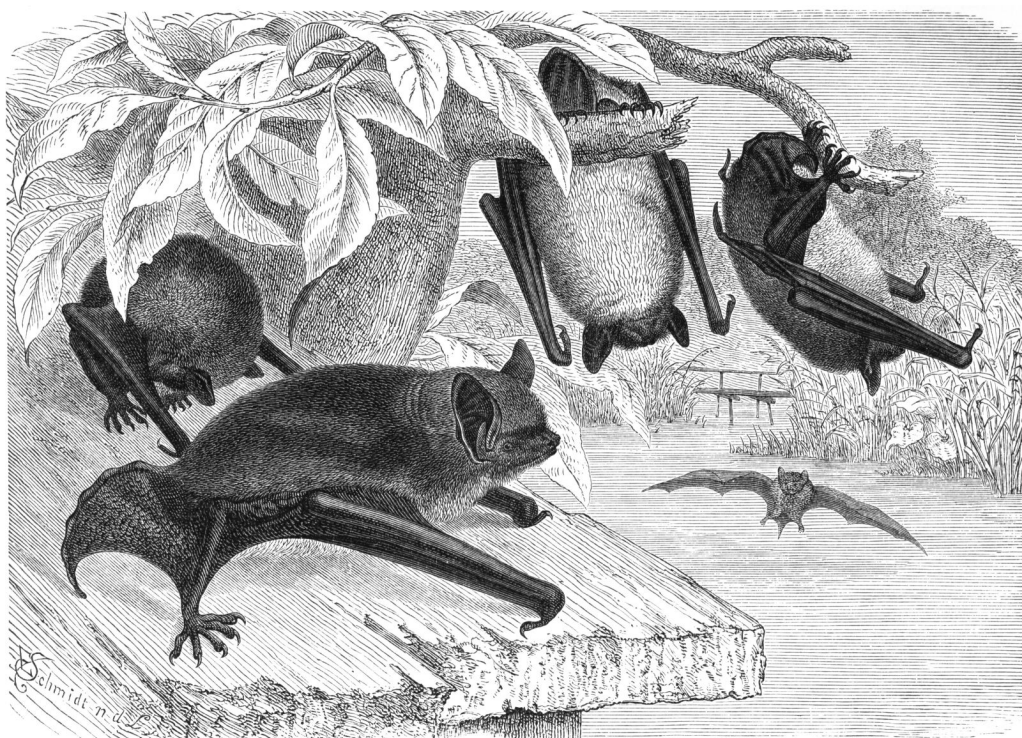


Tracking and synthesis of bat echolocating calls



Author: Tórur Andreassen (130277) *ta@imada.sdu.dk*
Supervisor: Prof. John Hallam *john@mmmi.sdu.dk*

University of Southern Denmark
September 1, 2008

Abstract

The design of a system capable of sampling receivers up to two million samples per second with at least sixteen simultaneous channels is developed. This makes the system well suited for measuring echolocating bats (Chiroptera order) which is an important end-goal of this system. Each individual component is tested, and the practical feasibility of the completed system is examined. Advice on completing a functional prototype is given.

The system is designed to be scalable and modular, and should be able to handle an arbitrary number of receivers. Another aim of this project is to use standard off-the-shelf products as much as possible, i.e. to ensure broad and enduring support regarding hardware, thus reducing price and increasing lifetime of the system. Equivalently the software is, as much as possible, chosen from the open source ranks, this again ensures low cost and longevity, and if we are lucky, free development.

Contents

Preface	viii
Nomenclature	1
I Analysis and design	2
1 Introduction	3
1.1 Inspiration	3
1.2 Relevant non-trivial concepts	3
1.2.1 Echolocation in bats	4
1.2.2 Time difference of arrival	5
1.2.3 Analogue to digital conversion	5
1.3 Specifying project parameters	5
1.3.1 Original specifications	5
1.3.2 Interpreting the original specifications	5
1.4 Setting realistic goals	7
1.4.1 Power consumption and portability	8
1.4.2 Number of microphones and bandwidth	9
1.5 Conclusions about design choices	9
2 Microphone and filter	11
2.1 Microphone	11
2.1.1 Knowles Acoustics SPM-0204-H-E-5	12
2.1.2 Knowles Acoustics SPM-0204-U-D-5	13
2.1.3 Other microphones	13
2.2 Filter	13
2.2.1 Bandpass filters	15
2.2.2 Designing the filter	15
2.3 Concluding the microphone and filter discussion	16
2.3.1 Choosing microphone	16

CONTENTS

2.3.2	Choosing filter design	17
3	Digitising microphone information	18
3.1	Finding product series	18
3.1.1	Criteria for the ADC	18
3.1.2	Bandwidth and latency	19
3.1.3	Driver issues	19
3.1.4	Determine supplier	20
3.2	Finding product series	20
3.2.1	PCI based products	20
3.2.2	PCI Express based products	20
3.3	Choosing product	22
4	Computers involved	23
4.1	Master computer	23
4.2	Node computer	24
4.2.1	Nano-ITX and Pico-ITX	24
4.2.2	Mini-ITX	25
5	Design conclusions	26
5.1	System design	26
5.2	Microphone and filter	26
5.3	A/D-Converter solution	28
5.4	Master and node computer	28
II	Hardware	29
6	Building the filter and microphone	30
6.1	Circuit technology	30
6.2	Choosing the correct tools	31
6.3	Design by construction	31
6.3.1	Filter design	31
6.3.2	Adaptation circuit design	34
6.4	Transferring the design to PCB	37
6.4.1	Schematics to PCB design	37
6.4.2	Producing in-house PCBs	38
6.4.3	Manufacture PCBs	39
6.4.4	Examples of produced PCBs	39
6.5	Conclusions about the built circuit	40

7	Building the node computer	43
7.1	Purchase components	43
7.2	Assembly of the node computer	44
8	Combining hardware components	47
8.1	The connected system	47
8.1.1	Notes on the first prototype	47
8.2	Concluding remarks about the built hardware	49
III	Software	50
9	Operating system on the node computer	51
9.1	Choice of Linux based operating system	51
9.1.1	Criteria for the operating system	52
9.1.2	Choosing distribution	52
9.2	Deployment of software	53
9.2.1	Manual deployment of Debian GNU/Linux	54
9.2.2	Automatic deployment of Debian GNU/Linux	54
9.3	Concluding remarks	58
10	A/D-board software	59
10.1	Driver software	59
10.1.1	Compile kernel module	59
10.1.2	Loading module	60
10.2	Compile module library (API)	61
10.2.1	Compile library	61
10.2.2	Install library	61
10.3	Sampling software	61
10.3.1	What is needed	61
10.3.2	Sampling program implementations	62
10.4	Concluding remarks	64
11	Handling of recorded data	65
11.1	Analysis software	65
11.1.1	Matlab	65
11.1.2	Octave	66
11.1.3	PDL or Perl Data Language	66
11.1.4	Conclusion about analysis software	66
11.2	Signal detection	66
11.2.1	Two layered wavelet tree	66

CONTENTS

11.2.2 Custom algorithm	67
11.3 Synchronisation	67
11.4 Comparing signals	67
11.4.1 Stationary or mobile	67
11.4.2 Directionality	67
11.4.3 Signal echo	68
11.5 Location algorithms	68
11.5.1 Spherical algorithm	68
11.5.2 Hyperbolic method	68
11.6 Synthesis of signals	69
11.7 Auto calibration	69
IV Using the system	70
12 Verification tests	71
12.1 Timer testing	71
12.1.1 Timer based on <code>usleep(3)</code>	71
12.1.2 Program based on <code>nanosleep(3)</code>	73
12.1.3 Timer conclusions	73
12.2 A/D-board	74
12.2.1 Sampling channel	74
12.3 Analysis software	74
12.3.1 Spectrogram generation	74
12.4 Filter verification	79
12.5 Microphone verification	79
13 System testing	80
13.1 Bandwidth testing	80
13.2 Compute time left	80
14 Conclusions	84
14.1 Things to improve	84
14.1.1 Hard drive failure	84
14.1.2 Mini-ITX underpowered	84
14.1.3 Microphone oversupplied	85
14.2 Preliminary final version specifications	85
14.3 Addressing items 8-16	85
14.4 Comparing result to original abstract	85
14.4.1 What needs to be done	86
14.5 Success or failure	86

Bibliography	86
A Open source software choices	97
A.1 PCB production	97
A.1.1 The Kicad design tool	98
A.1.2 The gEDA design tool	98
A.1.3 Concluding notes on design tools	101
B Source files for PCB production	102
B.1 Symbol files for <code>gschem</code>	102
B.1.1 Knowles SMD microphone	103
B.1.2 SMD voltage regulator	104
B.2 Custom PCB footprints	106
B.2.1 Knowles SMD microphone	106
B.2.2 Voltage regulator	107
B.3 Generating printouts for PCB	108
B.3.1 Printout quirks with <code>gschem</code>	112
B.3.2 Printout quirks with <code>pcb</code>	112
C A/D-board source code	115
C.1 Sampling programs	115
C.1.1 Support library	116
C.1.2 Software triggered sampling	121
C.1.3 Double buffered sampling	122
C.1.4 Simple oscilloscope implementation	124
C.1.5 Makefiles	129

List of Figures

1.1	Spectrogram of a Myotis Daubentonii chirp	4
1.2	Overview of the system	8
1.3	Updated of the design of the system	10
2.1	Knowles microphone part numbering	12
2.2	Two different sinusoids that fit the same set of samples	13
2.3	Transfer function plot of a bandpass filter	14
2.4	Biquad filter design	16
5.1	System design decided on	27
5.2	Node design decided on	27
6.1	Filter implemented on a breadboard	32
6.2	Bandpass filter schematic	33
6.3	Power adaptation implemented on a breadboard	35
6.4	Power adaptation schematic	36
6.5	Copper and component masks of the PCB produced	39
6.6	PCB prototype progression	41
7.1	Components of the node computer	45
7.2	Assembly of the node computer	46
8.1	First prototype of the whole system	48
9.1	Picture of 2.5" IDE hard disk connected through USB	55
10.1	Software layers used when sampling	62
12.1	Test of <code>usleep(3)</code> timer routine	72
12.2	Test of <code>nanosleep(3)</code> timer routine	73
12.3	Sampling verification setup, no voltage applied	74
12.4	Verifying that sampling works as expected	75
12.5	Time series of a chirp	76

LIST OF FIGURES

12.6	Chirp spectrogram generated by GNU Octave	76
12.7	Chirp spectrogram generated by the <i>PDL</i> library	77
12.8	Spectrogram of the whole recording	79
A.1	First filter prototype, schematic created with Kicad	99
A.2	First PCB filter prototype, design created with Kicad	100
B.1	gschem SMD microphone symbol	103
B.2	gschem voltage regulator symbol	105
B.3	Footprint for the Knowles microphone	107
B.4	Footprint for the Knowles microphone	107
B.5	Output from running make in <i>filter/</i> directory	111
B.6	Component layout of the PCB	114

List of Tables

1.1	Original specifications for the system	6
1.2	Interpreting original specifications	7
1.3	Comparison of battery types	9
2.1	Filter type benefits and drawbacks	15
3.1	Amplicon PCI based ADC products	21
3.2	Amplicon PCI Express based ADC products	21
4.1	Node computer criteria	24
6.1	Bill of materials for the bandpass filter	34
6.2	Bill of materials for the power adaptation circuit	37
6.3	PCB production steps at the Maersk institute	38
7.1	Purchased items for the node computer	44
7.2	Node computer assembly instructions	45
8.1	Comparing hardware to interpreted specifications	48
9.1	Node computer operating system criteria	52
9.2	Manual install of Debian GNU/Linux 3.1	54
9.3	Node computer needed software packages	56
14.1	Preliminary final version specifications	85

Preface

Reading guidance

This project has been divided into four parts each containing several chapters. Each part holds related subjects, the short description of these parts is as follows: analysis/design, implementation of hardware, implementation of software and tests of the system.

The long version:

- Part I holds an introduction to this project. Also analysis and design decisions are made in this part, thus shaping the focus of the rest of the report.
- Based on the decisions made in Part I, Part II moves forward with the implementation of the hardware choices made in the design sections. These include, but are not limited to, wiring, schematics, which hardware components have been used and how.
- Part III describes in depth what software has been used in this project. There are several areas of interest, e.g. operating system-, analysis- and device driver software.
- After going through the ins and outs of the system, Part IV tests all individual components of the system. Then follow tests of all the components together and thus, hopefully, solving the intended goal, i.e. tracking and synthesis of bat calls.

Multi disciplinary

As this project involves many disciplines and as I am not an expert in all of them, some subjects are not dealt with in as much depth as I would have liked, but I believe the most important issues have been address with enough

depth, and the other issues may be covered at a later time by consulting experts in the relevant fields.

Citing on-line material

During this project many on-line sources have been used. Given that I want to place credit where credit is due, I want to cite those sites/persons which provided the information.

Citing on-line material comes with its own set of issues though as discussed in [1,2] around 1996. Later it seems that the ISO 690* standard was revised in 1997 to include electronic media, so there exists an international standard, I just do not have access to it. Given that it would cost 90 Swiss francs[†] to acquire this standard, I have chosen to mix and match with the MISC entry type in BibTEX [3,4] and the instructions given in [5].

As the on-line medium is inherently changeable, it becomes important to note the time when the material was retrieved, also mentioned in the above sources. The material can then at a later time be accessed through archive functionality on the source in question, or, if that does not exist, use the version archived by the Internet Archive project[‡].

Using Wikipedia as reference

There have been much controversy about using the Wikipedia encyclopaedia[§] as reference. But given prior experience with the encyclopaedia on subjects I am more knowledgeable, the arguments given in the previous section and the fact that Wikipedia keeps a history of all changes to a given page, I believe that it can sometimes stay as the only reference, given that the content is checked and the date is noted.

Of course when dealing with more critical issues other sources should also be referenced to establish more legitimacy, and this has also been a priority.

*Titled: "Documentation – Bibliographic references – Content, form and structure"

[†]http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=25921, which to my mind is ridiculous if you want to promote a standard.

[‡]<http://www.archive.org>

[§]<http://www.wikipedia.org>

Proprietary vs. open source solutions

At the risk of sparking a holy war – I believe that this question is very important for all academic work, this opinion is especially based on the arguments put forward in [6] which concerns mathematical software, though many of the conclusions are also valid in other fields of science.

Other concerns also arise from using proprietary solutions, three important issues will be discussed here. Firstly, integrating solutions becomes quite difficult when you do not understand the inner workings of all the implicated parts. Second, distributing your solution is hard without breaking several laws, even internal distribution within your own organisation might be unlawful. Thirdly, it becomes nearly impossible to share your solution with others.

These issues became blatantly obvious during this project when a great amount of data became unavailable because it was saved in a proprietary format and the software needed to decode it was “missing”. In my opinion, this is a ridiculous situation, when operating in the academic world, surely all formats used should be open and readily available to all?

Based on the arguments mentioned above and those put forth by the open source community as a whole, I have chosen to base this project as much as possible on software which is licensed with OSI [7] (Open Source Initiative) approved licenses, i.e. those based on the OSD [8] (Open Source Definition). In so doing many of the solutions I have found might be relevant to others, and so I have endeavoured to document these where the solution is non-trivial.

Writing style

I have opted for a somewhat informal writing style, as this makes it more enjoyable to write and thus, I hope, more enjoyable to read. The enjoyment factor increases the attention span thus benefiting both writer and reader.

Acknowledgements

I would like to thank my project supervisor Prof. John Hallam, who has been very helpful and inspiring in most aspects of this project. As electronics is not my strong suit I greatly appreciate the help I received from Kirsten Petersen, Richard Beck and Charlotte Ryberg in this regard. Help with the biological aspects of this project have also been important and I would like to thank Prof. Annemarie Surlykke for her help.

PREFACE

Also, special thanks to Randall Munroe for an excellent comic strip[¶] and for making them free for all to use. Some of these comics have been used as headings for the chapters in this report, again hopefully making the report more enjoyable to read.

Acknowledgement goes to Eduard Oscar Schmidt being the originator of the front-page illustration of *Myotis Daubentonii* [9], a Eurasian bat, this specie is present in the Biology department at University of Southern Denmark, and some recordings done with these bats (not by this system) will be used in this project.

Thanks to all the proofreaders and their valuable advice, and special thanks to my family for their patience.

[¶]<http://www.xkcd.org>

Nomenclature

File names look like this.

`Program names` look like this.

Variable names look like this.

[number] or [number, notes] means that the reference “number” has further information on the current context, “notes” make the reference more precise. The reference is listed in the bibliography under “number”.

