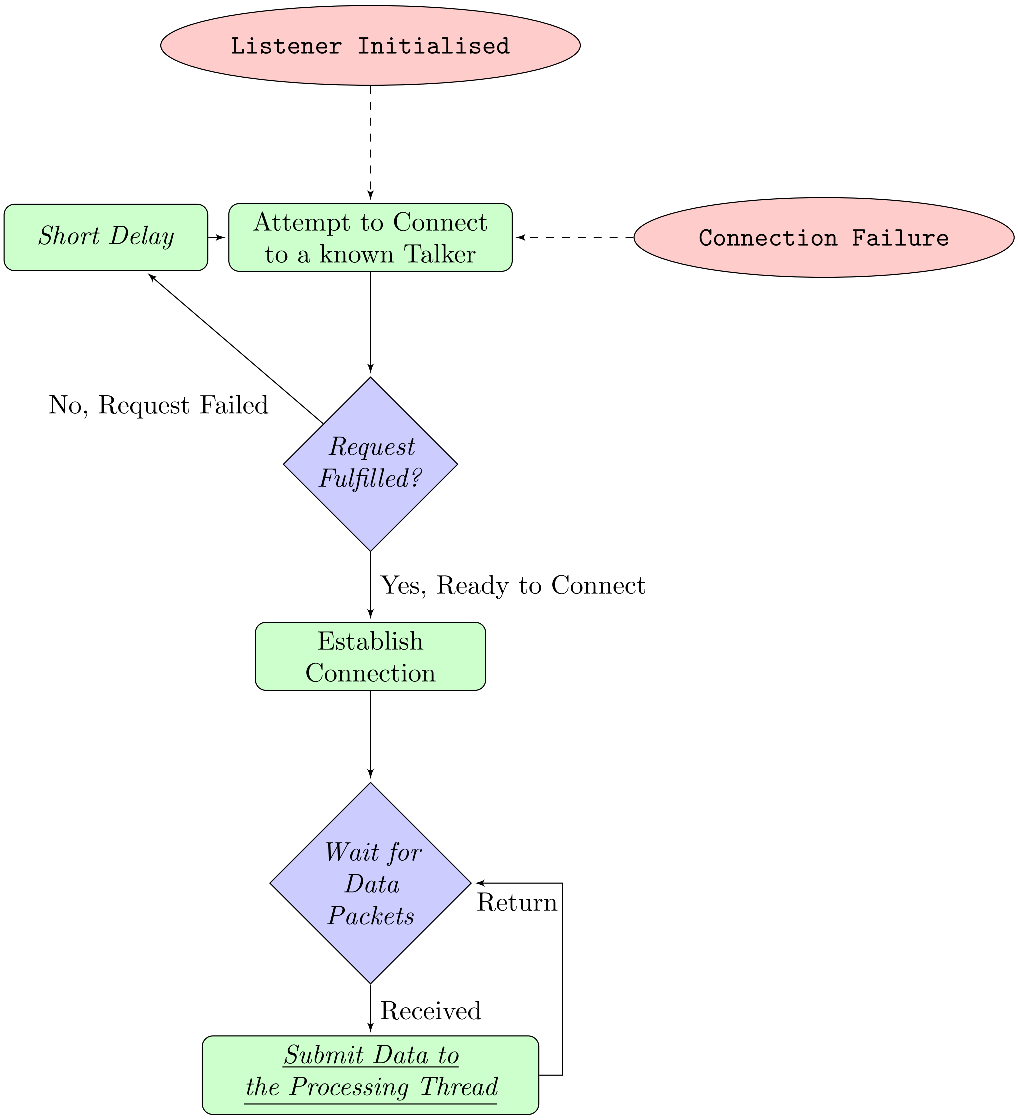


Figure 6.1: *Flow-chat of simple Listener program*

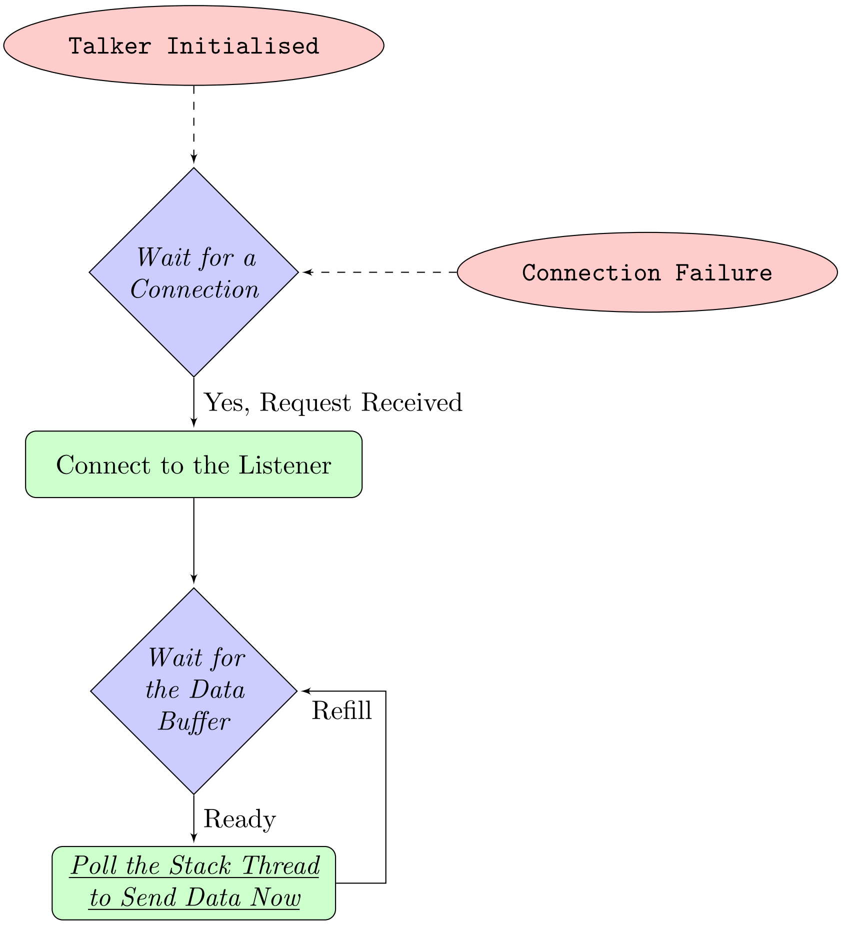


Figure 6.2: *Flow-chat of simple Talker program*

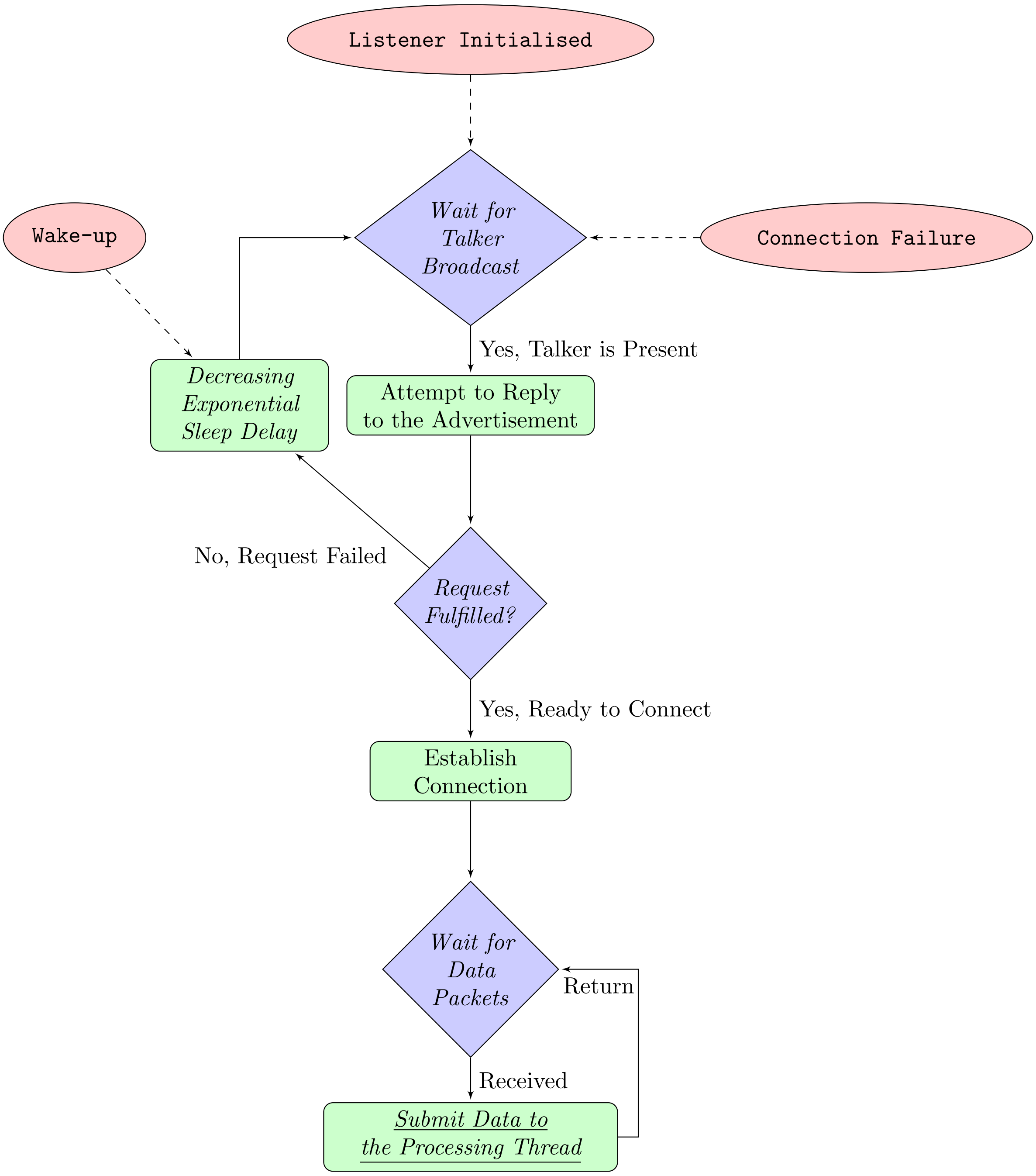


Figure 6.3: *Flow-chat of self-advertising Listener program*

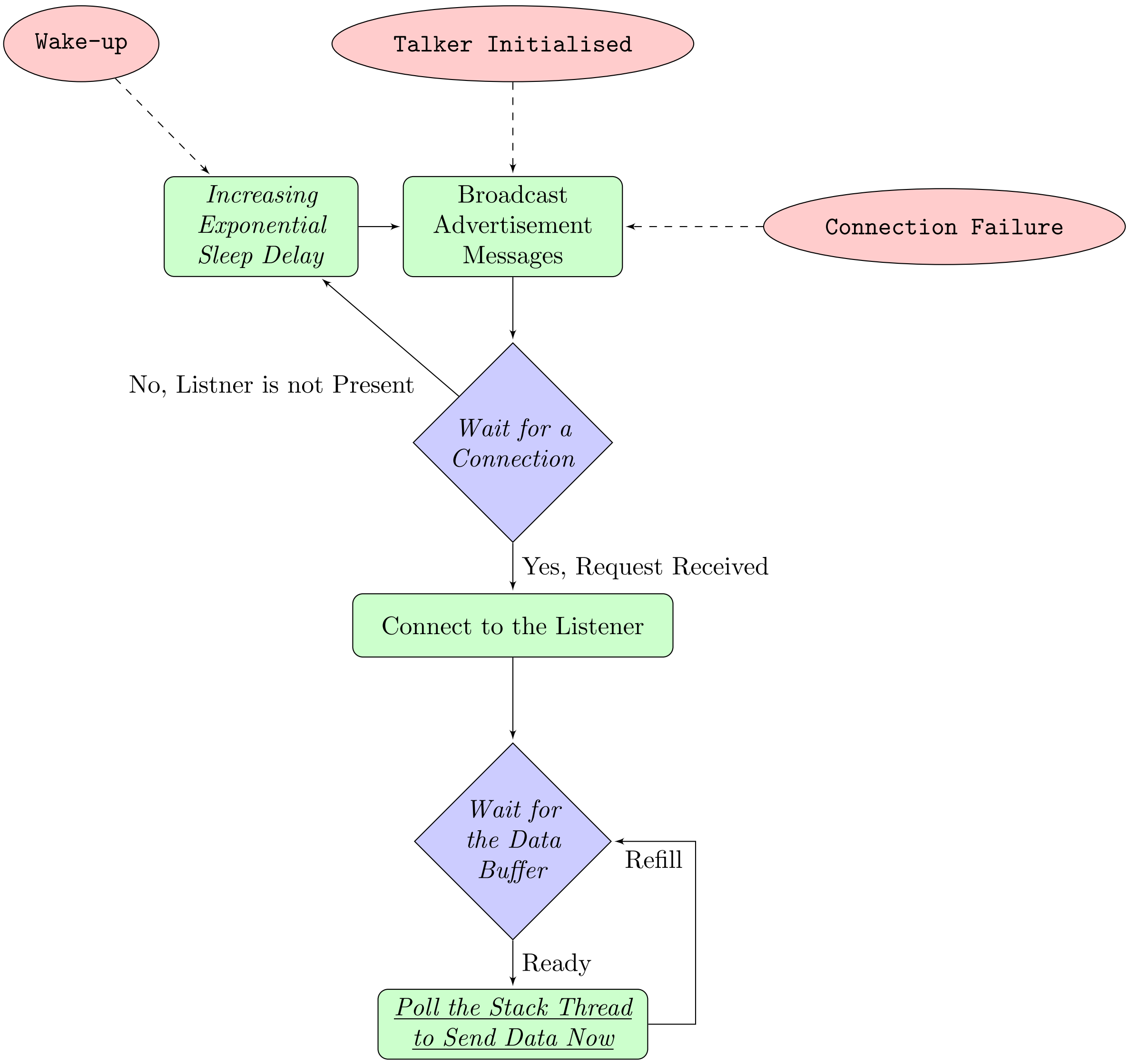


Figure 6.4: *Flow-chat of self-advertising Talker program*

6.5 Implementation Decisions

Two programs where written to test transmission of MIDI. The concept of server/client communication could be applied, nevertheless it had been found more appropriate to name the parties as a *Talker* and a *Listener*. At the time of writing of this report the *Talker* program had been fully tested and debugged. The second program can be considered only as a prototype and does require further work.

The *protosocket* macros had been utilised for the implementation, hence the communication protocol imposed by *protosocket* was TCP. There are various improvements that are still to be considered. The specification of TCP includes a notion for *urgent data* (*RFC0793* [119]), however this feature has received a very limited adoption and is known to be handled differently by various implementations of TCP [75]. It has been found that the *uIP* stack does implement some mechanisms for the *urgent data*, however the documentation on how it can be applied in *Contiki* application code, could not be found.

6.5.1 UDP Multicasting

A variety of packages exist which utilise UDP multicasting for streaming of the MIDI data on LAN. These include *ipMIDI* [110] for Microsoft Windows and Apple Mac OS X platforms as well as compatible packages for Linux - *multimidicast* [108] and *qmidinet* [121]. This two Linux packages had been tested with current implementation of the *Talker* program and have shown a suitable performance[5]. The multicasting approach is considered to be quite appropriate for streaming of MIDI signals on IP networks, since no configuration is required, i.e. none of the participants have to set-up connection between each other and only need to send data to the multicast IP address[6]. However the *ipMIDI* is not a formal protocol, it simply streams the MIDI signals to a UDP socket, i.e. there are no any error correcting mechanisms or hand-shaking.