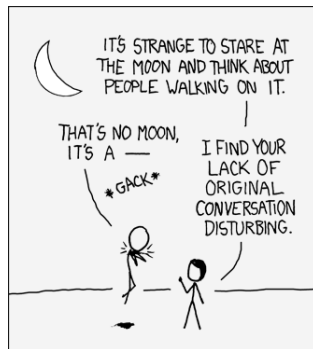
page(number) means that “page” is further documented in section “number” in the manual page called “page”. If on-line manual pages are not available to the reader, they can be accessed at [10].

Part I

Analysis and design

Chapter 1

Introduction



1.1 Inspiration

The original inspiration for this project comes from discussions between Prof. of Biology Annemarie Surlykke and Prof. of Artificial Intelligence John Ham. When studying bats with microphone arrays it is usually necessary to custom build the system. Building a scalable system with exchangeable components would alleviate such issues.

1.2 Relevant non-trivial concepts

To achieve the ultimate goal of recording, tracking and synthesising the calls emitted by an echolocating bat, some concepts need to be made clearer, so as to limit confusion.

1.2.1 Echolocation in bats

Echolocation has been developed several times in nature, most notably by shrews, dolphins, most whales and most bats. The term was coined in 1944 by Donald R. Griffin (a good introductory on the subject can be found in [11]). The reason for its development in bats is most likely that they lost the competition for daylight prey and switched from diurnal to nocturnal life which in turn gives great benefit to animals who master echolocation [12].

Echolocation is what bats, among others, use to navigate and hunt. It basically works like active sonar [13]. Bats, depending on the species, emit sounds ranging from 14 kHz up to around 150 kHz, through their nose or open mouth, again depending on the species. Most echolocating calls are ultrasonic (i.e. above 20 kHz) and are probably defined by the bats habitat and their hunting ground [14]. Generally, lower frequencies have longer range but worse resolution, while higher frequency calls are more attenuated but have higher resolution.

The emitted sounds are usually chirps i.e. frequency sweeps, e.g. see [15, page 1341] and [16, page 312-313], which probably are adapted to the bats habitat and what it is doing (searching, approaching pray or terminal call [15]). Figure 1.1 shows an actual chirp recorded in the Biology Department at University of Southern Denmark, the horizontal axis represents time and the vertical axis frequency, thus this chirp is emitted a around 100 ms into the recording and sweeps down from ca. 60 kHz to 30 kHz. See chapter 12.3 on page 74 for more on how this figure was created.

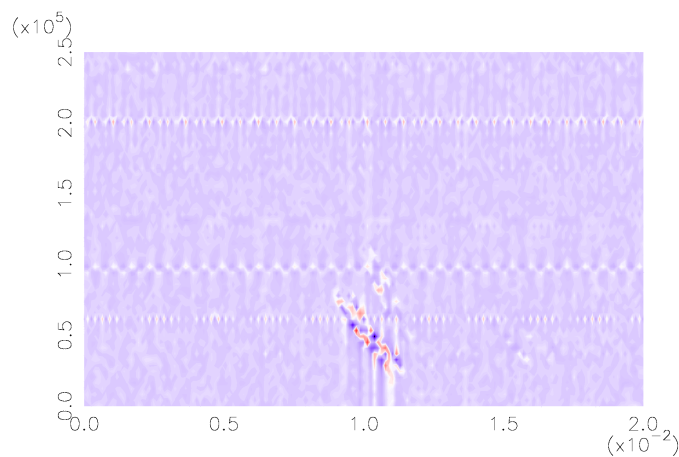


Figure 1.1: Spectrogram of a *Myotis Daubentonii* chirp

1.2.2 Time difference of arrival

Time difference of arrival (TDOA) is a class of methods which determine the location of a source with e.g. a microphone array if the source is acoustic.

The methods based TDOA, are able to determine the location of the source by looking at time-of-arrival and then solving the resulting equations. This, of course implies that the positions of the receivers are known within an arbitrary coordinate system.

1.2.3 Analogue to digital conversion

Analogue to digital conversion (ADC) is the process of digitizing (discrete) an analogue signal (continuous), e.g. sound. This process consist of sampling the analogue signal, each sample is quantized to a finite number of bits.

The result is then a sequence of equidistant samples, giving us a discrete signal which, if sampled correctly, is very similar to the continuous signal.

See also [17], [18, p. 63-73] and for different ADC strategies [19, p. 113-124].

1.3 Specifying project parameters

This section presents the original specifications for this project and their preliminary interpretations. These interpretations will be the basis to work from, and the goal to achieve.

1.3.1 Original specifications

As this project is intended to become a functioning system, initial specifications were made by the people most likely to use the system e.g. Prof. Annemarie Surlykke. Those specifications are listed in table 1.1 on the following page, which is composed from the original list entitled: “Wish-list for array for digitizing and storing high frequency (echolocation) sounds”. The list has been reorganized so that related items are grouped together.

Following discussions about this wish-list showed that list items 1-7 and 17-18 had highest priority. While the other items are important, they do have lower priority.

1.3.2 Interpreting the original specifications

Given the frequency range, we need microphones which have “good” characteristics up to 150 kHz. These microphones need to be connected to ana-

1. 12 simultaneous channels (16 would be better).
2. Sampling rate should exceed 300 kHz per channel.
3. Channel resolution should be 16 bit (12 would do).
4. Anti-aliasing filters on each channel.
5. Bandpass-filter for each/all channel (enable triggering recording of selected species).
6. External triggering (to synchronize with (high-speed) video).
7. Pre-triggering capability.
8. Choice of recording: either a separate file per channel or files with multiple channels.
9. Possibility of storing files in different formats (bin, wav, ...).
10. Automatic update of file names with each triggering of recording.
11. Header files with sample rate, sensitivity, number of channels.
12. Unlimited (or almost) duration of files (up to computer storage limit).
13. On-line display of time signal of 1-3 channels of choice, and/or even better: on-line display of spectrograms of one or more channels
14. (On-line) compressed view of long time with “peak-detection”.
15. On-line view of flight paths.
16. Play-back of HF-sounds (One-channel).
17. The system should be able to operate on battery.
18. The system should be portable.

Table 1.1: Original specifications for the system

logue to digital converters which sample above 300 kHz with resolution above 12 bit, and we need to do this in parallel with at least 12 channels.

Anti-aliasing and bandpass-filtering can be achieved in several ways, the most common being ready made solutions and custom designed op-amp circuits. These should then be connected to each channel.

All items below item 8 can be addressed by choosing a fairly powerful computer with a proper operating system and software tool-chain installed as core of the system, although items 17-18 need consideration in this regard.

So to summarize table 1.2 shows the preliminary list of things that should be included in the project to meet the specifications given in table 1.1 on the facing page.

Items 1-3	A/D-board with 12-16 channels sampling above 300 kHz at resolutions above 12 bit.
Items 4-5	Ready made solution or custom designed solution.
Items 1-5	Microphones which have “good” characteristics in the 10-150 kHz range.
Items 8-16 and 17-18	Fairly powerful computer with a modern operating system installed, constrained by power consumption and physical size.

Table 1.2: Interpreting original specifications

This segmentation of the problem implies that the problem at hand is reducible to these three sub problems, i.e. a microphone with an attached filter connected to an A/D-channel, which again is connected to the computer, see figure 1.2 on the next page.

1.4 Setting realistic goals

Having reduced the issues at hand to three sub problems, the feasibility of such a design will now be analysed, and possible design adaptations introduced which make it more practical.

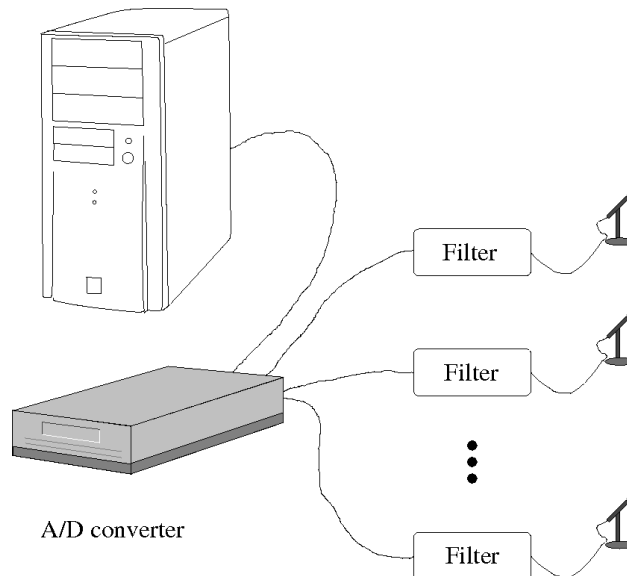


Figure 1.2: Overview of the system

1.4.1 Power consumption and portability

Given that this system is to be battery operated, and be used in the field, it puts some unique constraints on the design. The power consumption of each individual device needs to be considered and minimized. Same argument goes for physical size and weight.

The alternatives here are many, and the key feature to look for, based on our requirements, is the energy to weight ratio. Of course volume and price are also important properties. Rechargeable batteries are the most obvious option at this time, although some other technologies seem to be emerging, and may be available within relatively short time, e.g. fuel cells.

Some data has been gathered about rechargeable batteries, so as to get an overview of what is currently on the market. The numbers listed in table 1.3 on the facing page have been gathered from [20–26] and are typical for consumer grade products. These are all approximate values, as the battery market is a moving target, especially the price is somewhat debatable depending on the manufacturer and specific product.

There seems to be no clear advantage in any of the choices, at least not regarding our primary goals, if we also consider environmental goals, this becomes more clear, as long as the batteries are disposed of correctly. Bottom line is that the choice of battery will most likely depend on which products are available at the time of construction.

	Weight [$\frac{\text{Wh}}{\text{kg}}$]	Volume [$\frac{\text{Wh}}{\text{l}}$]	Cost [$\frac{\text{Wh}}{\text{US\$}}$]
Nickel iron	50	350-360	6-7
Lead acid	30-40	60-75	7-18
NiCd	40-60	50-150	0.5-1
NiMH	30-80	140-300	2.75
NiZn	60	170	2-3
Li-ion	160	270	2.8-5

Table 1.3: Comparison of battery types

1.4.2 Number of microphones and bandwidth

Making some preliminary calculations soon shows us that bandwidth will be an issue. Minimum requirements with sampling at 300 kHz on 12 channels at 12 bit resolution:

$$\text{Minimum} = \frac{300 \text{ kHz} \cdot 12 \text{ channels} \cdot 12 \text{ bit}}{8 \cdot 1024^2} \approx 5.15 \text{ MiB/s} \quad (1.1)$$

And with more optimal requirements:

$$\text{Optimum} = \frac{500 \text{ kHz} \cdot 16 \text{ channels} \cdot 16 \text{ bit}}{8 \cdot 1024^2} \approx 15.25 \text{ MiB/s} \quad (1.2)$$

This gives an indication of the bandwidth requirements this system would have, and if it should be scalable, this becomes a serious aspect to consider.

1.5 Conclusions about design choices

Given that this system should potentially be able to support more microphones a modular design is the most logical. Thus building modules supporting a few microphones each, which only send relevant data to a main system that analyses and presents it. This would allow for arbitrary sized arrays, as long as the bandwidth is not exceeded.

Thus a design like figure 1.3 on the next page, would be more practical, i.e. more scalable and potentially cover more area. This of course presents other challenges, protocol issues, synchronization between the node computers and knowing which channels contain relevant signals to relay to the master computer. These issues will be addressed in the relevant chapters.

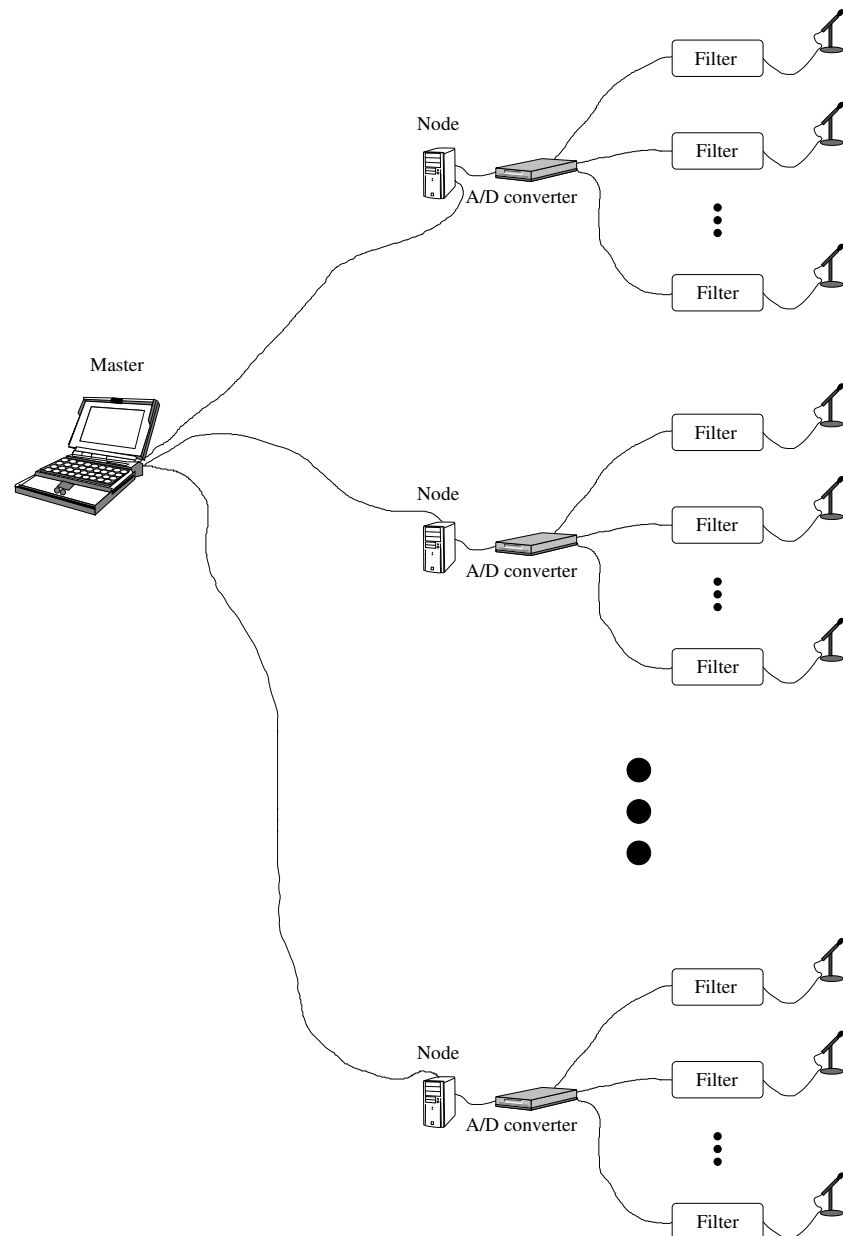
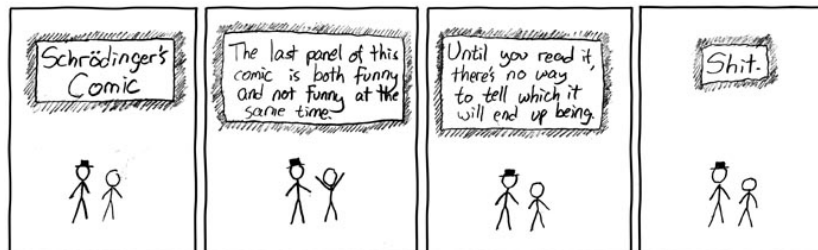


Figure 1.3: Updated of the design of the system

Chapter 2

Microphone and filter



Lets establish minimum requirements for these items to make them successful. To do that, we define the success criteria for each component:

- The microphone should have a “fairly” flat frequency response in the whole area of interest, i.e. 15-170kHz.
- The filter should have variable amplification and variable bandpass limits. The amplification limits will depend on the output from the specific microphone chosen. Bandpass limits depend on the specific signal that is being filtered.

Given that the filter specifications may vary a great deal these will have to be fairly broad.

Thus the microphone must be found first, so as to make the filter design easier.

2.1 Microphone

So what microphone do we choose? The available products on the market are many, which a search at e.g. Farnells website soon would verify.

CHAPTER 2. MICROPHONE AND FILTER

One important fact to consider when searching for components is, what is already available. In this case two fairly similar SMD* microphones with supposed “good” characteristics. These microphones are both from Knowles Acoustics and are described in more detail in the subsections below, their part numbers are SPM-0204-H-E-5 and SPM-0204-U-D-5 respectively. The latter will seem more likely candidate given our purpose, but was not available at first, thus much prototyping was done with SPM-0204-H-E-5, which was then replaced by SPM-0204-U-D-5.

The part number is to be interpreted as shown in figure 2.1. This has been copied from the Knowles design guide [28].

Since Knowles does not seem to supply datasheets for their products online, I have chosen to reference the datasheets provided by farnell.com instead. Thus the datasheets for SPM-0204-H-E-5 and SPM-0204-U-D-5 are available at [29] and [30] respectively.

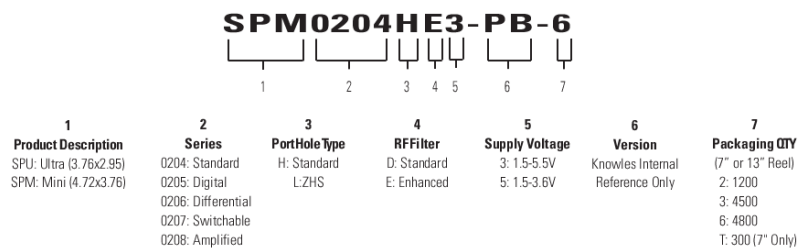


Figure 2.1: Knowles microphone part numbering

2.1.1 Knowles Acoustics SPM-0204-H-E-5

The part number divulges that this microphone is 4.72 by 3.76 by 1.15 mm in width, breadth and height, is from the standard series with standard porthole and enhanced RF-filter. Supply-voltage should be within 1.5 to 3.6 volts.

The datasheet shows nothing about which frequency response this device has above 8 kHz, but it was reported to be acceptable by electronics technicians at the Biology Institute at University of Southern Denmark.

So no further investigation was made at first for other candidates, which the implementation chapters also will show. Halfway through the project, SPM-0204-U-D-5 became available, and were substituted into where SPM-0204-H-E-5 had been used before. This type is described in the next section.

*Surface Mountable Device [27].

2.1.2 Knowles Acoustics SPM-0204-U-D-5

This part has also the dimensions 4.72 by 3.76 by 1.15 mm based on the part number and datasheet. It belongs to the standard series, but has a mesh grid instead of a porthole[†], standard RF-filter and its supply-voltage must be within 1.5-3.6 volts. This microphone is designed for ultrasonic operation up to at least 65 kHz according the datasheet.

2.1.3 Other microphones

There may very well be other microphones that would be a good choice for this application, but as time and resources did not permit, it was not researched further.

2.2 Filter

The filter should anti-alias [31] the signal. Aliasing occurs when a signal contains frequencies higher than half the sample rate, thus a sequence of samples might match several different signals if you do not get at least two samples per period, also known as the Nyquist rate [18, p. 6]. See for example figure 2.2 which comes from [32], this reference and [18, p. 12-27] also describes aliasing in greater detail.

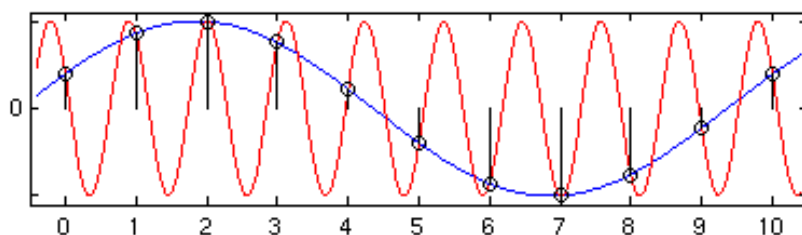


Figure 2.2: Two different sinusoids that fit the same set of samples

According to item 5 in table 1.1 on page 6 the filter is required to allow only a certain frequency band through, this is a bandpass filter. A bandpass filter is usually described by its centre frequency f_c and the width of the passband, delimited by f_1 and f_2 , see figure 2.3 on the next page which is from [33] (also [18, figure 11.6.6 p. 618] which is even more detailed).

[†]This distinction is obviously missing from the part number description given in the design guide [28]

CHAPTER 2. MICROPHONE AND FILTER

Bandpass filters are described more thoroughly in [33] and [18, p. 618-622]. An example transfer function for a bandpass filter might look like this (copied from [34, p. 237]):

$$H(s) = \frac{H_0 2\zeta\omega_0 s}{s^2 + 2\zeta\omega_0 s + \omega_0^2}$$

Where ω_0 is the resonance frequency, ζ is the attenuation factor and H_0 is the amplification. Note bandpass filters must at least be of second order, and to be symmetrical must be of even order.

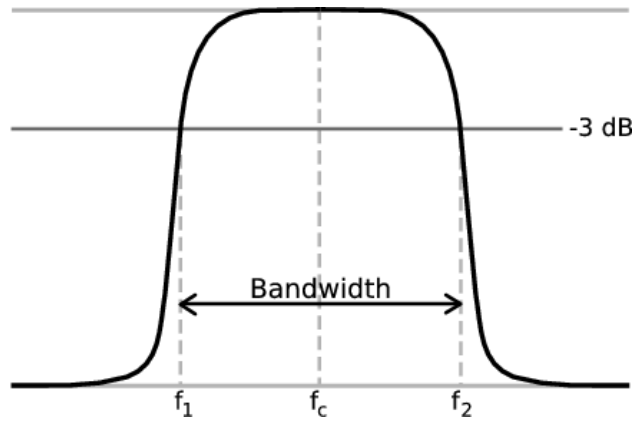


Figure 2.3: Transfer function plot of a bandpass filter

The bandpass filter should be tunable, so that the passband can be set to only record certain species. Different species use different frequency bands and with varying bandwidths [16], so the placement of f_c and the width of the bandwidth should be configurable. This requires flexible building blocks, thus the best choices here are either IC-solutions where the filter is varied with external components, or custom solutions built with operational amplifiers, resistors and capacitors.

While researching this subject, the custom solution seemed like the best candidates, and was the one chosen. After having spent much time on implementing such filters, I am fairly confident that the IC-solution would have been an easier approach, an integrated circuit, such as Maxim 274 or Maxim 275 (datasheet: [35]), I predict, would make the design phase somewhat faster, and would be the solution of choice given another opportunity to design a bandpass filter.

However, the circuit used in this project has been built with operational amplifiers and passive components, and thus it is the design phase of this circuit that will be described here.

2.2.1 Bandpass filters

There are a host of different ways to build bandpass filters with op-amps and passive components. The references used in the design phase are [18,33,34,36] and a trial-ware software program [37].

The most simple filter design is the one based on capacitors and inductors. Given that inductors are difficult components to work with, replacing them with gyrators [36, p. 266-267] is a good idea. Having established the building blocks of the circuit, we need to define the filter type. The filter types examined for this project were Butterworth, Chebyshev and Bessel filters each having its benefits and drawbacks, see table 2.1.

	Benefit	Drawback
Butterworth	Flat passband	Wide transition region, poor phase characteristics
Chebyshev	Sharper cut-off frequency	Ripples in the pass-band, not good phase characteristics
Bessel	Good phase characteristics	Ripples in the pass-band

Table 2.1: Filter type benefits and drawbacks

The above described filters can be built in several different ways, one choice is to build a voltage-controlled voltage-source (VCVS) filter based on the tables shown in [36, p. 274], but these filters are very sensitive to component tolerance [36, p. 270], thus a better choice is the state-variable-filter, or its close relative, the biquad filter ([36, p. 277-279] and [34, p. 249-253]). These have improved stability and are easy to adjust compared to the VCVS filters. They also have quite high Q -factor, i.e. short transition regions.

2.2.2 Designing the filter

As my experience in filter design is limited, a trial and error approach has been used when designing this filter. The starting point was a combination

of the trial-ware software program [37] and a biquad Bessel filter, the Bessel filter was chosen because of its good phase characteristics. Phase linearity is important because it linearises the group delay and ensures higher spectrogram accuracy. This design was tested on a breadboard, and shown to have acceptable characteristics, thus it has remained the basic design. This design is sketched in figure 2.4, and is related to the state-variable-filters. The design is known as a biquad filter and has the same characteristics as the state-variable-filters, i.e. good stability, high tolerance for component variation and high- Q .

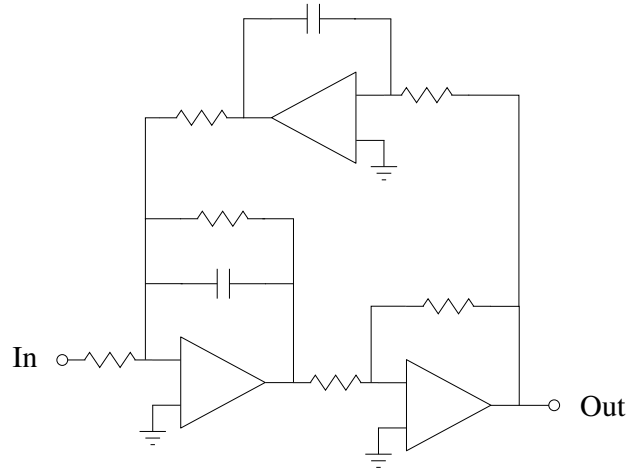


Figure 2.4: Biquad filter design

2.3 Concluding the microphone and filter discussion

Here we try to draw conclusions about what we learned from the analysis above. This is separated into two sections.

2.3.1 Choosing microphone

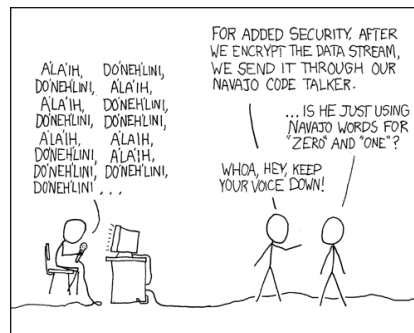
Given the limited research that has been done, the choices presented in section 2.1 on page 11 would indicate that the Knowles Acoustics ultrasonic microphone is the best choice (SPM-0204-U-D-5).

2.3.2 Choosing filter design

As discussed in section 2.2 on page 13, the choice, although many other viable solutions exist, is a 2nd order biquad-filter. Figure 2.4 on the facing page shows a sketch of such a filter.

Chapter 3

Digitising microphone information



How do we enable the computer to interpret the data generated by the microphone and filter (see Chapter 2 on page 11)? The information the microphone generates is analogue and thus needs to be digitised so that the computer can “understand” it. This is usually achieved by analogue-to-digital conversion (often abbreviated ADC or A/D-conversion, see section 1.2.3 on page 5 for more details on the concept).

3.1 Finding product series

3.1.1 Criteria for the ADC

There are many different solutions for performing A/D-conversion on regular computers, e.g. the products from National Semiconductor [38], Amplicon [39] and others, thus the choice is not trivial.

The main things to look out for here are those listed in table 1.2 on page 7, equation 1.2 on page 9 and the bandwidth conclusions found in section 1.5

on page 9. Taking this into consideration we need an A/D-board that:

- contains more than one channel, but not more than its bandwidth capability.
- has as low latency as possible, i.e. travel time from the microphone to the master.
- sample rates above of 300kHz, resolution above 12 bit.

3.1.2 Bandwidth and latency

To optimise bandwidth and reduce latency, the bus technology chosen for this system is very important. Seeing as there are several different interfaces to choose from [38], we will try to narrow the field by matching the interface up against our criteria. With the bandwidth requirements in our application the most applicable bus-technologies, which also are commonly available, would be: USB, SATA, PCI, PCI Express, firewire. Considering the latency factor and some latency issues found with the OHCI [40, p. 45] controller, which both USB and firewire use, this list shortens to SATA, PCI and PCI Express, considering also that I have not found any cards supporting the SATA interface this leaves PCI and PCI Express, which both have similar latency [41].

3.1.3 Driver issues

As discussed in the Preface, use of open source solutions is very important, meaning that the product used should work seamlessly with open source operating system kernels, GNU/Linux in particular and preferably others as well, e.g. one of the BSD kernels, OpenSolaris etc. This would pose no problem if the interface to the hardware is well described.

Driver issues narrow the field quite a bit more, the research only found one product series that offer anything resembling open source support, those sold by Amplicon. Further research in this regard may be warranted.

The best option, given the time, would be to construct our own A/D-board, this would enable us to design it specifically to our purpose, and would ease integration into any computer system we might choose. The time required to acquire enough expertise to construct the board would be extensive, and as such would probably blow any time-plan that is in place. Even though this approach is attractive it would most likely require its own full time project.

3.1.4 Determine supplier

The driver issues definitely clinch this discussion, given the importance of supported drivers for a custom built system that we are trying to build. Thus Amplicon seems to be the only choice here.

After contacting Amplicon to get more information about their driver, we were told that we could get the source code for both the driver and the API library if we would be willing to sign a NDA (non-disclosure agreement). This is not optimal, but seems to be the best we can do for the moment. If a more open solution presents itself at a later time, this should be chosen instead as it simplifies many things (integration, distribution, sharing, etc.).

3.2 Finding product series

In the previous section we determined the supplier, now we move forward to find a product that we want. We choose between the PCI and PCI Express products, the choices are listed in the following two sections.

3.2.1 PCI based products

The product list in table 3.1 on the facing page is a reproduction of the list on the Amplicon website [42].

Given the criteria we are working from (section 3.1.1 on page 18), the products that best fit our requirements are the 2000 series, e.g. 2010 supplies up to 2 MS/s and 14 bit resolution. The 9820 [43] (14 bit), 9812 [44] (12 bit) and 2200 [45] (16 bit) are also acceptable candidates given number of channels, sample rate and resolution, but are somewhat overqualified, and as such would cost more than needed.

3.2.2 PCI Express based products

As with the PCI based cards, a list has been fetched from the Amplicon website, the PCI Express products available are listed in table 3.2 on the facing page (adapted from [46]).

The range of products here are not as plentiful as with PCI based cards, even so the 2000 series is also available here, and again seems to be the best choice for our application.

Product name	Product description
instruNET	16 channel (expandable) direct sensor input
DAQ 2000 series	4 channel simultaneous sampling, up to 2MS/s
DAQ 2200	64 channel 3MS/s multifunction
DAQ 2208	96 channel 3MS/s analogue input
PCI230+	16 channel 16-bit multifunction analogue input, 500kS/s
PCI260+	16 channel 16-bit analogue input, 500kS/s
PCI 9812	4 channel 20MS/s simultaneous sampling digitiser
PCI 9820	2 channel 65MS/s digitiser
PCI 9111	Low cost 100kS/s multifunction
PCI 9112	16 channel 12-bit multifunction, 100kS/s
PCI 9113A	32 channel isolated analogue input, 100kS/s
PCI 9114 series	32 channel 16-bit analogue input, up to 250kS/s
PCI-9118/L	16 channel 12 or 16-bit analogue input, up to 330kS/s
PCI-9221	Low cost 16 channel 16-bit multifunction DAQ with 2 channel encoder
PCI-9222/9223	Multifunction 16-bit DAQ and motion control card
ZT410PCI	500MS/s, dual channel precision digital storage oscilloscopes
ZT431PCI	200MS/s, dual channel 12-bit digital storage oscilloscope
ZT450PCI	2.5GS/s, dual channel 8-bit digital storage oscilloscope
ZT4610PCI	4GS/s, dual channel 8-bit digital storage oscilloscope/digitiser

Table 3.1: Amplicon PCI based ADC products

Product name	Product description
DAQe-2000	PCI Express 4 channels simultaneous sampling
DAQe-2200 series	PCI Express 64 channel multifunction, up to 3MS/s
DAQe-2208	PCI Express 96 channel analogue input, 3MS/s
DAQe-2213/14	PCI Express 16 channel 250kS/s multifunction

Table 3.2: Amplicon PCI Express based ADC products

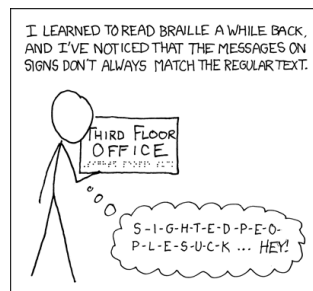
3.3 Choosing product

The discussion in this chapter, which is dependant on the current amount of research done in this regard, leads us to the conclusion that the Amplicon 2000 series is the best choice on the market, and this is valid in both preferred bus types, i.e. PCI and PCI Express.

Within this series [47], which contains four different products, 2005, 2010 and 2016 are all acceptable. Within this project we have chosen to use the 2010 product.

Chapter 4

Computers involved



If we consider the design choice made and illustrated in figure 1.3 on page 10, we find that we are going to need some computers, one master and potentially several node computers which are controlled by the master computer.

4.1 Master computer

This part of the system should have a fair bit of processing power so that the data it receives can be processed in near real-time. The data processing requirements could probably be handled by an average desktop/laptop computer sold today.

The finished solution might include a SBC (single board computer) with integrated screen, so that no other equipment is required. However, the preference of many might be to just handle the data stream on their own laptop.