There had been no implmentation of UDP multicast in *uIP* and there are certain difficulties associated with this, since *P802.15.4* only defines broadcasting and doesn't define multicasting. However, with the help of *Contiki* mailing list, a patch had been received from Anuj Sehgal and tested. There had been no feasible test results of using this implementation.

Tracked Issues

Issue #33 *: Testing UDP Multicasting (with uIP)*

One simple test has also been made with UDP broadcast. However, this is not appropriate to use, since that would be considered as flooding of the entire network. More suitable use of broadcasting was shown in the previous section (figures 6.3 & 6.4).

[5] *This has been achived by piping the data from the Talker node to a multicast client on the host and then receiving this at another host on the same LAN. This was done with a few standard network testing utilities available on Linux, however for a deployment, a specialised program needs to be developed.*

[6] *For compatibility with ipMIDI, the multicast IP address should be 225.0.0.37 and the port numbers from 21928 and above.*

6.5.2 MIDI payload for Real-time Transport Protocol

Two relevant Internet standards had been published by the IETF in 2006:

- *"RTP Payload Format for MIDI"* (*RFC4695* [91])
- *"An Implementation Guide for RTP MIDI"* (*RFC4696* [90])

This two papers had been studied and considered as a possible solution for encapsulating MIDI data stream on a *6loWPAN* networks, however the implementation would require a library for the *Real-time Transport Protocol* as well a large set of algorithms to handle various aspects of the *RFC4695*. It would be a very challenging task to design *RTP-MIDI* library which could run on a microcontroller.

One very important aspect that is defined in *RFC4695* [91], is the *recovery journal*. The journal has to be maintained by each participant and it is of a concern, whether the journal could be implemented on a microcontroller or it requires a larger amount of RAM. The function of the *recovery journal* in *RTP-MIDI* is to keep the history of events and, if a packet had been lost, prevent *stuck notes* in the musical performance.

This protocol is currently implemented as part of *Apple CoreMidi* [167] and an open-source library exists for Linux [89]. However, there had been major adoption for the *RTP-MIDI* in the field.

6.5.3 MIDI Networking in Linux

On Linux, networking of MIDI clients can be implemented in user-space with *ALSA ("Advanced Linux Sound Architecture")* utilities package [1]. The documentation on *ALSA Sequencer Network* is limited to the comments in the source code of the `aseqnet` [83] program. It is design using UDP and a specific data structure. The two programs mentioned in the previous section do not communicate with `aseqnet` [83], but use the *ALSA Sequencer* interface directly.

It would reasonable to implement the same protocol which *ALSA Sequencer Network* is using, however this will be incompatible with other operating systems.

6.6 Conclusion

The final decision had been to apply the simples data structure, i.e. *raw MIDI bitstream*, since it is more important to prove the concept before progressing towards an implementation of a more advanced protocol.

6.7 Test Results

The most generic test that provides nearly ultimate performance measure for an IP network, is the *ping* test. No other particular formal tests were performed during the work on this project, however some possible test scenarios had been considered. The results obtained from several ping tests are presented first, then other basic tests are described.

6.7.1 Latency Measurement: Idle

Ping tests, while the devices are idle, had been made first. These are summarised in terms of packet loss, and minimum/average/maximum ratio. A total of 8 tests had been made on two Linux hosts with different version of Linux kernel and different generation of commodity hardware (`host2` being a desktop PC from 2005 and `host1` being a laptop PC from 2009). As shown below, only at one test there had been 1% packets lost in transmission (0% at all other tests).

The tests done from `host1` shall be presented first. Being a newer generation laptop with more recent kernel, it appears to have faster USB interface (being either due to the chipset or the driver). It has also been noted that sending the code to the development board takes a shorter time the on `host2`.

```
* ping_idle_e1_set1: 64 bytes (host1 <=> econotag1)
128 packets transmitted, 128 received, 0% packet loss, time 127123ms
rtt min/avg/max/mdev = 31.592/32.374/33.810/0.534 ms

* ping_idle_e1_set2: 24 bytes (host1 <=> econotag1)
128 packets transmitted, 128 received, 0% packet loss, time 127210ms
rtt min/avg/max/mdev = 21.617/22.517/23.810/0.544 ms

* ping_idle_f1_set1: 64 bytes (host1 <=> freescale-usb1)
128 packets transmitted, 128 received, 0% packet loss, time 127183ms
rtt min/avg/max/mdev = 20.707/21.775/25.786/0.688 ms

* ping_idle_f1_set2: 24 bytes (host1 <=> freescale-usb1)
128 packets transmitted, 128 received, 0% packet loss, time 127196ms
rtt min/avg/max/mdev = 13.222/14.530/16.305/0.544 ms
```

The above results show that the `freescale-usb1` device is responding faster, since it is connected directly to the host. Also, due to the device being idle, there is not fluctuation in the timing of these responses.

Below are the results obtained from the `host2`, which appear to be generally slower by 10 milliseconds. And there had been a packet lost in `ping_idle_e1_set3`.

```
* ping_idle_e1_set3: 64 bytes (host2 <=> econotag1)
128 packets transmitted, 126 received, 1% packet loss, time 127237ms
rtt min/avg/max/mdev = 41.780/42.089/44.828/0.585 ms
```

```
* ping_idle_e1_set4: 24 bytes (host2 <=> econotag1)
128 packets transmitted, 128 received, 0% packet loss, time 127262ms
rtt min/avg/max/mdev = 34.896/35.919/40.954/0.673 ms

* ping_idle_f1_set3: 64 bytes (host2 <=> freescale-usb1)
128 packets transmitted, 128 received, 0% packet loss, time 127322ms
rtt min/avg/max/mdev = 31.016/35.269/46.437/2.706 ms

* ping_idle_f1_set4: 24 bytes (host2 <=> freescale-usb1)
128 packets transmitted, 128 received, 0% packet loss, time 127355ms
rtt min/avg/max/mdev = 25.006/27.833/33.993/1.443 ms
```

The idle test results show the precise measurement of the potential network performance. A few improvements can be made here, the kernel on the `host2` needs to be updated and tested afterwards. Also, the border router program should be tested with higher baud rate settings and flow-control enabled on UART1; however, for the time being, this was kept unmodified to prevent any unexpected behaviour.

6.7.2 Latency Measurement: Glitch

Much less ideal results had been found in non-idle operation mode, that is when the `econotag1` had been receiving interrupts on UART2 and copying the buffer without proceeding to transfer the data, before the TCP connection had been established. The *"glitch"* results are shown in 6.5[7].

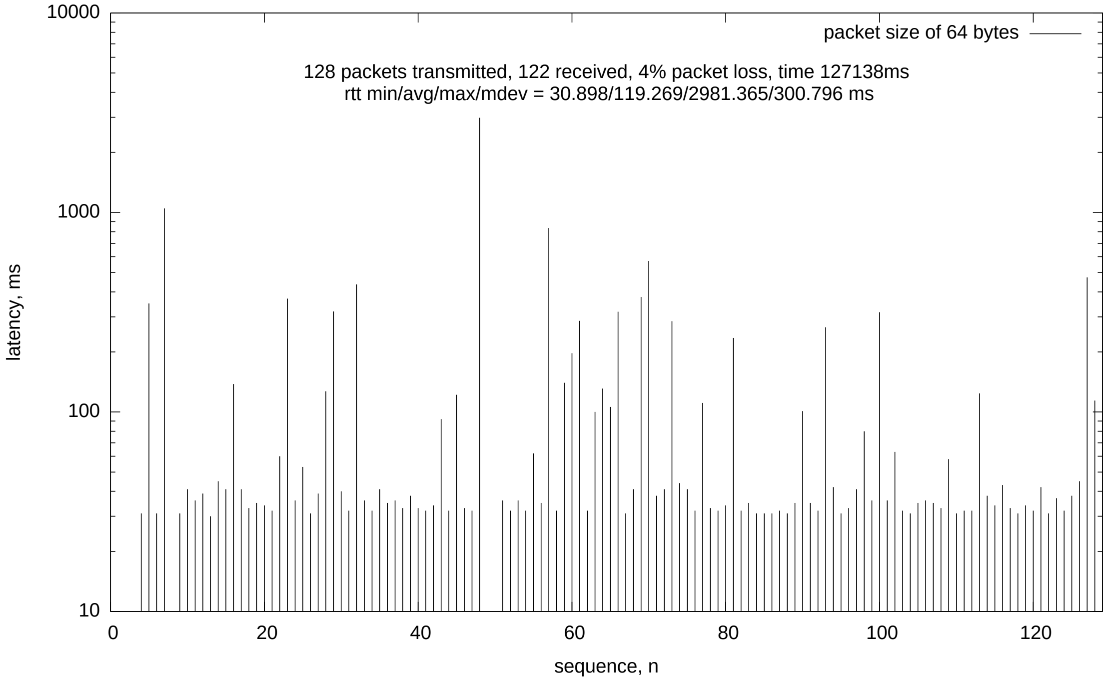
The stability of the idle latency leads to consideration of the non-idle response being a subject to event-driven design of the *uIP* stack and *Contiki OS* in general. It is reasonable to propose that non-idle ping response can be disregarded, since there is no practical purpose for it to be prioritised.

6.7.3 Other Test Scenarios

It had been desired to set-up a test, where one MIDI device would send data to the same host via a wireless link and a wired MIDI interface simultaneously. However, wireless transceiver would need to provide packet timestamping, otherwise no precise measurement can be taken. This facility could be added to *MC1322x* driver code, though it has not yet been studied how this would need to be integrated. Additionally, synchronisation of the timer on the node and the clock of the host system is necessary, which largely extends the complexity of the problem. Without synchronisation protocol there is no particular use for hardware packet timestamps and vise versa.

Few tests were performed with a scripts written in Ruby programming language, the source code of these scripts can found in the project repository under directory `projects/wmi/native/tests/`.