Otherwise we do not put any special requirements on the master computer, the analysis software should be as platform agnostic as possible.

4.2 Node computer

This is an important component of the system. This computer should:

- accommodate whatever A/D solution is chosen.
- be fast enough to record data from all channels and send it onward.
- be easily battery operated.
- be portable, i.e. not too heavy or cumbersome.
- be supported by an open source operating system.

Table 4.1: Node computer criteria

Besides the items listed above, there is one other consideration regarding this project, due to the A/D-board market we are limited to those architectures supported by the software provided for the A/D-board by Amplicon (see discussion in section 3.1.3 on page 19) and their product series is only supported on the IA-32 architecture, we could port the driver, since we got the source code, but this would be rather time consuming and, to my mind, futile as we would not be able to share it because of the NDA.

Given that the system should be portable, based on products which are well established in the marketplace and support either PCI or PCI Express, our choice is limited to the SBCs that support those bus technologies. The limit on the form factor will vary according to whom you ask, but we have chosen the Mini-ITX as the largest viable alternative. This leaves us with the following form factors to consider, they are defined by the Taiwanese company Via*, Mini-ITX version 1.0 [48], Mini-ITX version 2.0 (due to be available in the fourth quarter of 2008 according to [49]), Nano-ITX [50][†] and Pico-ITX [51], which all come in IA-32 (32 bit intel architecture [52]) versions.

4.2.1 Nano-ITX and Pico-ITX

As promising as these form factor might seem, considering power consumption and physical size, they do have one major problem with regards to this project, they do not have a PCI port. They only have a Mini-PCI port.

*<http://www.via.com.tw/en/>

[†]This is not a white paper, as none could be found, but rather a specific Nano-ITX product. Some notes about Nano-ITX are in the Pico-ITX white paper [51].

If other A/D-boards were available that supported the Mini-PCI interface however, these would become very well suited in this project.

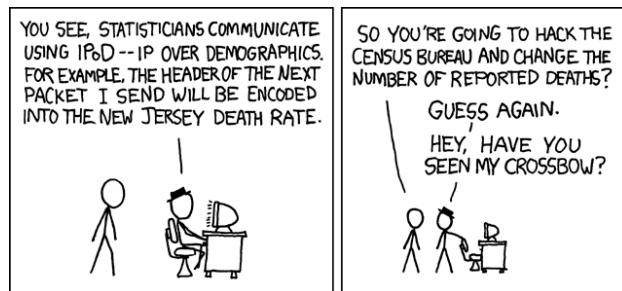
4.2.2 Mini-ITX

Most Mini-ITX products contain PCI ports, and some also contain PCI Express ports, thus this is the most viable choice at the current time. With regards to processing power, they come in many different sizes, and we should be able to accommodate our needs. As we have a regular PCI bus with bandwidth as high as 533 MiB/s, we should have no issues there either.

Using these SBCs as node computers should be quite easy as they are well supported by the open operating systems, at least well enough for our purpose. Forwarding recorded data should not be any problem either, as the boards have built in ethernet controllers which are connected to the DMA controller, thus enabling us to move data quickly.

Chapter 5

Design conclusions



This chapter summarises the important findings done in the analysis phase of this project. Thus the most important conclusions are repeated here.

5.1 System design

The design for the system is hierarchical with a master computer at the top and with arbitrarily many nodes below. Top-down illustrations are shown in figures 5.1 to 5.2 on the next page.

5.2 Microphone and filter

The proposed microphone for this system is the Knowles SPM-0204-U-D-5 SMD microphone, this may change pending various test results. The chosen filter for the system is a 2nd order Bessel biquad filter.

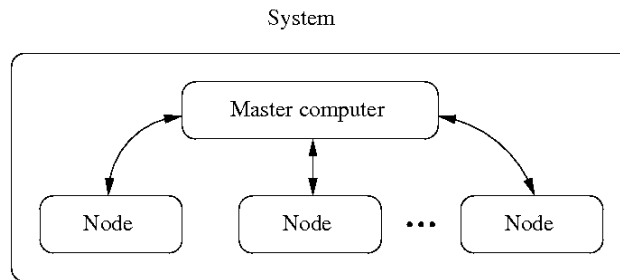


Figure 5.1: System design decided on

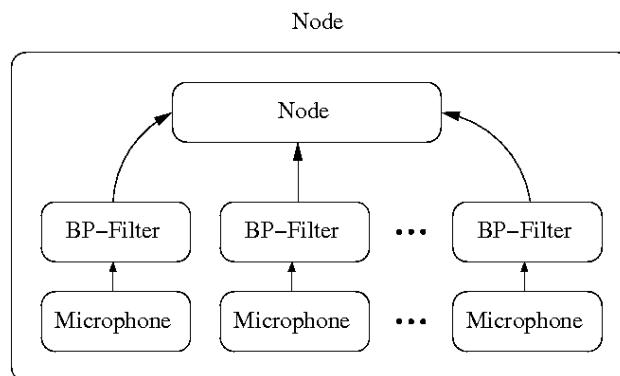


Figure 5.2: Node design decided on

5.3 A/D-Converter solution

The A/D converter solution proposed is a 2010 product for either PCI or PCI Express. This product includes Linux kernel drivers.

5.4 Master and node computer

The master computer should be fairly powerful to be able to run the analysis of the recordings. Other than that it should be platform agnostic, so it can be implemented on any operating system and architecture.

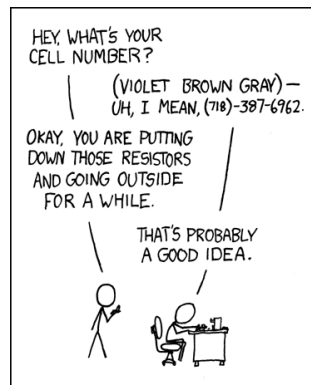
The node computer proposed is of the Mini-ITX form factor, which would make the system small yet moderately powerful. This system also provides us with a PCI or PCI Express slot required for the A/D-board decided on.

Part II

Hardware

Chapter 6

Building the filter and microphone



Based on the discussion in 2 on page 11 we need to build a filter. Given that there are a multitude of available technologies to accomplish this, a choice needs to be made. As it has been hinted earlier, the design of this filter circuit has been a process of trial and error or design by construction.

One interesting aspect of designing circuits, is in choosing what software solutions to use, especially when a side-goal is to use open source software as much as possible.

6.1 Circuit technology

Given the technologies that have been at my disposal: breadboard, through-hole and PCB manufacture with bubble-etching, and the fact that there are size constraints to consider, the PCB solution seems to be the way to go.

Of course designing the circuit is much easier done on a breadboard, and thus this is where the design by construction started.

6.2 Choosing the correct tools

When a design reaches a certain size it becomes increasingly important to transfer the design to a computer, especially if there are plans to fabricate any of the involved components. As we plan to create a PCB of the filter, this is highly relevant.

As this might be considered non-relevant to the project core, this discussion has been moved into Appendix A.1 on page 97. The discussion has been retained as it has important bearing on how to continue work with this design, thus if others would like to copy the approach used here they can do so easily.

6.3 Design by construction

This section first determines the final design of the filter, and then any adaptation circuitry needed to make the filter work with the computer.

6.3.1 Filter design

First of all we need to determine which component values to use, and acquire operational amplifiers with acceptable frequency characteristics. After some searching on the RS-online website*, the DIL version of OP275 opamp [53] was found and acquired. This operational amplifier has a bandwidth of 9 MHz, which should be sufficient for our purpose as we plan on approximately a gain of ten.

Since the circuit we are building is a biquad filter (see figure 2.4 on page 16), we know how many and what type of components we need, the values can be determined from the tables listed in Horowitz and Hill [36] or as we have done it, using initial values found from the trial-ware software program [37], and then tweak them as the experiment warrants.

During the tests it was found that cascading two of these filters improved the transition region giving us four poles, two on each transition region. This means that we now have a 4th order bandpass filter. One of the intermediate versions of the breadboard filter can be seen on figure 6.1 on the next page. Most of the seen capacitors are decoupling capacitors. The orange capacitors are part of the filter and are placed in parallel because suitable values were not available.

*<http://dk.rs-online.com/>

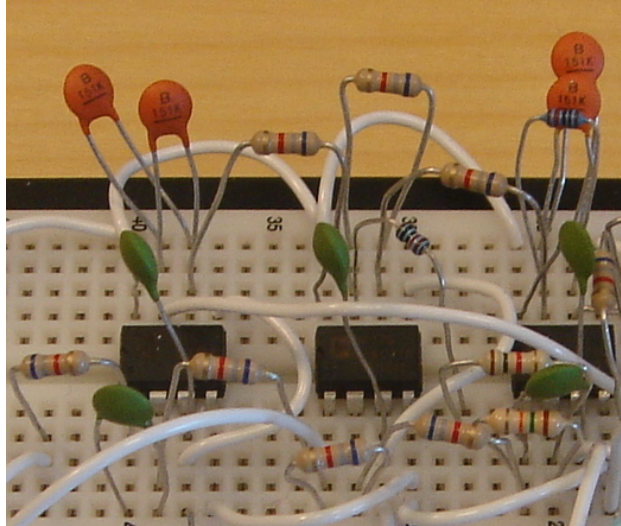


Figure 6.1: Filter implemented on a breadboard

Schematic of the design

As established in Appendix A.1 on page 97 the software of choice for this task is gEDA [54]. From this package the `gschem` program [55] is used for creating schematics. While developing this schematic, it was found to be necessary to create a new symbol which represents the four terminal Knowles SMD microphone. The symbol file for `gschem` can be seen in Appendix B.1 on page 102.

The schematic for the final circuit can be seen on figure 6.2 on the facing page. The component sizes were the ones found while experimenting on the breadboard. Although some of the capacitors on the schematic are shown as polarised, it turned out that ceramic capacitors of sufficient size were available, they were used instead.

Looking at the schematic, we see that more components have been added. The centre part of the circuit contains the cascaded biquad filter. Just before the filter, a capacitor (C22) has been added which removes the DC-offset of the signal from the microphone. The size was chosen to allow all interesting frequencies to pass.

The diode was added to allow for a 0.7 V voltage drop, so the microphones would not be supplied with too high voltage, C14 decouples the diode. This was later found to be the wrong way to go, this will be discussed further in section 14.1.3 on page 85. C5 improved the response from the microphone under some conditions, but needs to be verified by further testing. C7 again removes DC-offset from the signal before sending it onwards to the A/D

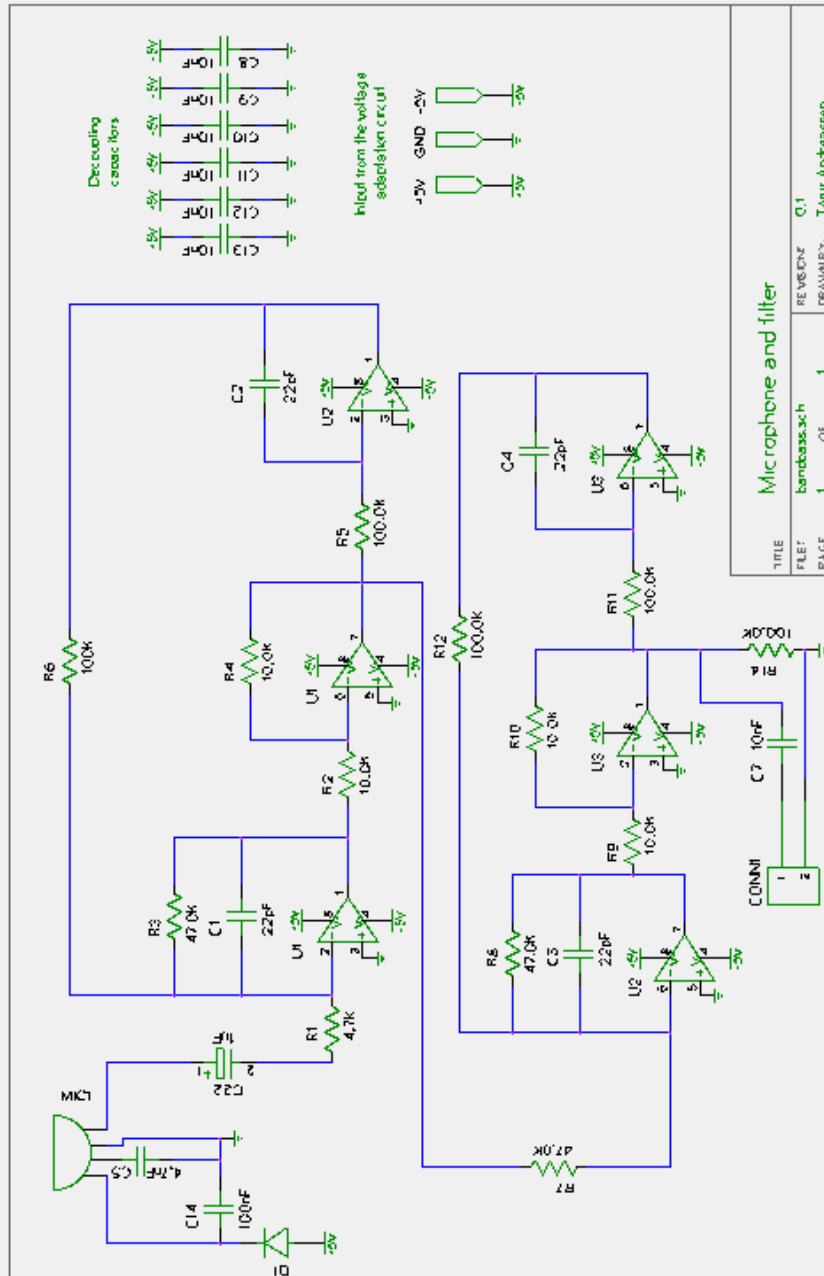


Figure 6.2: Bandpass filter schematic

CHAPTER 6. BUILDING THE FILTER AND MICROPHONE

converter. C8 through C13 decouple the power supply for the op-amps, both negative and positive supply.

Components used

What remains now is to transfer the schematic to a PCB design tool, to do this we need to know what components are to be used with the circuit. The cascaded biquad filter consists of several passive components and six operational amplifiers. Based on the design, components have been found, both for through-hole/breadboard and SMD, table 6.1 shows the BOM (bill of materials) in both cases.

Note that the operational amplifier has been changed to TSV912ID, because the OP275 was not available as SMD component at the time. The datasheet for this component can be found here [56], this component has 8 MHz bandwidth, which is still sufficient for our purpose.

Component	Value	Through-hole	SMD
R1	4.7 K	Axial	0805
R2, R4, R9, R10	10.0 K	Axial	0805
R3, R7, R8	47.0 K	Axial	0805
R5, R6, R11, R12, R14	100.0 K	Axial	0805
C1, C2, C3, C4	22.0 pF	Radial	1206
C5	4.7 nF	Radial	1206
C7, C8, C9, C10, C11, C12, C13	10.0 nF	Radial	0805
C14	100.0 nF	Radial	1206
C22	1.0 μ F	Radial	1206
MIC1	SPM0204UD5	NA	NA
CONN1	NA	100 mil	100 mil
U1, U2, U3	OP275GSZ	DIL	NA
U1, U2, U3	TSV912ID	NA	SO-8

Table 6.1: Bill of materials for the bandpass filter

6.3.2 Adaptation circuit design

After the filter design worked, the tests were made more realistic. The power supply for the node-computer was used to power the filter circuit. This is 15 V

supply and needs to be adapted so we can supply the operational amplifiers with symmetric power, e.g. ± 5 V. Meaning we need a power regulator and a balanced ground. An example circuit was designed during discussions with Prof. Hallam.

Figure 6.3 shows the final design settled on. The following section discusses this further and develops a schematic for.

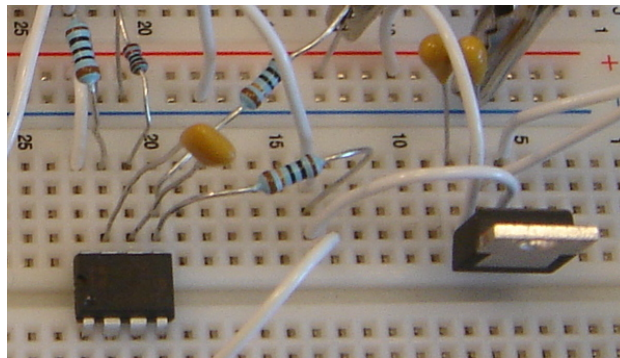


Figure 6.3: Power adaptation implemented on a breadboard

Schematic of the design

See figure 6.4 on the following page for the schematic for this circuit. This circuit works by first using a voltage regulator which caps the input voltage to the circuit. The output from the voltage regulator is then split (with R15 and R16), impedance corrected (op-amp) and stabilised, C17, R17 and R18. Finally the new balanced ground is decoupled with C24 and C25. C20 and C21 decouple the negative and positive supply to U4. The output from this circuit will depend on where the voltage regulator caps the input, e.g. with a 9 V regulator the output will be ± 4.5 V, and with 12 V it will be ± 6 V.