[7] *These results are deliberately presented on separate graphs for clarity. Even if simultaneous ping tests were to be attempted, those may not be combined in one-to-one graphs, unless the size of the network had been greater then 2 nodes. Then it would be appropriate to consider overlaying the datasets based on latency intervals, instead of sequential timebase as presented here.*



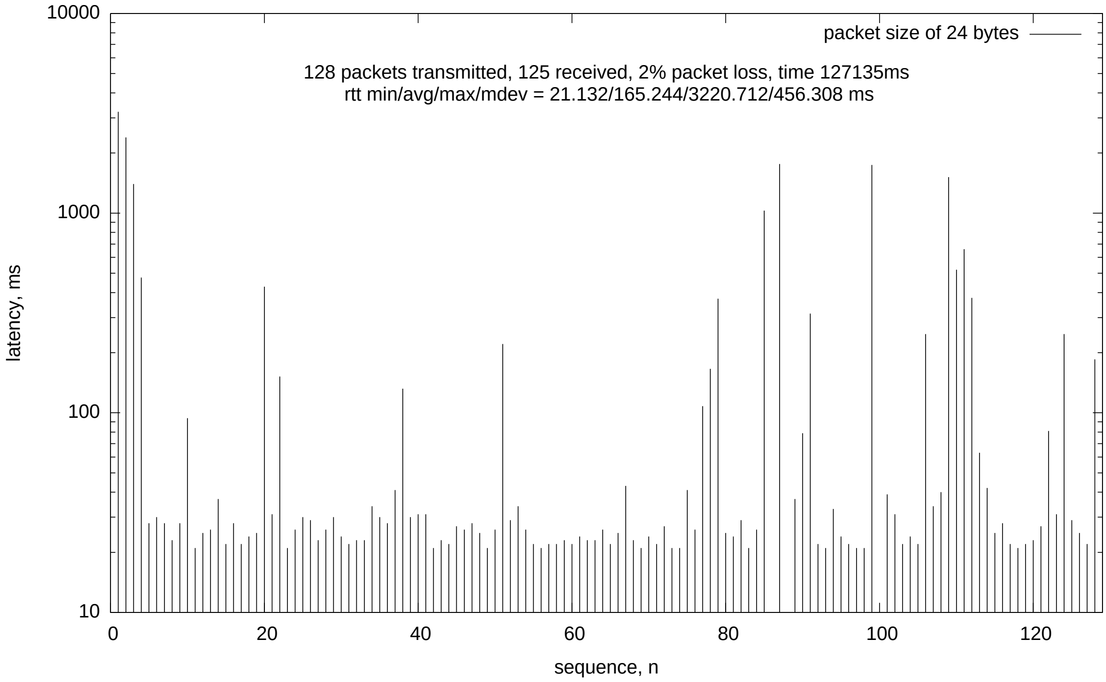


Figure 6.5: *Non-idle ping test response (64 and 24 byte packets)*

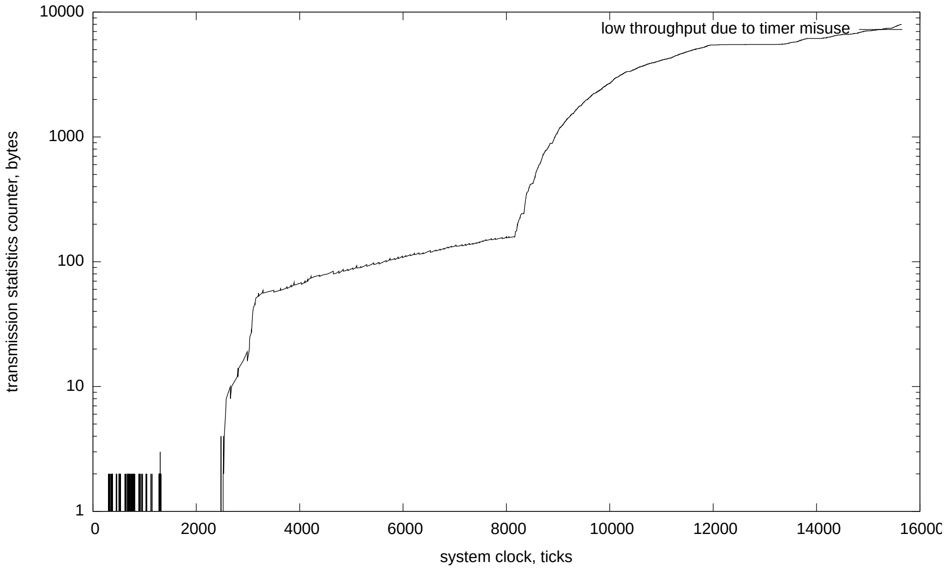


Figure 6.6: *Throughput failure (accumulating untransmitted bytes)*

Throughput Observations

An attempt has been made to measure relative throughput of wireless MIDI link, it would be quite difficult to measure this precisely and more work is required to find out the most accurate way to calculate exact figures of bytes arriving from the MIDI port and bytes received on the other end of the wireless link per unit time. As said above, precision is an issue due to the lack of support for packet timestamping in the current code. First estimations of the performance was certainly a difficult task. The reason why this has been difficult is because the estimated figures did not match any expectations and it had been difficult to compare results between subsequent tests. A graph in figure 6.6 shows the worst case of system throughput. This measurements were take once before improved algorithm with additional polling callback was implemented.

With the help of these test results a bug was found and eliminated. This problem appeared to be due to needless and misused timer in the application code and missing callback to poll the TCP thread. Nevertheless, at first this was misunderstood as a limitation in the *uIP* stack or the capabilities of the device under test. Further test has proven that performance of the system is adequate. Figures 6.7 demonstrate this, though the time base is rather relative, it serves the purpose for this test. The statistics counters were omitted from the listing 6.1 for clarity. The current implementation can be viewed in the repository (`projects/wmi/mc1322x/midi/psock-talker.c`).

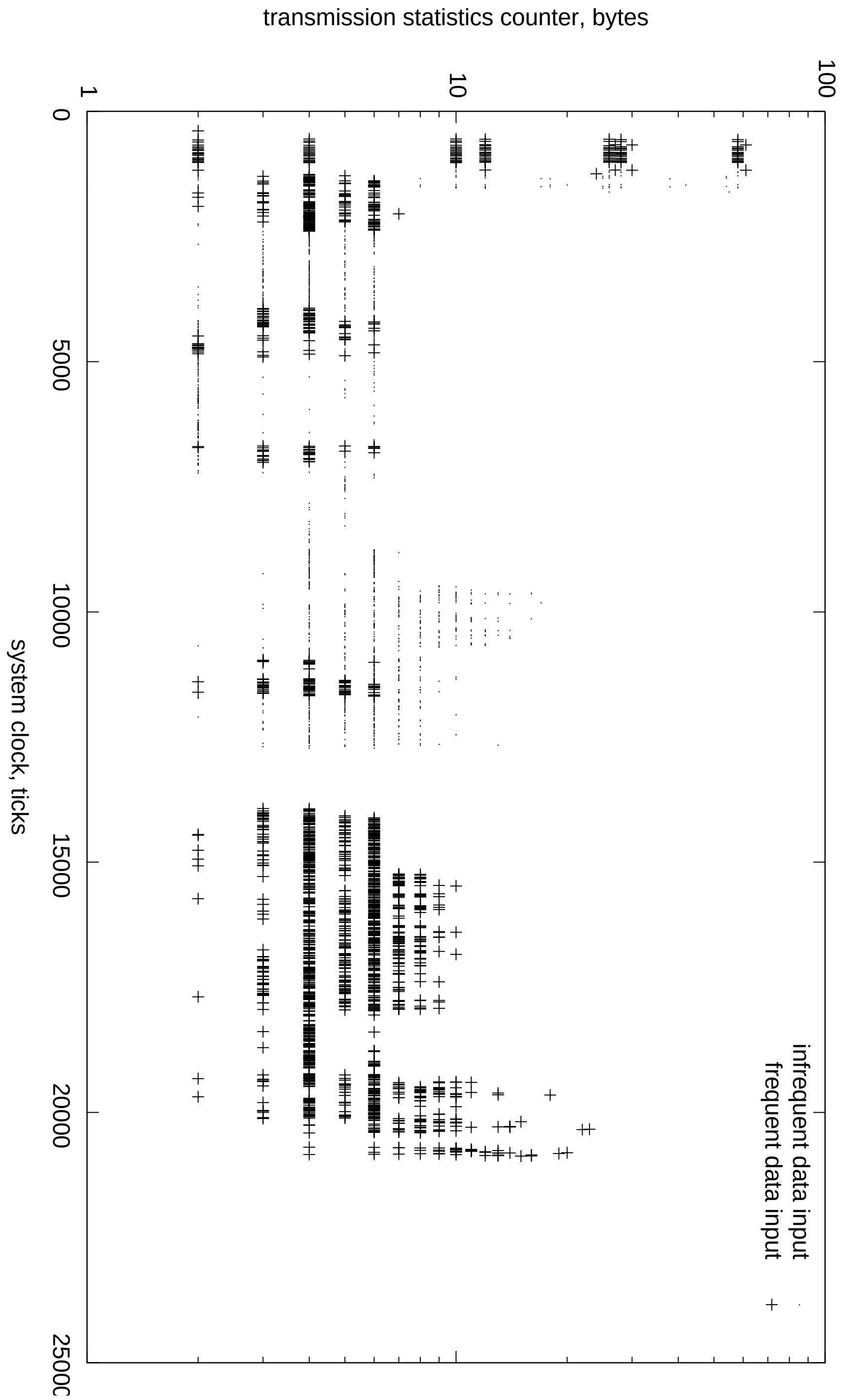


Figure 6.7: *Normal throughput results (two tests combined)*

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

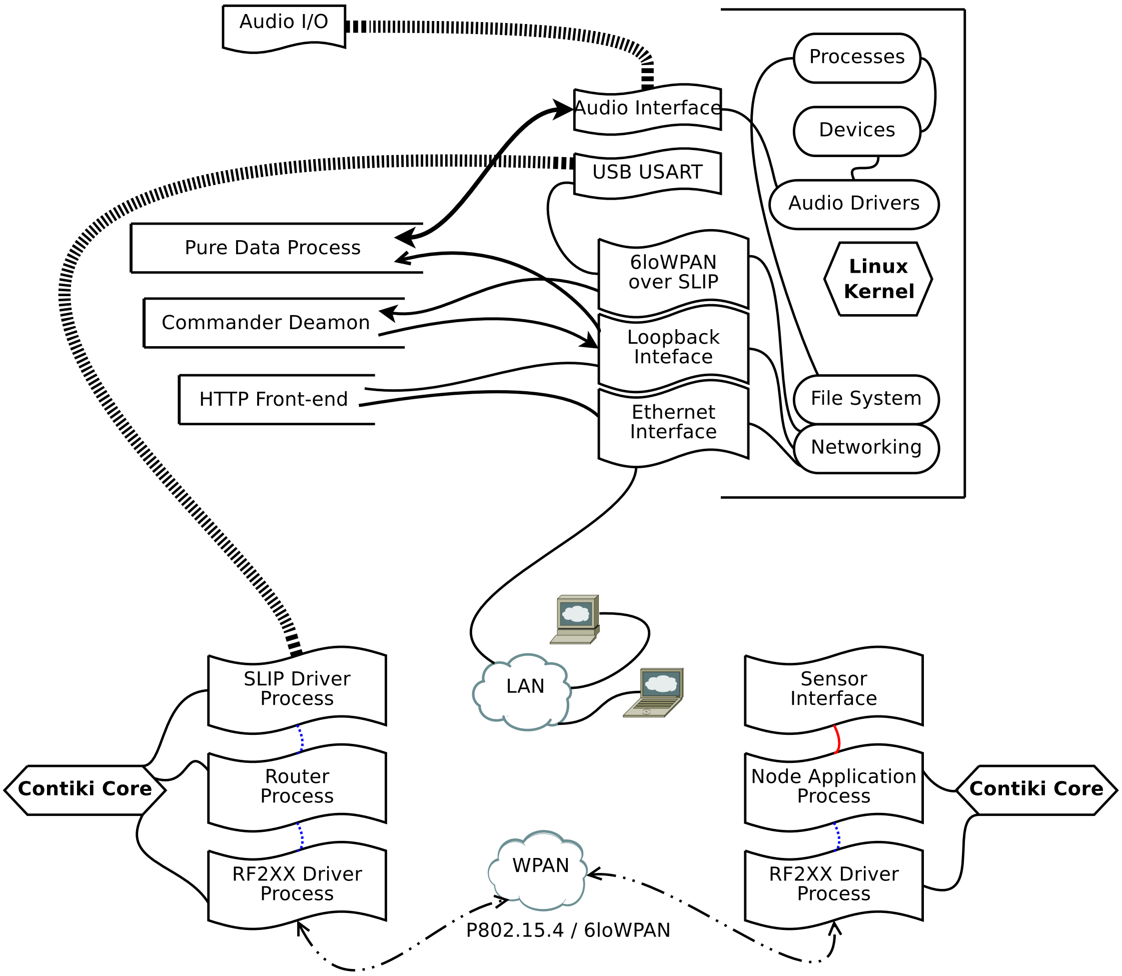


Figure 7.1: *Abstract Sketch of the Development System*

7.2 Aspects of Final Product Design

Many commercial uses of *LR-WPAN*in the field of audio control were considered. The block diagrams in figure 7.2 briefly illustrate a few interesting solutions. Some of the device proposed here can be used in a variety of applications and others are rather specific to live sound and stage performance.

Current implementation can be used to some extend, however one very important modification of hardware needs to be considered. The connectivity between the host and border router, in current development system, is implement via the USB serial interface. This involves unnecessary hardware and software components and should eliminated from a commercial design. One Solution would to utilise SPI bus, which is available on most of the SoCs. A transceiver can be connected directly to the host SoC and there will need to be a Linux driver for it. The second solution is to implement an abstract Linux/-Contiki framework, such that would use SPI for the datapath and GPIO for the interrupts. In this way, a very robust device can be designed (see the *"Wired Gateway Board"* diagram in figure 7.2).

Another important idea presented in 7.2, is the *"Tiny Node"*. It proposes a general-purpose boar, which would suite a wide range of application. Provided that an appropriate physical connection is defined, this board may have removable sensor part for each of the possible uses.

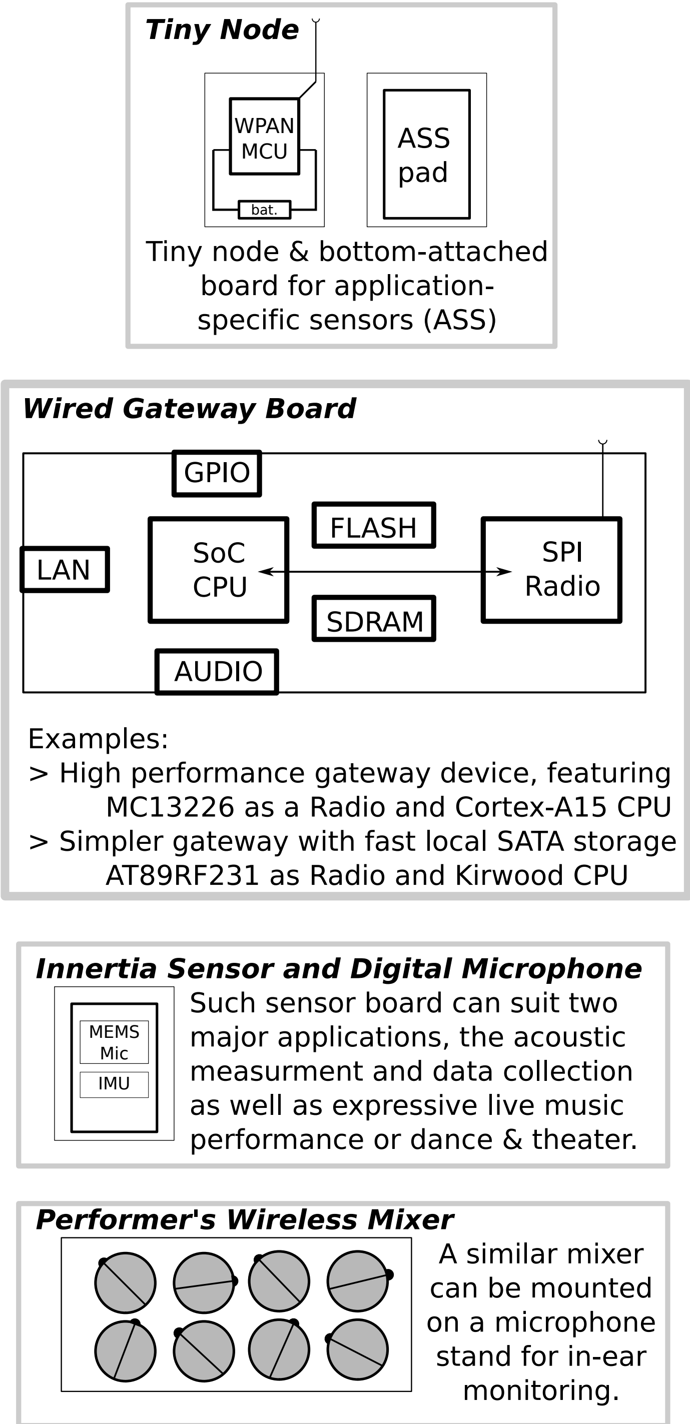


Figure 7.2: *Block diagrams of some possible commercial solutions*

Chapter 8

Looking Ahead

The major emphasis of this project had been in the area of development tools, platforms and code architecture for embedded software design. Though, originally, it was intended as an application development project, study of various alternatives of how the simple task can be achieved, lead to extended research in embedded operating systems and various programming tools, including: compiler toolchains, cross-platform integration techniques as well as the languages. Some of the most relevant aspects had been covered in this report, leaving the unexplored depth behind.

One of initial interest areas had been in the field of networking protocols for wireless sensing and control. Current best practice suggests that IP-enabled networking technology is most suitable for the global adoption, hence simple to integrate with various existing applications. It is rather difficult to imagine where, in the modern technology world, a non-IP network may be useful. Certainly, non-IP networking technology is available on the market, however if such devices are going to be used in any application, there are several limitations imposed on integration into various existing infrastructures. That implies physical connectivity as well as software support. One example is the popular *ZigBee* family of protocols, which is gaining some popularity, thought no global scale of deployment has been observed. Not too long ago, there used to exist quite a few non-IP networking protocols, such as *AppleTalk* and *IPX*, for example. The majority of the published papers are oriented towards the use of IP-enabled networks for wireless sensing, just a few praise proprietary technologies.