Components used

As with the filter, we need to determine the BOM for the circuit. This is easily gathered from the schematic design and can be seen on table 6.2 on page 37. The voltage regulator used with the various circuits built has varied, specifically both 9 V and 12 V regulators have been used with acceptable results.

Figure 6.4: Power adaptation schematic

Component	Value	Through-hole	SMD
R15, R16	10.0 K	Axial	0805
R17	100.0	Axial	0805
R18	100.0 K	Axial	0805
C17	100 nF	Radial	1206
C18	330 pF	Radial	1206
C23, C24, C25	1 μ F	Radial	1206
CONN2	NA	100 mil	100 mil
U4	OP275GSZ	DIL	NA
U4	TSV912ID	NA	SO8
U5	MC7809CT	DIL	NA
U5	MC7809CDTG	NA	SO-8

Table 6.2: Bill of materials for the power adaptation circuit

6.4 Transferring the design to PCB

Here we go through the steps of converting the schematics created above into a PCB. There were two separate ways used during this project, an in-house production method and creating them professionally elsewhere. Needless to say the in-house approach is much cheaper and better suited for prototyping. Both of these ways require a PCB design to be created first, although the final steps differ.

6.4.1 Schematics to PCB design

We now proceed with the PCB design creation, the **gEDA** software collection includes a PCB design program [57,58] for just this task, and it is fairly well integrated with the rest of the programs. Thus with the exported netlist from **gschem** together with the **gsch2pcb** script [59], it is easily imported into the PCB design tool, which incidentally is called **pcb**. As with most PCB design tools, after creating a schematic the PCB design tools includes rats-nests which makes manual placement of components, and laying down wires very easy, though it may be tedious.

Like with the **gschem** program, some of the needed components were not available, and thus needed to be created. This process is described more thoroughly in Appendix B.2 on page 106.

6.4.2 Producing in-house PCBs

To create in-house PCBs, we need to use a bubble-etch bath. The process of transferring the schematic to a PCB is fairly straightforward. The steps needed for our setup are listed in table 6.3. These steps are adapted to our software choice, and the production material available to us.

1. Create initial *.pcb*, *.net* and *.cmd* files from the schematics file by running the *gsch2pcb* script.
2. Open the *.pcb* file, load netlist file (the *.net* file), execute the *.cmd* file, all components should now be correctly named and the rats-nest should be updated.
3. Set up initial variables, such as wire clearings, component clearing etc. Place components and connect all rats-nests. To ensure all design rules have been kept, run the design rule checker (DRC).
4. Print both top and bottom of the PCB onto transparencies, the component-side should of course be mirrored.
5. Attach the transparencies to the photosensitive copper-layers, and subject them to UV-light. Ensure that the sides are aligned, so that connected vias are directly opposite each other.
6. Put the PCB into a solution that dissolves the photosensitive layer that was subjected to light.
7. Put the PCB into the bubble-etch bath, it should remain there until only the design remains.
8. Rinse the PCB and optionally treat it with corrosion protection.
9. Solder components onto the board, connect the vias.

Table 6.3: PCB production steps at the Maersk institute

Item 4 in table 6.3 refers to component mask and copper mask drawn in *pcb*, the masks for the fourth prototype can be seen on figure 6.5 on the next page (not to scale). When applying the UV-light we line up the transparencies to the surrounding lines on the masks.

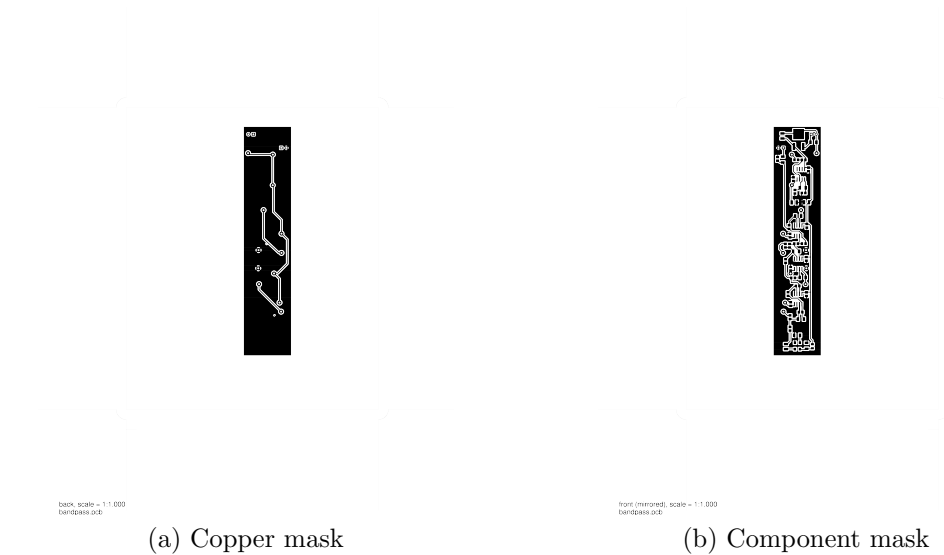


Figure 6.5: Copper and component masks of the PCB produced

6.4.3 Manufacture PCBs

This process follows some of the same steps used in in-house production, specifically steps 1-3 in table 6.3 on the facing page, where we create the PCB design on a computer.

The divergence is mostly about creating Extended Gerber files [60] (also known as RS-274-X) from the PCB design. As `pcb` already supports the Extended Gerber format, this is just a question of exporting the design in the correct format, and mailing it to the manufacturer. This process results in a much more professional looking PCB, with silk layer, protective coating and coated vias.

The design should be fairly stable before resorting to such production method as it is rather costly.

There were two comments when submitting the Gerber files, firstly the manufacturer required there to be a solder mask around the PCB design, secondly the custom made footprint for the voltage regulator (Appendix B.2 on page 106) had some issues when being exported. These issues should be examined further.

6.4.4 Examples of produced PCBs

As mentioned earlier, the PCB production process is rather new to me, and thus has been a trial and error process. This has resulted in several interme-

diated prototypes. Figure 6.6 on the facing page shows several of these, it also shows the manufactured prototype. The prototypes all had different issues, the prototypes and some of their issues are briefly described below.

First prototype (figure 6.6a) This was created with the Kicad program (see section A.1.1 on page 98). The problems here were mostly production related, and when design flaws began to emerge, such as the orientation of the op-amps and the dimensions of the board, the decision to move forward to the second prototype (and another software collection) was taken.

Second prototype (figure 6.6b) The first prototype developed with the gEDA program collection. The major issues here were that of clearance, the board was short-circuited in several places. Thus the clearance values were increased for the next prototype.

Third prototype (figure 6.6c) This prototype worked fairly well, after some ad-hoc solutions were applied. The black wire connects two ground planes which accidentally were not connected, when the clearance values were increased.

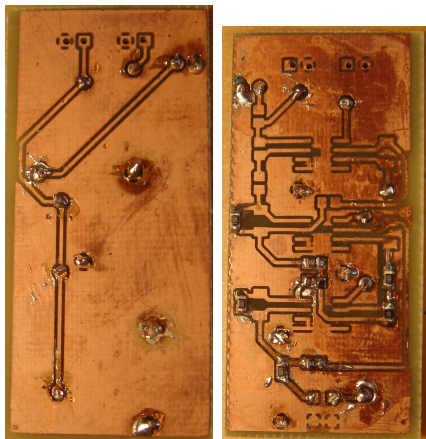
The white wires seen on the picture were added when it was found that the circuit needed negative as well as positive supply voltage. This prototype was the foundation for the fourth and final PCB version, final for this project at least.

Fourth prototype (figure 6.6d) This is the prototype that was produced professionally. Of course it was thought that the prototype was now complete and without serious errors. As it turns out, a last minute change of voltage regulator proved to be fatal for the circuit, it overloads the operational amplifiers. Another serious mistake has also been found too late, the supply voltage for the microphone should be below 3.6 V and not 5.5 V as the schematic shows. This mistake is due to a mix-up of product and datasheet.

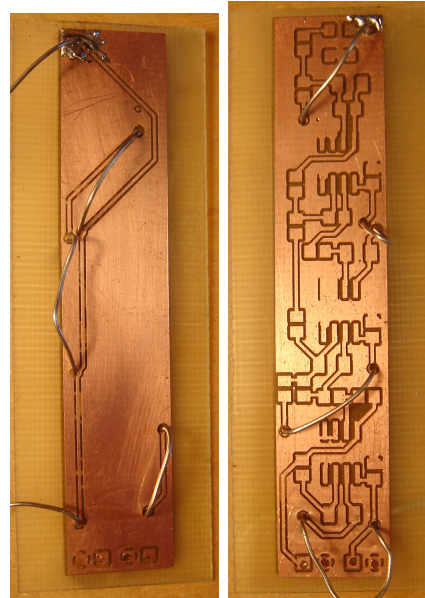
Needless to say this purchase was a failure and due to my inexperience in designing PCBs, or more positively, this is not a mistake I will make again.

6.5 Conclusions about the built circuit

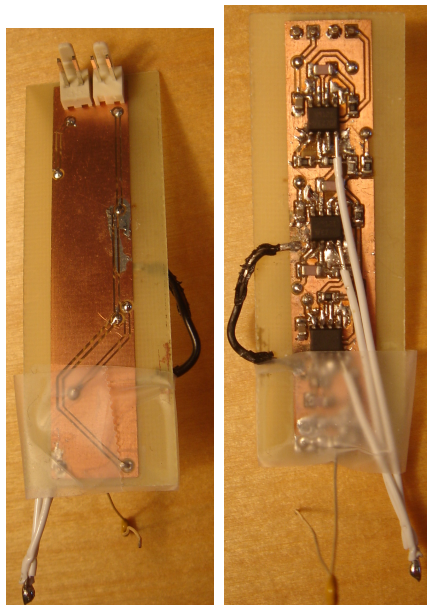
The breadboard design works, and this should be demonstrated in the test chapters. Most of the other things that were attempted have failed in some



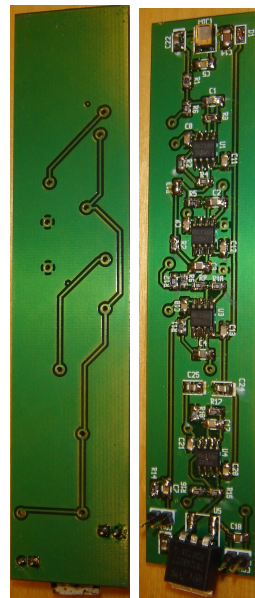
(a) First prototype



(b) Second prototype



(c) Third prototype



(d) Fourth prototype

Figure 6.6: PCB prototype progression

CHAPTER 6. BUILDING THE FILTER AND MICROPHONE

degree. The schematic contains serious errors with regards to microphone supply voltage. The PCBs produced all had some errors, most seriously the professionally produced PCBs fail, probably due to a voltage regulator that was switched in at the last moment without being tested.

The professionally produced boards should be tested without the microphone and with the originally planned voltage regulator.

Chapter 7

Building the node computer



Following the discussion in section 4.2 on page 24, we now need to purchase the items identified. This chapter goes through which items were purchased, and then the assembly of the node computer, the master computer is assumed to be any workstation with reasonable resources.

7.1 Purchase components

To build the node computer, several items need to be acquired. The items that were purchased are listed in table 7.1 on the next page. All prices but one (it was already in DKK) were converted from British Sterling pounds (GBP) to Danish kroner (DKK), the conversion used the GBP value listed by Danske Bank *. Needless to say these prices are approximate, but the total should be reliable within circa ± 1000 DKK.

The angled Riser board is needed to be able to fit the PCI A/D-board into the enclosure. The HID devices (keyboard, mouse and display) have been left out of this list, as they are not to be used in the final system, and as such will not affect the cost of the final product.

*Left hand side of <http://www.danskebank.dk>

CHAPTER 7. BUILDING THE NODE COMPUTER

Item	Type	Price (DKK)
A/D-board	Adlink Technology 4 channel PCI DAQ 2010	7000
Mini-ITX motherboard	EPIA-MII12000G 1.2GHZ C3 Nehemiah CPU with cooling fan	1100
Enclosure	Wall mount enclosure for mini-itx boards	250
DC to ATX DC converter	PicoPSU - 60W plug-in DC-ATX converter. 6-26V input	250
Hard disk	40 GB EIDE 2.5" 4200 RPM or 5400 RPM hard disk drive	400
RAM	DIMM-1GB-333-DDR-NR	700
Angled Riser board	Rohs compliant pci Riser	50
Total		9750

Table 7.1: Purchased items for the node computer

Figure 7.1 on the facing page shows the different components that the node computer consists of.

7.2 Assembly of the node computer

This is a fairly straightforward process, and is described by the photo series shown in figure 7.2 on page 46, and described briefly in table 7.2 on the facing page.

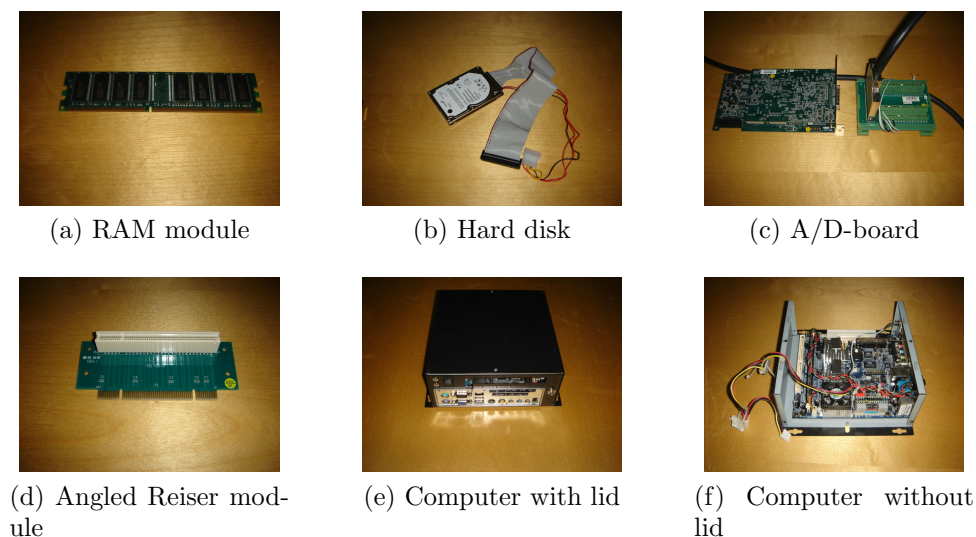
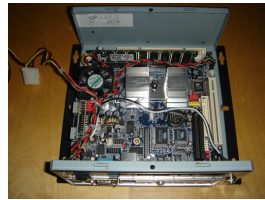


Figure 7.1: Components of the node computer

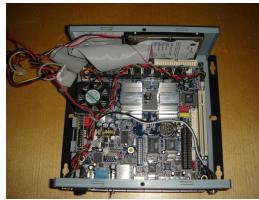
1. The assembler should be grounded to the chassis at all times during assembly.
2. Fasten motherboard to the enclosure.
3. Connect RAM module.
4. Connect and fasten hard disk.
5. Connect PicoPSU.
6. Connect Riser-board to A/D-board.
7. Connect Reiser and A/D-board the PCI bus and fasten it to the enclosure.
8. Fasten the lid of the computer.
9. Connect A/D-boards I/O-board.
10. Connect HID devices.
11. Connect power supply or battery.