The project, which had been described here, has achieved its most fundamental goals, i.e. it has been proven that the MIDI data is handled as expected on a *P802.15.4/6loWPAN* network and the platform had been established for future experimentation. The bigger project is now to commence.

Bibliography

[1] Advanced Linux Sound Architecture. Developer Zone (*http://alsa-project.org/main/index.php/Developer_Zone*).

[2] Analog Devices, Inc. Free Design Library (*http://blackfin.uclinux.org/gf/project/stamp/frs*).

[3] Analog Devices, Inc. *ADMP441 Datasheet (*http://www.analog.com/static/imported-files/data_sheets/ADMP441.pdf*)*.

[4] Analog Devices, Inc. SHARC Processor Family (*http://www.analog.com/static/imported-files/product_highlights/SHARC_Proc_Family_(B)_Final.pdf*).

[5] Anthony Rowe Raj Rajkumar Ryohei Suzuki, R. M. Voice over Sensor Networks (*http://www.nanork.org/raw-attachment/wiki/Pubs/rtss06.pdf?format=raw*).

[6] Damondrick Jack, R. L. Beat Boxing: Expressive Control for Electronic Music Performance and Musical Applications (*citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.59.3707&rep=rep1&type=pdf*).

[7] Dhiraj Goel Raj Rajkumar, A. R. FireFly Mosaic: A Vision-Enabled Wireless Sensor Networking System (*http://www.nanork.org/attachment/wiki/Pubs/rtss-mosaic-07.pdf?format=raw*). Real-Time Systems Symposium (RTSS).

[8] Jennifer Carlile, J. F. SoniMime: Movement Sonifcation for Real-Time Timbre Shaping (*https://ccrma.stanford.edu/~jcarlile/250a/fox-carlile-nime05.pdf*).

[9] John R. McCarty Jeffrey Traer Bernstein, P. S. Muggling! A Wireless Spherical Multi-Axis Musical Controller (*https://ccrma.stanford.edu/~jmccarty/Muggling.pdf*).

[10] Rahul Mangharam Raj Rajkumar, A. R. FireFly: A Time Synchronized Real-Time Sensor Networking Platform (*http://www.nanork.org/attachment/wiki/Pubs/nrk-chapter06.pdf?format=raw*). Wireless Ad Hoc Networking: Personal-Area, Local-Area, and the Sensory-Area Networks, CRC Press (Book Chapter).

[11] ARM Holdings PLC. Floating Point Support (*http://www.arm.com/products/processors/technologies/vector-floating-point.php*).

[12] Atmel Corporation. AVR Tools and Software (*http://www.atmel.com/dyn/products/tools.asp?family_id=607*).

[13] Atmel Corporation. *BitCloud ZigBee PRO (*`http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4495`*).*

[14] Atmel Corporation. *AVR2070: Route Under Medium Access Control (RUM) with IPv6 and 6LoWPAN (*`http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4578`*).*

[15] Atmel Corporation. *AVR2025: IEEE 802.15.4-2006 Medium Access Control (MAC) (*`http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4675`*).*

[16] Atmel Corporation. *AVR2102: ZigBee RF4CE Profile (*`http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4712`*).*

[17] Atmel Corporation. *AT86RF231 Datasheet (rev C) (*`http://www.atmel.com/dyn/resources/prod_documents/doc8111.pdf`*)*, September 2009.

[18] Atmel Corporation. *ATmega128RFA1 Preliminary Datasheet (rev A) (*`http://www.atmel.com/dyn/resources/prod_documents/doc8266.pdf`*)*, December 2009.

[19] AVID/M-Audio. Products: MidAir (*http://www.m-audio.com/products/en_us/MidAir.html*).

[20] Beekmans, G. Linux From Scratch (read on-line) (*http://www.linuxfromscratch.org/lfs/read.html*).

[21] Berkeley Wireless Embedded Systems (WEBS). Berkeley Lightweight IP (BLIP) Project Homepage (*http://smote.cs.berkeley.edu:8000/tracenv/wiki/blip*).

[22] Berkeley Wireless Embedded Systems (WEBS). Project Homepage (*http://webs.cs.berkeley.edu/*).

[23] BuildRoot Project. Homepage (*http://buildroot.uclibc.org/*).

[24] CCRMA. Homepage (*https://ccrma.stanford.edu/*).

[25] CME. Products: WiDi (*http://www.cme-pro.com/en/product-list.php?bc=6*).

[26] CNMAT. Homepage (*http://cnmat.berkeley.edu/*).

[27] Contiki. Homepage at the Swedish Institute of Computer Science (SICS) (*http://www.sics.se/contiki/*).

[28] Contiki. Conding Conventions (*http://www.sics.se/contiki/developers/contiki-coding-conventions.html*).

[29] Contiki. Setting-up Eclipse for Contiki Development (*http://www.sics.se/contiki/wiki/index.php/Setting_up_Eclipse_for_Contiki_Development*).

[30] CreativeCommons.org. Attribution Share-Alike Licence (version 3.0) (*http://creativecommons.org/licenses/by-sa/3.0/*).

[31] Dark Matter Audio, Ltd. Homepage (*http://www.darkmatteraudio.com/*).

[32] Dawson-Haggerty, S. BLIP 2.0 Platform Support Guide (*http://docs.tinyos.net/index.php/BLIP_2.0_Platform_Support_Guide*).

[33] de Campos Bartolomeu, P. J. Evaluating Bluetooth for the wireless transmission of MIDI (*http://www.ieeta.pt/~bartolomeu/docs/bartolomeu-msc-dissertation.pdf*). Master's thesis, Universidade de Aveiro, Departamento de Electrnica e Telecomunicaes, 2005. *IEEE paper reference 10.1109/ETFA.2005.1612507.*

[34] devl.org. Contiki *MC1322x* Port (Homepage) (*http://mc1322x.devl.org/*).

[35] devl.org. Setting-up Eclipse with OpenOCD for source-level debugging (*http://mc1322x.devl.org/eclipse.md*).

[36] devl.org. Contiki *MC1322x* Port (Development Hardware) (*http://mc1322x.devl.org/hardware.html*).

[37] devl.org. Setting-up OpenOCD with the Econotag (*http://mc1322x.devl.org/openocd.html*).

[38] devl.org. Setting-up GCC toolchain for *MC1322x* (*http://mc1322x.devl.org/toolchain.md*).

[39] Digi International, Inc. Product Catalogue: Connect-ME (*http://www.digi.com/products/wireless-wired-embedded-solutions/solutions-on-module/digi-connect/digiconnectme.jsp*).

[40] Digi International, Inc. Product Range: Mesh Wireless Networking (*http://www.digi.com/products/wireless/zigbee-mesh/*).

[41] Digi International, Inc. Getting Started Guide to XBee Programming (*http://www.digi.com/wiki/developer/index.php/Programmable_XBee_-_Getting_Started*).

[42] Dmitrichenko, I. WMI Project Wiki: ARM Wireless SoCs (*http://wmi.new-synth.info/projects/wmi/wiki/ARM_Wireless_SoCs*).

[43] Dmitrichenko, I. WMI Project Wiki: ATMEL Software Stacks (*http://wmi.new-synth.info/projects/wmi/wiki/ATMEL_Software_Stacks*). *(work in progress).*

[44] Dmitrichenko, I. WMI Project Wiki: Bibliography (*http://wmi.new-synth.info/projects/wmi/wiki/Bibliography"*).

[45] Dmitrichenko, I. WMI Wiki: Development Hardware (*http://wmi.new-synth.info/projects/wmi/wiki/Development_Hardware_Options*).

[46] Dmitrichenko, I. WMI Project Wiki: Embedded Operating Systems (*http://wmi.new-synth.info/projects/wmi/wiki/Embedded_Operating_System*). *(work in progress)*.

[47] Dmitrichenko, I. WMI Project Wiki: On-Line Refences (*http://wmi.new-synth.info/projects/wmi/wiki/Links*).

[48] Dmitrichenko, I. WMI Project Wiki: Other Platforms (*http://wmi.new-synth.info/projects/wmi/wiki/Other_Platforms*).

[49] Dmitrichenko, I. Wireles Music Instruments Project (Interim Report), 2010.

[50] Dresden Elktronik. Radio Controller Board RCB128RFA1 (ver 6.3.1) (*http://www.dresden-elektronik.de/shop/prod128.html?language=en*).

[51] Dresden Elktronik. Sensor Terminal Board RCB128RFA1 (ver 2.0) (*http://www.dresden-elektronik.de/shop/prod75.html?language=en*).

[52] Dunkels, A. Contiki: On-line Reference Manual (v2.4) (*http://www.sics.se/~adam/contiki/docs/*).