Table 7.2: Node computer assembly instructions

CHAPTER 7. BUILDING THE NODE COMPUTER



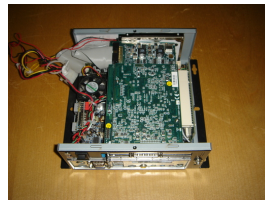
(a) RAM module added



(b) Hard disk added



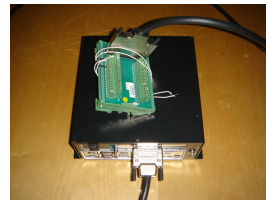
(c) Reiser module and A/D-board combined



(d) A/D-board added



(e) Enclosure fastened

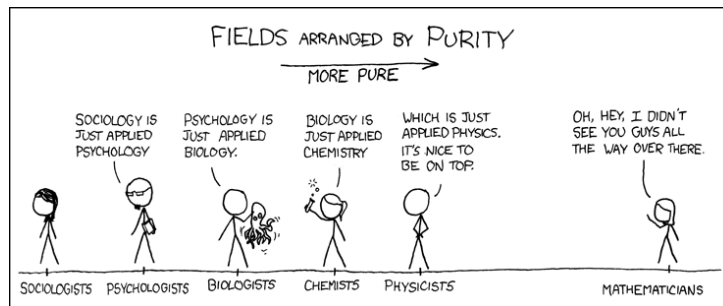


(f) I/O board connected to the A/D-board

Figure 7.2: Assembly of the node computer

Chapter 8

Combining hardware components



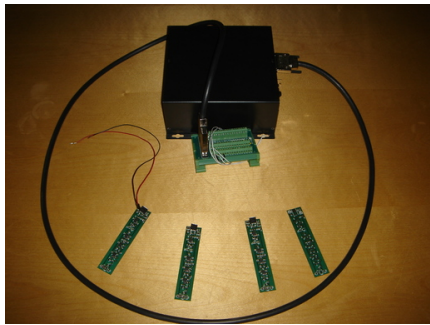
Now that both the filter and node computer have been built, these are connected to each other.

8.1 The connected system

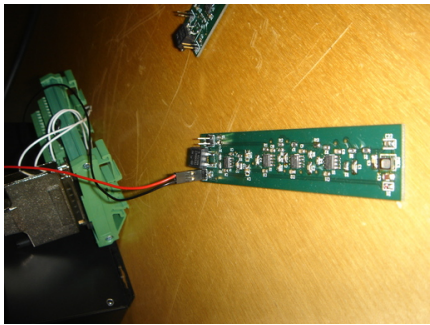
Looking at the images on figure 8.1 on the next page, we can see the prototype built. This is far from being ready to be deployed in the field, but well suited for laboratory testing. We discuss its application compared to the initial specifications.

8.1.1 Notes on the first prototype

We have now built a system that should be able to meet all the criteria set forward in table 1.2 on page 7. Recreating the table with the found components we get table 8.1 on the next page.



(a) All hardware components



(b) Close-up of the microphone and filter

Figure 8.1: First prototype of the whole system

Items 1-3	Four channel PCI A/D-board which can sample at 2 MHz with 14 bit resolution.
Items 4-5	Custom bandpass filter created which should meet all requirements.
Items 1-5	Microphone has “good” characteristics at least up to 60 kHz.
Items 8-16 and 17-18	All software considerations will be addressed in the following chapters. Mobility has been achieved as far as the current system is concerned, though the battery issue still needs consideration.

Table 8.1: Comparing hardware to interpreted specifications

As can be seen from the table, several issues need more attention. New microphones should be considered, the filter needs to be improved and made more flexible and the battery situation needs to be addressed.

8.2 Concluding remarks about the built hardware

It is difficult to conclude anything at this stage, more testing is needed, so most conclusions regarding the current hardware prototype will be referred to Part IV on page 71.

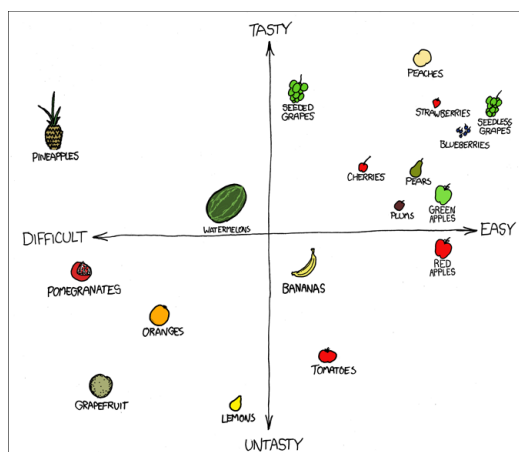
We have now built a fairly flexible and scalable base which can easily be expanded, with possibility for drop-in replacements almost everywhere in the system. So some of the design goals have clearly been reached at this point.

Part III

Software

Chapter 9

Operating system on the node computer



As discussed earlier, the choice of operating system for the node computer are limited by the A/D-board driver support. So if we are determined to use an open source operating system, the only choice is the Linux kernel [61], we are not limited to any specific distribution as we have the source code for the driver. Needless to say there are many options here, even though most of them have the same core elements.

9.1 Choice of Linux based operating system

There are many different ways to go here. There is the LFS (Linux From Scratch) [62] way, which maximises flexibility, but is rather time consuming to configure. There are several solutions oriented toward small-system

deployment e.g. DSL (Damn Small Linux) [63], and then there are the we-do-everything-for-you solutions like Ubuntu, Fedora and openSUSE [64–66].

It is clear that we would rather use a fairly complete distribution, which supplies most audio and data analysis packages as binaries. This would make experimenting and prototype development easier.

9.1.1 Criteria for the operating system

Besides running the Linux kernel, the system should be POSIX compliant*, such as including the tools provided by the FSF GNU project [67]. A well developed and well populated software package system is also a big advantage. Being able to deploy the system in batch-mode, would make it easy to install new node computers. It should not be too bloated, which makes it difficult to configure, and would probably limit available resources. These criteria are summarised in table 9.1.

Criteria
POSIX compatible
Software package system
Batch-mode deployment
Not too bloated

Table 9.1: Node computer operating system criteria

It seems pretty clear that to meet these criteria a distribution is needed, which one, is discussed below.

9.1.2 Choosing distribution

The smaller systems like LFS and DSL would be good choice, if not for the “software package system” requirement. The everything-but-the-kitchen-sink choices would not be able to meet the not-bloated criterion, nor would they be easy to deploy in batch mode. We need something that lies in between.

Although not considered seriously during this project, some kind of Live Media might serve our purpose very well. These could be configured and plugged directly onto the IDE or USB bus. The system could then be booted off of this media. With our chosen design, the node computers do not need to write a whole lot of data, only relay it when requested. The options that Live Media presents should be explored further.

*IEEE 1003.n family of standards, again unavailable to the public.

The best candidate found in this middle-ground is Debian GNU/Linux [68], it is POSIX compatible, their `apt-get(1)` software package system is superior to most, batch-deployment is fairly straightforward with `debootstrap(1)` and installing small systems is easy. Thus this is the operating system chosen for the node computer. It now remains to choose which branch of this operating system to use, the most common are described in the following subsections. We are currently running with the unstable branch, but this may change in the future.

Stable branch

This branch only allows for software known to work, its security updates are always current, and back-ported if necessary. The result from these criteria is that much of the software is outdated and lacking in features.

Testing branch

This branch is a compromise between the Stable and the Unstable branches. Packages are migrated here faster than into Stable, but much slower than into Unstable, usually the software is up to half a year behind the releases from the various projects.

Unstable branch

Here the packages are usually updated within a week of upstream release, and, if appropriate they track the unstable releases. This often means SCM (source code management [69]) checkouts of the latest source code.

The benefits are that you always have the latest features and bug fixes, and the drawbacks are that you have software that might crash, or be unstable in some other way.

9.2 Deployment of software

To get a feel for what we are dealing with a manual install was done, after that experiments with automatic install procedures were done. This would come in handy if we were to install multiple node computers, which would be likely if a 16 channel system is to be built. It would be preferable that this process could be automated as much as possible.

9.2.1 Manual deployment of Debian GNU/Linux

The first install of the operating system was done manually with CD-ROM images, i.e. the machine was booted with a Debian GNU/Linux CD, which contained a Debian 3.1 net-install image which can be found here [70].

The stepwise process of installing Debian GNU/Linux on the node computer is described in table 9.2. This would create a fairly usable system.

1. Boot the system with a Debian image, net install CD is fine.
2. Partition hard drive and mount partitions through the menu driven installer.
3. Install base system, and exit the menu driven installer.
4. Choose a branch to follow, see section 9.1.2 on page 52.
5. Move the system to the chosen branch by correcting */etc/apt/-source.list* and running `apt-get dist-upgrade`.
6. Install necessary software with `apt-get(1)`.
7. Install driver for the A/D-board.

Table 9.2: Manual install of Debian GNU/Linux 3.1

9.2.2 Automatic deployment of Debian GNU/Linux

With the experience gathered with the manual install process, it was not difficult to begin contemplating automatic install procedures. To allow for automatic install the hard drive should be connected to an already installed system, e.g. through USB. We can then use `debootstrap` to handle installation of base system, i.e. kernel, init tools and package system.

The procedure developed so far, and described below, is not fully automated yet, but this is only a matter of writing a script with built-in fail-safes. The procedure described below has been adapted from [71].

Automatic deployment how-to v. 0.1

Connect hard disk to a running system which supports `debootstrap`, in our case we used an Ultra Products adapter [72], see figure 9.1 on the facing page. Partition the hard disk, e.g. create a swap partition of a couple of GiB, and

a system partition of the rest. This could be automated with `fdisk(1)`, but has not been done yet.



Figure 9.1: Picture of 2.5” IDE hard disk connected through USB

Run `debootstrap`, the run should contain the following arguments: distribution, mount point and which Debian mirror to use. An example can be seen below.

```
debootstrap etch /mnt http://mirrors.dotsrc.org/debian/
```

The above example uses the Etch branch which is the current codename for the Stable branch, we assume that the hard disk has been mounted at `/mnt` and we use the Danish Dotsrc mirror.

We now have a working init system and `apt-get` installation, we proceed as follows:

```
mount -t proc none /mnt/proc
chroot /mnt
```

This means mount the host `proc` file system within the new file system, we then start a new shell from within the installation made by `debootstrap`. The next step is to install software packages, it has been found that a substantial number of packages should be installed to make the system properly usable, these are listed in table 9.3 on the next page. Install these with `apt-get(1)`.

To hash the locales database run the following and choose which character sets should be hashed, it is best to have both ISO-8859-1 [73, 74] and UTF-8 [75, 76] support:

CHAPTER 9. OPERATING SYSTEM ON THE NODE COMPUTER

Package name	Package description
locales	locales support, needed by libc
console-data	better keyboard support
console-common	better keyboard support
linux-image-2.6-686	Linux kernel
usbutils	USB utilities
pciutils	PCI utilities
bzip2	decompress bzip archives
sysfsutils	sysfs query tool
dhcpc3-client	DHCP client
resolvconf	DNS setup
acpid	ACPI daemon
acpi	ACPI client
sudo	Privilege separation
grub	boot manager

Table 9.3: Node computer needed software packages

```
dpkg-reconfigure locales
```

We now proceed with configuring the system, first the network. Set the hostname:

```
echo node-computer > /etc/hostname
```

Assuming that the hostname of the node computer should be **node-computer**. Initialise */etc/hosts*:

```
echo 127.0.0.1 localhost > /etc/hosts
```

Set the ethernet interface to use DHCP for address setup:

```
cat <<EOT > /etc/network/interfaces
iface lo inet loopback
iface eth0 inet dhcp
auto lo eth0
EOT
```

This sets up both the local loop interface and the ethernet interface *eth0* to use DHCP, the assumption is that there is a DHCP server running on the connected network, probably on the master computer. We can use DHCP to set the local address, which gateway to use and which name resolvers to use (DNS).

Set the time zone for the system:

```
cd /etc
rm -f timezone
ln -s /usr/share/zoneinfo/LOCAL_TIME_ZONE timezone
```

Where *LOCAL_TIME_ZONE* is replaced by the time zone that the system will be working in, e.g. *Europe/Copenhagen*. Initialise mounting of file systems (*/etc/fstab*):

```
cat <<EOT > /etc/fstab
/dev/hda1 / ext2 defaults,errors=remount-ro 0 1
/dev/hda2 none swap sw 0 0
proc /proc proc defaults 0 0
EOT
```

This file includes several assumptions, first we assume that *hda* is the hard disk to be used, and that the partition layout is as listed. Also the file system type is assumed to be *ext2* [77].

Onwards to user management, run **passwd** to set the *root* password. To add a non-privileged user and add him to the *adm* group, we use **adduser(1)**. We also enable the user to run privileged commands with **sudo(1)** by adding the *adm* group to the */etc/sudoers* file:

```
adduser new-user
adduser new-user adm
echo "%adm ALL=(ALL) ALL" >> /etc/sudoers
```

All that is needed now is the boot loader **grub(8)**, see also [78, 79]. This is not straightforward, and still needs some work, the procedure so far is copy the stage binaries into */boot/grub* and create a viable *menu.lst* file:

```
mkdir /boot/grub
cp /usr/lib/grub/i386-pc/stage[12] /boot/grub
cat <<EOT > /boot/grub/menu.lst
timeout 5
color cyan/blue white/blue

title Debian GNU/Linux, kernel 2.6.22
root (hd0,0)
kernel /vmlinuz root=/dev/hda1 ro
initrd /initrd.img

title Debian GNU/Linux, kernel 2.6.22 (single-user mode)
root (hd0,0)
kernel /vmlinuz root=/dev/hda1 ro single
initrd /initrd.img
EOT
```

Assuming */vmlinuz* points to a *2.6.22* kernel.

CHAPTER 9. OPERATING SYSTEM ON THE NODE COMPUTER

Now start the `grub` shell by running `grub`, within this shell run the following:

```
grub> install (hd1,0)/boot/grub/stage1 (hd1) (hd1,0)/boot/ ↵  
grub/stage2 (hd1,0)/boot/grub/menu.lst
```

Here $(hd1,0)$ refers to the first partition of the mounted hard disk, to automate this, we need a way of determining what reference number the attached disk receives. The `menu.lst` file refers to $(hd0,0)$ which is the number the disk should receive when inserted into the new system.

The system needs the A/D-board driver, it has only been manually installed so far, but automation should be possible.

The hard disk should now be ready to use, unmount partition and disconnect the drive. Connect it to the node computer and turn it on.

9.3 Concluding remarks

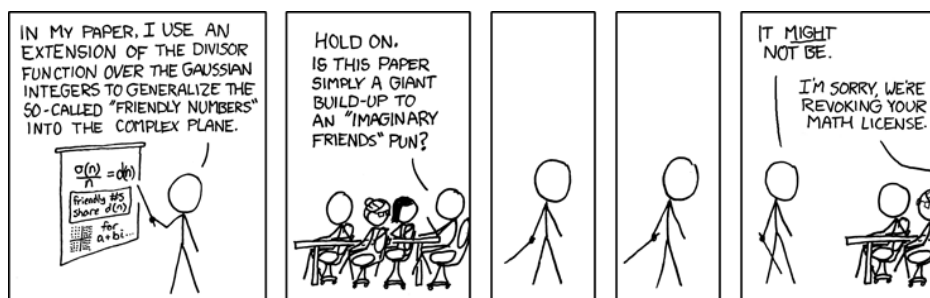
The Debian GNU/Linux has proved to be a very flexible and functional choice. There have been no software issues since we installed the system.

With our auto-deployment method (section 9.2.2 on page 54) we are confident that this should be an easy and cheap way of mass producing such systems, even with limited resources.

Given that we already fetched the kernel headers to be able to compile the kernel module, it is no great leap to compile the whole kernel ourselves. This would bring in some great advantages, we could remove all unused kernel code and modules and thus decrease boot time and memory footprint.

Chapter 10

A/D-board software



There are two central software issues to solve with regards to the A/D-board. First there are the driver issues, where we have been “lucky” enough to get the source code by signing a NDA [80]. Second we need to write various sampling programs to copy recorded data onto another media, e.g. hard disk or network.

10.1 Driver software

Here we discuss first how to compile the module and the issues with doing this. Then we discuss how to insert this module into the kernel and how to do this automatically at boot time.

10.1.1 Compile kernel module

First of all we need to make the node computer capable of compiling the driver module, and since we have settled on which operating system and distribution to use (section 9.1.2 on page 52), this is as simple as uttering:

```
apt-get install linux-headers-686
```

Prepend `sudo` if higher privilege level is required. That out of the way, we need to apply some minor patches to the driver source tree, because of the NDA these can not be made public.

Apart from the patches mentioned above, the driver source compiles without a hitch, thus creating a `daq2010.ko` module file. Now copy the module to the modules directory, this can be done like this:

```
mkdir -p /lib/modules/$(uname -r)/other
cp daq2010.ko /lib/modules/$(uname -r)/other
depmod -a
```

After running `depmod(1)` the module should be present in the `modprobe -l` output, and thus be usable with the running kernel.

10.1.2 Loading module

These steps have been derived from the installation scripts bundled with the driver software package received from Adlink. This was done because the 200+ line script called `dask2k_inst.pl` was pretty incomprehensible, and in the process of learning what it does, the steps below were deduced.

Copy firmware files to `/etc/pdask_2k/fw`:

```
mkdir -p /etc/pdask_2k/fw
cp *.rbf /etc/pdask_2k/fw
```

Open `/etc/rc.local` and add the following:

```
modprobe daq2010 BufSize=14336,3584,0,0

DAQ_DEVICE="/dev/DAQ2010W0"
mknod      $DAQ_DEVICE c $(grep daq2010 /proc/devices | cut -<
    -d' ' -f1) 0
chgrp audio $DAQ_DEVICE
chmod ug+rw $DAQ_DEVICE
```

The first line insert the module, setting the **BufSize** variable, these numbers should match internal buffers on the A/D-board, and were derived from [81]. In some cases the module needs to be reinserted, this is something that needs to be improved upon.

The last four lines create and set privileges for the `daq2010` device file, this should also be possible to do through the `udev` [82] kernel subsystem and should be investigated further. The most interesting command in those four lines is the call to `mknod` which creates the device node, `c` indicates that it should be a character device, the major number is found with `grep(1)` from `/proc/devices`, `cut(1)` is used to extract the number assigned to the module. Thus the extracted number is the major number, which ties the device to the

module and hence to the A/D-board, the last number is the minor number i.e. zero because we have only one device.

10.2 Compile module library (API)

Here again we have two things to do before the library works, we first compile it, and then make it known to the runtime linking system.

10.2.1 Compile library

In the current system this works out-of-the-box, i.e. it is enough to do the following:

```
cd daq2010_linux26_drvsrc/lib/  
make
```

10.2.2 Install library

Again the process here is straightforward as with any other library, copy the library to a `ld(1)` watched directory, and run `ldconfig(1)`, i.e. :

```
cp libpci_dask2k.so /usr/local/lib  
ldconfig
```

This assumes that a line containing `/usr/local/lib` is present in either the `/etc/ld.so.conf` file or the `/etc/ld.so.conf.d/` directory tree.

10.3 Sampling software

Sampling the channels on the A/D-board requires writing a program that interfaces the API (application programming interface) provided by Adlink. The API then forwards the call to the kernel module, which in turn queries the hardware, see figure 10.1 on the following page.

10.3.1 What is needed

To be able to sample a channel, we need to know which channel to connect the input to, this information can be found in the user manual [81], which shows that pins 1-4 and 35-38 are to be used for four channel sampling and pin 39 is used as reference ground. When everything is correctly connected, we need to construct a program that correctly interfaces with the library provided by

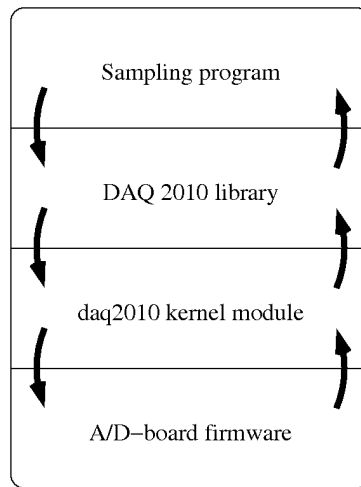


Figure 10.1: Software layers used when sampling

Adlink. After researching the User's Guide [83] and the Function Reference Manual [84] it is apparent that this program should be implemented in the *C* programming language [85, 86].

Depending on the requirement there are many different ways to sample the analogue signal: unipolar-, bipolar-, software-triggered-, single-buffered- and double-buffered sampling. All of these can be mixed when it makes sense, and all of them have hardware support on the A/D-board.

10.3.2 Sampling program implementations

Here are listed the significant parts of two of the successful implementations, i.e. a software triggered sampling program and a double-buffered sampling program. The full listing of these programs can be seen in Appendix C.1.2 to C.1.3 on pages 121–122.

Software triggered sampling program

The pertinent bits to make this work are described in the following. First we register the device, arguments are *card type* and *card number*, so if you only have one card this would become zero:

```
14 }

```

Configure the registered card, arguments are *card number*, *channel* and *A/D range*:

```
17 if(D2K_AI_CH.Config (daq_card , 1, AD.B.10.V) != NoError)

```

We are now ready to use the card, this can be done by running:

```
24 D2K_AI_VReadChannel(0, 1, &sample1);
```

Which saves a voltage-interpreted value into the **sample** variable. First and second argument refer to *card number* and *channel number*.

Double buffered sampling

This is a bit more complicated as it requires setting up two buffers for the library to fill up in circular mode, i.e. fill up first buffer, set a signal, fill up next buffer, set a signal, start from beginning. This process will run separately from the program, so the program can work on the buffer not currently being written to. It is important that whatever operation is to be performed, it should be completed before the “next” buffer is full.

Here follows a run-down of what is needed to get this to work. Allocate memory:

Register card:

```
35 U32 ai_iterator;
36 int i;
```

Configure all channels to bipolar ± 10 V operation, and configure sampling mode, refer to [84, p. 21-24] for description of the arguments:

```
38
39 if( signal(SIGINT, &trap_sigint) == SIG_ERR)
40     error(-1, 0, "Unable to register signal handler");
```

Now tell the library about the allocated buffers:

```
42 if(debug) fprintf(stderr, "Registering card\n");
43
44 if( (card = D2K_Register_Card(DAQ_2010, 0)) < 0)
45     error(-1, 0, "Registering card error");
```

Set the A/D-board to continuous operation, reading all channels:

```
48 || D2K_AI_Config(card, 0, 0, 0, 0, 0, 1) != 0)
49     error(-1, 0, "Error while configuring channels");
```

Spin-lock until a buffer is ready to be operated on, then operate on it, in this case write it to disk:

```
51 if(D2K_AI_AsyncDblBufferMode (card, TRUE) != 0
52     || D2K_AI_ContBufferSetup(card, ai_buf_first,  ←
        ALBUFFER_SIZE, &buf_id) != 0
```

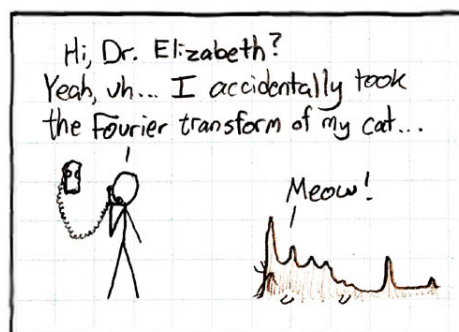
```
53     || D2K_AI_ContBufferSetup(card, ai_buf_second, ↵  
        AI_BUFFER_SIZE, &buf_id) != 0)  
54     error(-1, 0, "Error while enabling double-buffered mode");  
55  
56     time_stamp();  
57     if(D2K_AI_ContReadMultiChannels(card, 4, chans, 0, ↵  
        AI_BUFFER_SIZE/4, SAMPLE_RATE_MAGIC_NUMBER, 0, ASYNCHOP ↵  
        ) != 0)  
58         error(-1, 0, "Error while calling ContReadMultiChannels");  
59  
60     while(1) {  
61         ai_iterator = 0;
```

10.4 Concluding remarks

We have reaffirmed that proprietary software limits the choices we have, i.e. no publishing of patches. That aside and except for some minor issues encountered when compiling the module, the A/D-board, module and library have all worked well so far, so I would conclude that the Adlink product still looks like a good candidate for the final product deployment, barring that a viable open source alternative does not rear its pretty head.

Chapter 11

Handling of recorded data



This chapter is dedicated to what we do with the recorded data. This means considering which software solutions are at our disposal. Which algorithms to use, both for finding the signal in the recorded data and when tracking the signal in three dimensions. We also discuss what is needed to be able to synthesise the original signal. Auto-calibration of the system is considered. Some of the software requirements made in the original specifications are discussed.

11.1 Analysis software

There are several solutions when it comes to signal analysis, those that have been considered for this project are: `Matlab` [87, 88], `Octave` [89, 90] and `PDL` [91, 92].

11.1.1 Matlab

`Matlab` definitely is able to solve the tasks we have in mind, but this would restrict us when we want to deploy this system to many nodes. Bearing

this in mind the decision to move to an open source solution was made, the natural move would be to `Octave`.

11.1.2 `Octave`

`Octave` is considered a `Matlab` clone in as much that they try to parse `Matlab` code and thus be API compliant. `Octave` is fully open source and is able to do all the calculations that we need, thus would be an acceptable choice.

11.1.3 PDL or Perl Data Language

PDL makes it possible to import arbitrarily large data-structures* and operate on them in-place, thus it is place conserving and fast. As with the options discussed above, it also has built in *Fourier* transform functions which will be needed when generating spectrograms.

11.1.4 Conclusion about analysis software

As I have never been quite satisfied with the `Matlab` way of doing things, I went forward in search for an alternative. Being a fan of the perl [93] programming language, PDL looks like a very good alternative. It allows for creating so called “piddles” which can contain N -dimensional data structures, it is fully object oriented and makes it easy and fast to do operations on “piddles”, e.g. the `++` operator can be applied to increment all numbers in a “piddle” in-place. Thus PDL was chosen as the primary analysis software.

11.2 Signal detection

It is fine to talk about comparing signals received by different receivers, but one problem remains, which is how do we determine that a detected signal in one receiver is the same as one recorded by another receiver. Methods considered here are the one presented in [94] and a custom developed method.

11.2.1 Two layered wavelet tree

This method was initially chosen on the merits of its abstract, after researching it further it was found to be rather complicated to implement, and given the time constraints this approach was postponed. It should certainly be explored further as it may prove to be a superior signal detection algorithm.

*Barring we run out of memory.

11.2.2 Custom algorithm

This algorithm looks at the data-stream creating an average as we go, if the signal varies much from this average a signal detection flag is raised, the width of the signal is determined and a spectrogram is calculated. If this method can run in real-time is yet to be seen.

An optimised version was made that picks out hundred random indices and uses their average as the overall chunk average, and this proved to be quite accurate. Further testing is needed.

11.3 Synchronisation

When operating a distributed system, such as the one we are building, and trying to use TDOA methods, it is imperative that the systems agree on a common time reference. This could be achieved by running NTP (network time protocol [95, 96]) servers and creating timestamps by conferring the originating nodes NTP server.

11.4 Comparing signals

Assuming we have detected the start and end of a signal correctly, how do we go about comparing it to another signal recorded in another receiver? This is complicated by multiple issues, is the source stationary or is it mobile, is the source directional and is the source an echo.

11.4.1 Stationary or mobile

If the source is moving towards or away from us the signal will be Doppler shifted [97, 98], i.e. if moving towards us, even at an angle, the frequency will be higher, and the opposite if the source is moving away from us. What this means for comparing signals that may be recorded in any direction relative to the source, is that we need to take into account that the same signal may have stretched or compressed spectrograms, so the possibility of false-negatives is high. What is needed is a threshold that alleviates the situation.

11.4.2 Directionality

Bats emit different sounds depending on what they are doing, searching, approaching or snatching up prey [15], also the direction in which most of the sound energy is emitted, narrows a lot when a prey is approached, thus

making it harder to track the source if the microphones are far away from the prey. This might result in microphones that are relatively close together, do not hear the same signal.

11.4.3 Signal echo

As with all signals, they bounce back from surfaces, the amount of echo will depend on the type of surface that the signal bounces off of. Thus we need to take measures that filter echoes out of our dataset. This could be based on signal travel-time and perhaps direction of the signal.

11.5 Location algorithms

There are many ways to utilise TDOA (see 1.2.2 on page 5). The methods examined here are the spherical and hyperbolic methods. These methods are discussed further below.

11.5.1 Spherical algorithm

The spherical algorithm assumes that the signal has been spread homogeneously through the media, and so extending spheres around the receivers, would in principle all intersect at, if enough receivers “heard” the signal, one point in space. The trouble with this method is that it results in an NP complete problem which needs heuristics to solve, one such heuristic is proposed in [99]. This method is rather unstable, i.e. small changes in measurement might change position drastically in edge conditions.

11.5.2 Hyperbolic method

It is well known that TDOA between a pair of microphones locate the source on a hyperboloid of revolution with foci at the microphones. Examining [100, 101] it can be concluded that finding a position of a source in three dimensions needs at least four microphones, increasing that number will improve accuracy.

The hyperbolic method in 2-D is as follows: we have N receivers distributed arbitrarily in a 2-D plane. Let $\mathbf{d} = [d_{2,1}, d_{3,1}, \dots, d_{N,1}]^T$ where $i \in 2, 3, \dots, N$ denote the time delay vector, which represents the delay between receiver i and the first receiver. If the unknown position is (x, y) and the microphone locations are denoted (x_i, y_i) , the squared distance between the source and receiver i is:

$$\begin{aligned} r_i^2 &= (x_i - x)^2 + (y_i - y)^2 \\ &= K_i - 2x_i x - 2y_i y + x^2 + y^2 \end{aligned}$$

Where $i = 1, 2, \dots, N$ and $K_i = x_i^2 + y_i^2$. We now have the equations we need to find (x, y) :

$$r_{i,1} = cd_{i,1} = r_i - r_1$$

Where c is the speed of sound. This defines a set of non-linear equations which, if solved, give the value for (x, y) .

11.6 Synthesis of signals

The success of synthesising the original signal comes down to how good our detection algorithms are, and how good are we at comparing signals (11.2 to 11.4 on pages 66–67).

The strategy here is to determine where the bat is and which way it is facing, the microphone closest would have the most accurate recording. If we also know the speed of the bat, we can correct the signal for Doppler shift.

11.7 Auto calibration

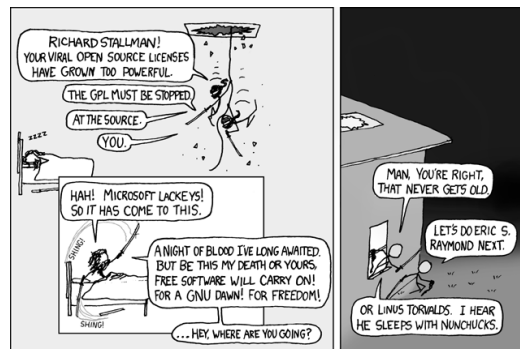
Auto calibration of the system refers to that the system should be able to determine the positions of all the receivers automatically, i.e. no manual measurements. This could be achieved in several ways, the proposed strategies here are: have a speaker emit an ultrasonic pulse at each receiver and employ the same TDOA approach as for the sources, this could be done either by walking around with a speaker to each receiver or attach a speaker to each receiver. Put a very directional speaker onto a stick, rotate the speaker such that it sends out pulses in a semi-sphere, this would allow for determining direction and distance to each receiver.

Part IV

Using the system

Chapter 12

Verification tests



This chapter concerns component testing, i.e. verify that the various developed components work as designed. Thus several different test setups are made which test a specific feature of the system or a specific component.

12.1 Timer testing

It is hard to get trusted timers in a non-realtime system, there have been made tests to see how trustworthy they might be.

12.1.1 Timer based on usleep(3)

Letting the timer do sleeps for 1 μ s to 1 ms results in the plot seen on figure 12.1 on the following page.