[53] Dunkels, A. uIP: On-line Reference Manual (v1.0) (*http://www.sics.se/~adam/uip/uip-1.0-refman/*).

[54] 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).

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

[56] 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.

[57] Dunkels, A., Finne, N., Eriksson, J., and Voigt, T. Run-time dynamic linking for reprogramming wireless sensor networks. In *Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys 2006)* (Boulder, Colorado, USA, Nov. 2006).

[58] 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).

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

[60] 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).

[61] Duta, P. Homepage (*http://www.cs.berkeley.edu/~prabal/*).

[62] Duta, P. Epic Mote Project Homepage (*http://www.cs.berkeley.edu/~prabal/projects/epic/*).

[63] 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 (*http://www.cs.berkeley.edu/~stevedh/pubs/p1-Dutta.pdf*). In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys'10)* (November 2010).

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

[65] Freed, A., and Schmeder, A. Features and Future of Open Sound Control version 1.1 (*http://cnmat.berkeley.edu/system/files/attachments/Nime09OSCfinal.pdf*).

[66] Freed, A., and Wright, M. Open Sound Control: A New Protocol for Communicating with Sound Synthesizers (*http://archive.cnmat.berkeley.edu/ICMC97/postscript/0118030b.org-OSC-paper.ps*).

[67] Freescale Semiconductors. i.MX Product Family Overview (*http://cache.freescale.com/files/32bit/doc/brochure/FLYRIMXPRDCMPR.pdf*).

[68] Freescale Semiconductors. i.MX5 Fact Sheet (*http://cache.freescale.com/files/32bit/doc/fact_sheet/IMX5CNFS.pdf*).

[69] Freescale Semiconductors. Application Note AN3751: Frequency Analysis in the Industrial Market Using Accelerometer Sensors (*http://cache.freescale.com/files/sensors/doc/app_note/AN3751.pdf*).

[70] Freescale Semiconductors. *i.MX32 Reference Manual (*http://www.freescale.com/files/32bit/doc/ref_manual/MCIMX31RM.pdf*).*

[71] Freescale Semiconductors. *1322XUSB Reference Manual (*http://www.freescale.com/files/rf_if/doc/ref_manual/1322xUSBRM.pdf*).*

[72] Freescale Semiconductors. *MC1322X Reference Manual (*http://www.freescale.com/files/rf_if/doc/ref_manual/MC1322xRM.pdf*).*

[73] Future Technology Devices International, Ltd. *FT2232H Dual High Speed USB to Multipurpose UART/FIFO IC (*http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT2232H.pdf*)*, 2010.

[74] Gentoo Embedded Platfrom Development Team. Embedded Handbook (*http://www.gentoo.org/proj/en/base/embedded/handbook/*).

[75] Gont, F., and Yourtchenko, A. On the implementation of the TCP urgent mechanism (*http://tools.ietf.org/pdf/draft-ietf-tcpm-urgent-data-07*). In *Internet Drafts* (October 2010), IETF.

[76] Hope Microelectronics. Product Range: 2.4GHz Transceivers (*http://www.hoperf.com/rf_fsk/*).

[77] IEEE 802.15 Wireless Personal Area Networks. Standard Homepage (*http://www.ieee802.org/15/*).

[78] IEEE 802.15 WPAN Task Group 4. TG4 Homepage (*http://www.ieee802.org/15/pub/TG4.html*).

[79] Incudetech. WiMi: Xbee Midi Development Board (*http://www.includetech.com/default.aspx?pg=40eacac0-5810-421e-9ed9-355f08ee115c*).

[80] ISPO Alliance. Homepage (*http://ipso-alliance.org/*).

[81] ISPO Alliance. Incorporating IEEE 802.15.4 into the IP architecture (*http://ipso-alliance.org/resource-library*).

[82] ISPO Alliance. IP for Smart Objects (*http://ipso-alliance.org/resource-library*). *Original version September 2008.*

[83] Iwai, T. *aseqnet (command manual) (*http://manpages.debian.net/cgi-bin/man.cgi?query=aseqnet&sektion=0&format=ascii*).*

[84] JazzMutant. Products: Lemur (*http://www.jazzmutant.com/lemur_overview.php*).

[85] Jordan, R. Sensory Response Systems (*http://ryanjordan.org/SRSThesis_Ryan_Jordan_2009.pdf*). Master's thesis, Goldsmiths Digital Studios, University of London, 2009.

[86] Kenton Electronics, UK. Products: MidiStream (*http://www.kentonuk.com/products/items/wireless/midistream.shtml*).

[87] 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).

[88] Ladyada.net. Tutorial: Xbee Midi Link (*http://www.ladyada.net/make/xbee/midibee.html*).

[89] Lazzaro, J., and Wawrzynek, J. RTP MIDI: An RTP Payload Format for MIDI (reference desing) (*http://www.cs.berkeley.edu/~lazzaro/rtpmidi/*).

[90] Lazzaro, J., and Wawrzynek, J. An Implementation Guide for RTP MIDI (*http://tools.ietf.org/pdf/rfc4695.pdf*). In *Internet Standards* (November 2006).

[91] Lazzaro, J., and Wawrzynek, J. RTP Payload Format for MIDI (*http://tools.ietf.org/pdf/rfc4695.pdf*). In *Internet Standards* (November 2006).

[92] Lenzen, C., Locher, T., Sommer, P., and Wattenhofer, R. Clock Synchronization: Open Problems in Theory and Practice (*http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.148.75&rep=rep1&type=pdf*). In *36th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM), pindlerv Mln, Czech Republic* (January 2010).

[93] Lenzen, C., Sommer, P., and Wattenhofer, R. Optimal Clock Synchronization in Networks (*http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.147.8792&rep=rep1&type=pdf*). In *7th ACM Conference on Embedded Networked Sensor Systems (SenSys), Berkeley, California, USA* (November 2009).

[94] Levis, P., and Gay, D. *TinyOS Programming (*http://sl.stanford.edu/~pal/pubs/tos-programming-web.pdf*)*. Cambridge University Press, 2009.

[95] Levis, P., Tavakoli, A., and Dawson-Haggerty, S. Overview of Existing Routing Protocols for Low Power and Lossy Networks (*http://tools.ietf.org/pdf/draft-ietf-roll-protocols-survey-07.pdf*). In *Internet Drafts* (October 2009), IETF.

[96] Linaro Foundation. Homepage (*http://www.linaro.org/about-linaro/*).

[97] Linux for Devices News Portal. Reference Guide on Single Board Computer Systems (*http://www.linuxfordevices.com/c/a/Linux-For-Devices-Articles/Single-Board-Computer-SBC-Quick-Reference-Guide/*).

[98] Livid. Products: Block64 (*http://www.lividinstruments.com/hardware_ohm64.php*).

[99] Ma, Y., and Wobschall, D. Synchronization of Wireless Sensor Networks Using a Modified IEEE 1588 Protocol (*http://eesensors.com/pdf/ES25_WTM1588.pdf*). In *International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication (ISPCS)* (October 2010).

[100] Maroti, M., and Biro, A. Source Code Repository of ATmega128RFA1 port for TinyOS (*http://szte-wsn.svn.sourceforge.net/viewvc/szte-wsn/tinyos-2.x/trunk/*).

[101] Marvell Semiconductor, Inc. Plug Computer Platform (*http://www.marvell.com/platforms/plug_computer/*).

[102] MARVELL SEMICONDUCTOR, INC. ARMAD Processor Family (*http://www.marvell.com/products/processors/armada.html*).

[103] MARVELL SEMICONDUCTOR, INC. Kirkwood Processor Family (*http://www.marvell.com/products/processors/embedded/kirkwood/*).

[104] MENTOR GRAPHICS CORPORATION. Mentor Embedded Linux - Technical Brief (*http://www.mentor.com/embedded-software/upload/embedded-linux-brief.pdf*).

[105] MONOME. Details of OSC implementation in MONOME hardware (*http://docs.monome.org/doku.php?id=tech:protocol:osc*).

[106] MONOME. Homepage (*http://monome.org*).

[107] MORSE, D., MCGRATH, R., AND FRYSINGER, M. *make (command manual) (*http://manpages.debian.net/cgi-bin/man.cgi?query=make&sektion=1&format=ascii*)*.

[108] MULTIMIDICAST. Project Homepage (*http://llg.cubic.org/tools/multimidicast/*).

[109] NANO-RK. Research Publication (*http://nano-rk.org/wiki/Pubs*).

[110] NERDS.DE. ipMIDI software for Windows and Mac OS (*http://nerds.de/*).

[111] NETLOGIC MICROSYSTEMS. Multi-core MIPS64 Processor Family (*http://www.netlogicmicro.com/Products/MultiCore/*).

[112] NOBLE, J. *Programming Interactivity: A Designer's Guide to Processing, Arduino, and OpenFrameworks.* O'Reilly Series. O'Reilly, 2009.

[113] NORDIC SEMICONDUCTORS. Product Range: 2.4GHz Transcievers (*http://www.nordicsemi.com/index.cfm?obj=menu&act=displayMenu&men=2*).

[114] OPENEMBEDDED PROJECT DOCUMENTATION. *User Manual (*http://docs.openembedded.org/usermanual/usermanual.pdf*)*.

[115] PATCHMAN MUSIC. Products: MidiJet (*http://www.patchmanmusic.com/MidiJetPro.html*).

[116] PHIDGETS, INC. Homepage (*http://www.phidgets.com/*).

[117] PHIDGETS, INC. Products: SBC (*http://www.phidgets.com/products.php?category=21&product_id=1070*).

[118] PLUG COMPUTER COMMUTY. Homepage (*http://www.plugcomputer.org/*).

[119] POSTEL, J. (EDITOR). Transmission Control Protocol (*http://tools.ietf.org/pdf/rfc0793.pdf*). In *Internet Standards* (September 1981), IETF.

[120] Puckette, M. S. Pure Data Software (*http://crca.ucsd.edu/~msp/software.html*).

[121] qmidinet. Project Homepage (*http://qmidinet.sourceforge.net/qmidinet-index.html*).

[122] Redwire, LLC. Econotag Development Boar (design files, revision 2) (*http://mc1322x.devl.org/files/econotag-r2-release.zip*).

[123] Rowe, A., Mangharam, R., and Rajkumar, R. RT-Link: A Time-Synchronized Link Protocol for Energy Constrained Multi-hop Wireless Networks (*http://www.ece.cmu.edu/~agr/pubpg/secon_06.pdf*).

[124] ScratchBox Project. Homepage (*http://www.scratchbox.org/*).

[125] Seeed Studio. Products: Seeeduino Film (*http://www.seeedstudio.com/depot/seeeduino-film-v09b-p-689.html*).

[126] Sense/Stage. Homepage (*http://sensestage.hexagram.ca/*).

[127] Sense/Stage. MiniBee Hardware (*http://sensestage.hexagram.ca/topic/documentation/hardware/minibee*).

[128] Sense/Stage. Software Clients (*http://sensestage.hexagram.ca/topic/documentation/software/clients*).

[129] Sjuve, E. Prototype GO: Wireless Controller for Pure Data (*http://nime2008.casapaganini.org/documents/Proceedings/Demos/156.pdf*).

[130] Solis, R., Borkar, V., and Kumar, P. A New Distributed Time Synchronization Protocol for Multihop Wireless Networks (*http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.124.1230&rep=rep1&type=pdf*). In *45th IEEE Conference on Decision and Control* (December 2006).

[131] Sommer, P. Source Code Repository of ATmega128RFA1 port for TinyOS (*https://github.com/phsommer/tinyos-main/*).

[132] Sommer, P., and Wattenhofer, R. Symmetric Clock Synchronization in Sensor Networks (*http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.139.5095&rep=rep1&type=pdf*). In *ACM Workshop on Real-World Wireless Sensor Networks (REALWSN), Glasgow, Scotland* (April 2008).

[133] Sommer, P., and Wattenhofer, R. Gradient Clock Synchronization in Wireless Sensor Networks (*http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.146.2537&rep=rep1&type=pdf*). In *8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), San Francisco, USA* (April 2009).

[134] Stallman, R. M., McGrath, R., and Smith, P. D. *GNU Make: A Program for Directing Recompilation (*http://www.gnu.org/software/make/manual/make.pdf*).*

[135] Stallman, R. M., Pesch, R., and Shebs, S. *Debugging with gdb - The GNU Source-Level Debugger (*http://www.gnuarm.com/pdf/gdb.pdf*).*

[136] Stallman, R. M., and Pesch, R. H. *gdb (command manual) (*http://manpages.debian.net/cgi-bin/man.cgi?query=gdb&sektion=0&format=ascii*).*

[137] Sundararaman, B., Buy, U., and Kshemkalyani, A. D. Clock synchronization for wireless sensor networks: a survey (*http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.135.1746&rep=rep1&type=pdf*), 2005.

[138] TEP Document. Resource Arbitration (*http://www.tinyos.net/tinyos-2.x/doc/pdf/tep108.pdf*).

[139] TEP Document. Sensors and Sensor Boards (*http://www.tinyos.net/tinyos-2.x/doc/pdf/tep109.pdf*).

[140] TEP Document. Creating a New Platform for TinyOS 2.x (*http://www.tinyos.net/tinyos-2.x/doc/pdf/tep131.pdf*).

[141] TEP Document. Packet timestamping (*http://www.tinyos.net/tinyos-2.x/doc/pdf/tep132.pdf*).

[142] TEP Document. Packet-level time synchronization (*http://www.tinyos.net/tinyos-2.x/doc/pdf/tep133.pdf*).

[143] TEP Document. TEP Structure and Keywords (*http://www.tinyos.net/tinyos-2.x/doc/pdf/tep1.pdf*).

[144] TEP Document. Hardware Abstraction Architecture (*http://www.tinyos.net/tinyos-2.x/doc/pdf/tep2.pdf*).

[145] Texas Instruments. OMAP Documentation Wiki: DSP Brige Project (*http://www.omappedia.org/wiki/DSPBridge_Project*).

[146] Texas Instruments. OMAP Documentation Wiki: Panda Board Spec. (*http://www.omappedia.org/wiki/PandaBoard*)

[147] Tinker.it. Arduino Library (*http://arduino.cc/en/Tutorial/Foundations*).

[148] Tinker.it. Arduino Homepage (*http://www.arduino.cc/*).

[149] TinyOS. Network Simulator TOSSIM (*http://docs.tinyos.net/index.php/TOSSIM*).

[150] TinyOS. nesC Language Project Homepage (*http://nescc.sourceforge.net/papers/nesc-ref.pdf*).

[151] TinyOS Wiki. Documentation Resources (*http://docs.tinyos.net/index.php/Main_Page*).

[152] TinyOS.net. Project Homepage (*http://www.tinyos.net/*).

[153] 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).

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

[155] Watt, D., and Osborne, R. XMOS Tools Developer Guide (*http://www.xmos.com/published/tools-development-guide*).

[156] Wikipedia. 6loWPAN (*http://en.wikipedia.org/wiki/6LoWPAN*).

[157] Wikipedia. ANT Network (*http://en.wikipedia.org/wiki/ANT_(network)*).

[158] Wikipedia. Carrier Sense Multiple Access with Collision Avoidance (*http://en.wikipedia.org/wiki/Carrier_sense_multiple_access_with_collision_avoidance*).

[159] Wikipedia. Creative Commons Licenses (*http://en.wikipedia.org/wiki/Creative_Commons_licenses*).

[160] Wikipedia. DASH7 (*http://en.wikipedia.org/wiki/DASH7*).

[161] Wikipedia. Direct-Sequence Spread Spectrum (*http://en.wikipedia.org/wiki/Direct-sequence_spread_spectrum*).

[162] Wikipedia. IEEE 802.15 - Wireless PAN (Personal Area Network) (*http://en.wikipedia.org/wiki/IEEE_802.15*).

[163] Wikipedia. IEEE 805.15.1 (Medium-Rate WPAN) / BlueTooth (*http://en.wikipedia.org/wiki/IEEE_802.15.1*).

[164] Wikipedia. IEEE 802.15.4 (Low-Rate WPAN) (*http://en.wikipedia.org/wiki/IEEE_802.15.4*).

[165] Wikipedia. Real-time Operating System (*http://en.wikipedia.org/wiki/List_of_real-time_operating_systems*).

[166] Wikipedia. Low Level Virtual Machine (*http://en.wikipedia.org/wiki/LLVM*).

[167] Wikipedia. MIDI Usage and Applications: RTP-MIDI (*http://en.wikipedia.org/wiki/MIDI_usage_and_applications\#RTP-MIDI_Transport_Protocol*).

[168] Wikipedia. Quadrature Phase-Shift Keying (*http://en.wikipedia.org/wiki/Phase-shift_keying\#Quadrature_phase-shift_keying_.28QPSK.29*).

[169] Wikipedia. Pure Data (*http://en.wikipedia.org/wiki/Pure_Data*).

[170] Wikipedia. Real-time Operating System (*http://en.wikipedia.org/wiki/Real-time_operating_system*).

[171] Wikipedia. Representational State Transfer (*http://en.wikipedia.org/wiki/Representational_State_Transfer*).

[172] Wikipedia. TinyOS (*http://en.wikipedia.org/wiki/TinyOS*).

[173] Wikipedia. ZigBee (*http://en.wikipedia.org/wiki/ZigBee*).

[174] Yeo, W. S. The Bluetooth Radio Ball Interface (BRBI): A Wireless Interface for Music/Sound Control And Motion Sonification (*https://ccrma.stanford.edu/~woony/publications/Yeo-ICMC06.pdf*).

[175] Zolertia.com. Product Store (*http://shop.zolertia.com/*).