The program that generated this figure can be seen below.

```
#include <stdio.h>
#include <time.h>
#include <my_utils/timer.h>
```

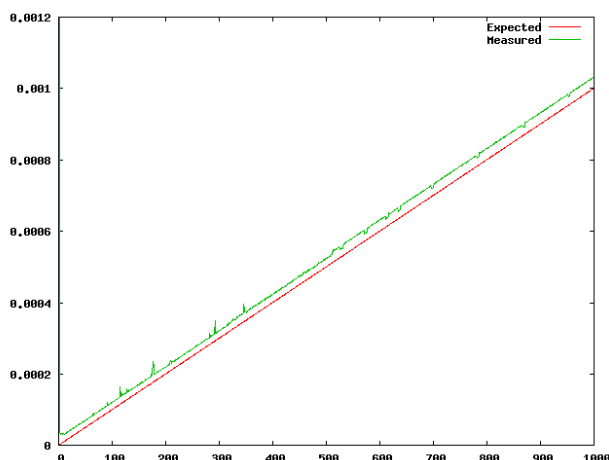


Figure 12.1: Test of `usleep(3)` timer routine

```

5 | #define _USE_XOPEN_EXTENDED
   | #include <unistd.h>                /* usleep */
   |
   | #define NUMBER_OF_MEASUREMENTS 1
10 |
   | int main(void) {
   |     int i;
   |     double sleep_length;
   |     double measurements[NUMBER_OF_MEASUREMENTS];
15 |     double average;
   |
   |     printf("set terminal png\n");
   |     printf("set output 'timer_test.png'\n");
   |     printf("set yrange [ 1E-6 : 1200E-6 ]\n");
20 |     printf("plot '-' with lines title 'Expected', ");
   |     printf("'-' with lines title 'Measured'\n");
   |     for(sleep_length = 1; sleep_length < 1000; sleep_length += ←
   |         1)
   |         printf("%.9f\n", sleep_length * 1E-6);
   |         printf("e\n");
25 |
   |     for(sleep_length = 1; sleep_length < 1000; sleep_length += ←
   |         1) {
   |         for(i=0; i<NUMBER_OF_MEASUREMENTS; i++) {
   |             time_stamp();
   |             usleep(sleep_length);
30 |             measurements[i] = time_elapsed();
   |         }
   |     }

```

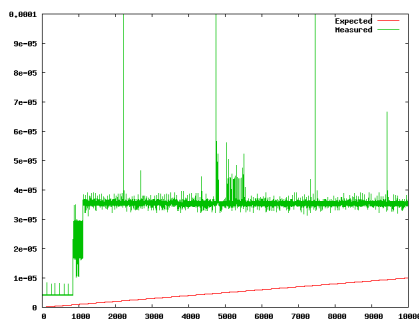
```

    for(i=0, average=0; i<NUMBER_OF_MEASUREMENTS; i++)
        average += measurements[i];
35  average /= NUMBER_OF_MEASUREMENTS;
    printf("%.9f\n", average);
    }
    printf("e\n");
40  return(0);
    }

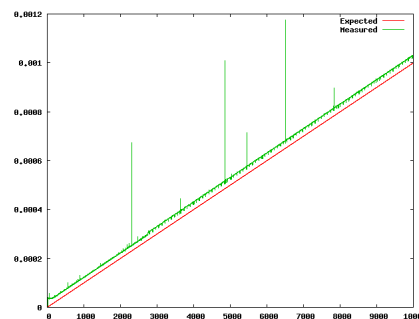
```

12.1.2 Program based on nanosleep(3)

This uses the custom written support library listed in Appendix C.1.1 on page 116. This routine was tested for two intervals, 1 ns to 10 μ s and 1 ns to 1 ms. The test results can be seen in figure 12.2.



(a) 1 ns to 10 μ s



(b) 1 ns to 1 ms

Figure 12.2: Test of nanosleep(3) timer routine

12.1.3 Timer conclusions

None of the timer routines tested are trustworthy in their lower reaches, but become increasingly accurate when waiting for longer time. Given the data above, `usleep(3)` seems to be more stable.

12.2 A/D-board

12.2.1 Sampling channel

Using the software triggered version of the sampling program (Appendix C.1.2 on page 121). This test sets the program to output measurements to the console, the newest sample is always at the bottom. We then apply a voltage to the A/D-converter channel being measured.

Figure 12.3 shows the setup before applying any voltage, and figures 12.4a to 12.4d on the next page show the result of applying 1, 6.7, -1 and -6.7 V to the channel.

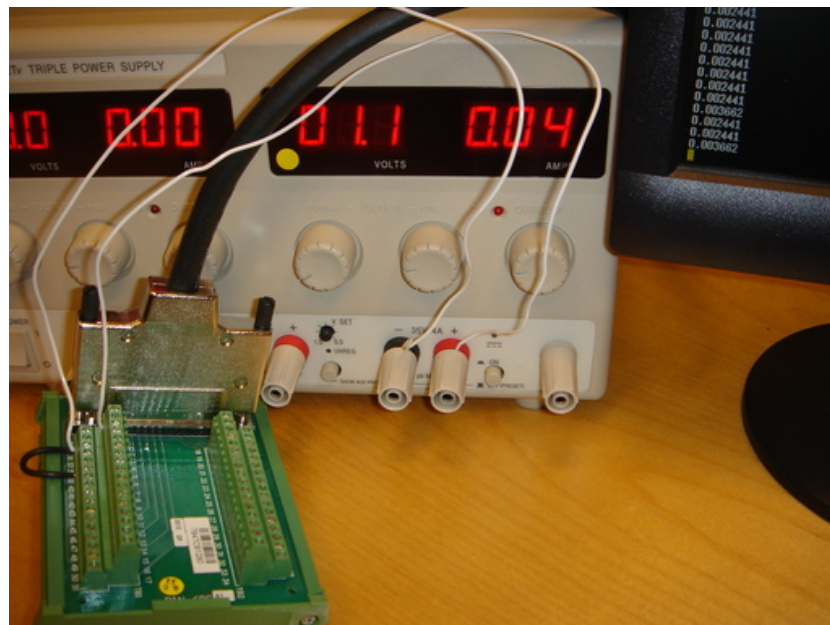


Figure 12.3: Sampling verification setup, no voltage applied

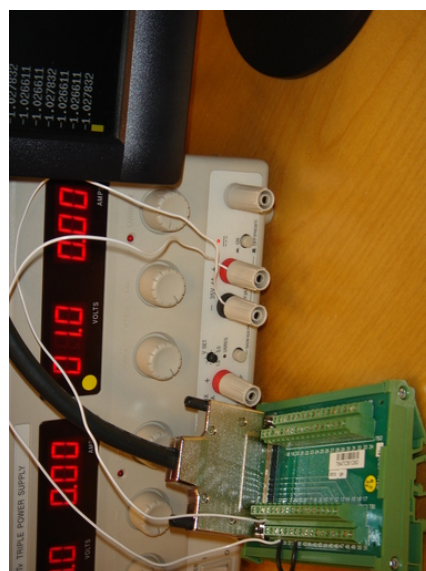
12.3 Analysis software

12.3.1 Spectrogram generation

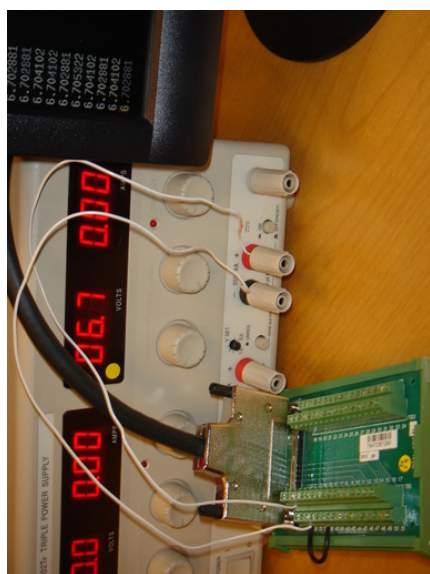
To ensure that the spectrogram generation with *PDL* is working properly a test has been devised, which compares the output from *octave* to that of the *PDL* library. To this end a specific time series from recordings done in the Biology Department by a Belgian scientist group has been chosen. The



(a) Applying 1 V



(b) Applying -1 V



(c) Applying 6.7 V



(d) Applying -6.7 V

Figure 12.4: Verifying that sampling works as expected

CHAPTER 12. VERIFICATION TESTS

numbers can be found on the accompanying optical disk. A chirp was found on this recording at approximately 120 ms on the 6th channel, see figure 12.5

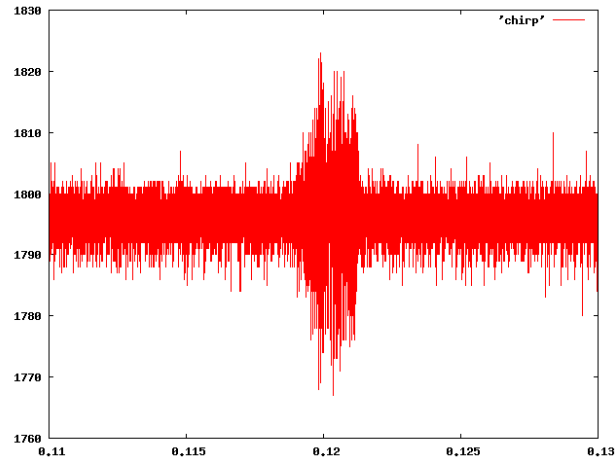


Figure 12.5: Time series of a chirp

GNU Octave version

The octave result can be seen on figure 12.6, the program that generated this spectrogram can be seen below.

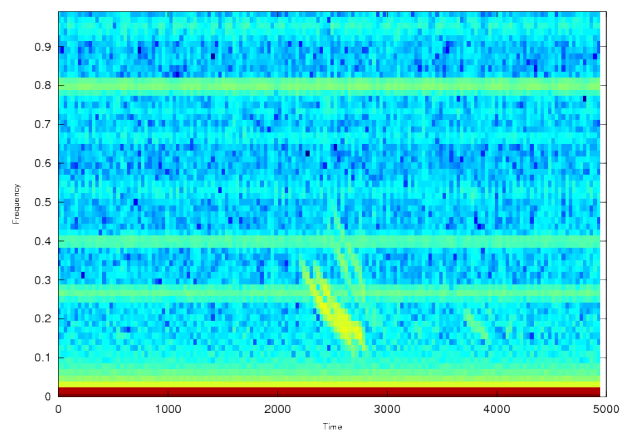


Figure 12.6: Chirp spectrogram generated by GNU Octave


```

% Author:  Thor Andreassen <ta@imada.sdu.dk>
%
% Description: Create spectrogram.
5 load daubentonii.mat;
  specgram(A, 128);
  print -depsc2 daubentonii.eps;

```

PDL version

The PDL version of the same chirp can be seen on figure 12.7. The program that generated this spectrogram can be seen below.

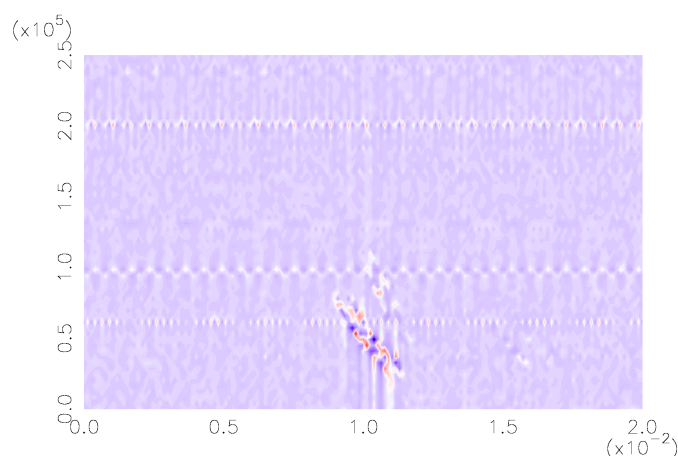


Figure 12.7: Chirp spectrogram generated by the *PDL* library

```

#!/usr/bin/perl

# Author:  Thor Andreassen <ta@imada.sdu.dk>
#
5 # Description: Calculates spectrogram of a given time-series, ←
  and saves it.

use strict;
use warnings;
use PDL;
10 use PDL::NiceSlice;          # better slice syntax
   use PDL::Graphics::PLplot;
   use PDL::FFT;

```

```

use PDL::Lvalue;

15 use constant DELTA_T      => 2e-6;

my $data_file = $ARGV[0];

my @data;

20 @data = rcols("$data_file", 0);

my $seq = sequence($data[0]->nelem);
my $spectrogram = PDL->new([]);
25 my $window_size = 2048;
my $window_middle = $window_size / 2;

for ($seq->(0 : -$window_size - 1 : $window_size / 2)->list) {
    my $subsample = $data[0]->($_ : $_ + $window_size - 1)->↵
        sever;
30     realfft($subsample);

    # Calculate magnitude
    for my $i (0 .. $window_middle - 1) {
        $subsample->($i) =
35         sqrt($subsample->($i)**2 + $subsample->($i + ↵
            $window_middle)**2);
    }
    $spectrogram =
        $spectrogram->append($subsample->(1 : $window_middle - 1) ↵
            ->transpose->copy);
}

40 my $pl = PDL::Graphics::PLplot->new (DEV => 'xfig', FILE => '↵
    spectrogram.fig');
$pl->shadeplot($spectrogram->( :, 1:-1), 100,
               BOX => [0, $data[0]->nelem * DELTA_T, 0, 1 / ↵
                   DELTA_T / 2]);

```

The program calculates only the magnitude of the FFT and ignores the phase data. A spectrogram that covers the whole second of the recording can be seen on figure 12.8 on the next page. If we look carefully at this figure we can see multiple chirps throughout the recording.

Conclusions on spectrogram generation

The *PDL* library seems to make as good a job as *octave*, although some more source code must be written, as no spectrogram library functions were found.

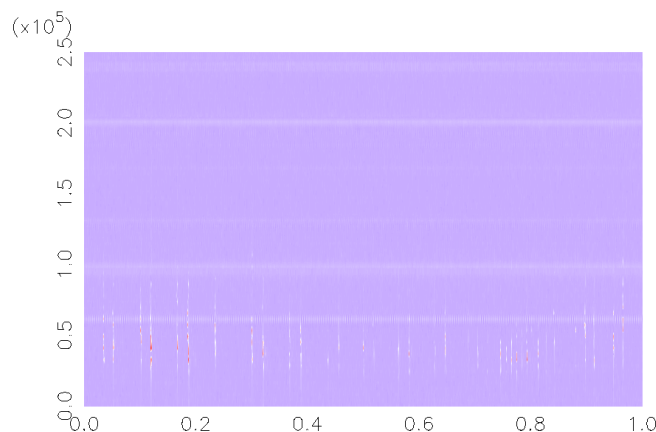


Figure 12.8: Spectrogram of the whole recording

12.4 Filter verification

Many tests were made with the filter, mostly with a signal generator and an electronic oscilloscope. The oscilloscope was able to calculate the FFT and send it to the computer, this was done for several frequencies, so as to be able to create a topographical landscape of the transfer function. As it turns out the captured data was not saved as expected, and there is no time to recreate them, thus no figure.

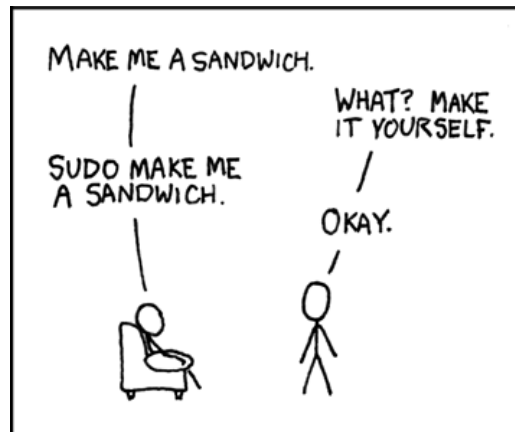
The filter was clearly verified to work at the time, seeing the FFT of the signal from the signal generator be attenuated away outside the passband.

12.5 Microphone verification

This has been promised in several places throughout the report, but the microphones have never been properly put to the test. This is mostly because other issues were more pressing.

Chapter 13

System testing



13.1 Bandwidth testing

The double buffered sampling program listed in section C.1.3 on page 122, has been set to run for one second, ten seconds and sixty seconds. All these tests went well and indicated that the system was performing in near real-time.

The next test would be to see if it is possible to write all the recorded data to a network device.

13.2 Compute time left

When operating in double-buffered mode, it becomes interesting how much time we have left over to analyse the data while the other buffer is being

filled up. This comes down to how big a buffer you allocate, and how much you are willing to let the calculations lag behind real-time.

Tests were made with a program adapted from the double-buffered sampling program, it can be seen below.

```

5  /*
   * Author: Thor Andreassen <ta@imada.sdu.dk>
   *
   * Description: Compute time test.
   */

#include <stdio.h>
#include <stdlib.h>
#include <error.h>
10 #include <signal.h>          /* sigint */
#define _USE_XOPEN_EXTENDED
#include <unistd.h>
#include <daq2010/d2kdask.h>
#include <my_utils/timer.h>
15
#define ALBUFFER_SIZE 1048576

enum { false, true } bool;

20 char debug = true;
    I16 card;

    void trap_sigint(int signum);

25 #define BUFFER_SIZE 10

int main( void )
{
    I16 ai_buf_first[ALBUFFER_SIZE], ai_buf_second[ ←
        ALBUFFER_SIZE], *ai_buf_active;
30    U16 chans[4] = {0,1,2,3};
    U16 active_buffer = 0;
    BOOLEAN is_half_ready, is_stopped;

    int i;
35    double times[BUFFER_SIZE];

    if( signal(SIGINT, &trap_sigint) == SIG_ERR)
        error(-1, 0, "Unable to register signal handler");

40    if(debug) fprintf(stderr, "Registering card\n");

    if( (card = D2K_Register_Card(DAQ_2010, 0)) < 0)
        error(-1, 0, "Registering card error");

```

CHAPTER 13. SYSTEM TESTING

```
45  if(D2K_AI_CH.Config(card, -1, AD_B_10_V) != 0
    || D2K_AI.Config(card, 0, 0, 0, 0, 0, 1) != 0)
    error(-1, 0, "Error while configuring channels");

    if(D2K_AI_AsyncDblBufferMode (card, TRUE) != 0
50    || D2K_AI_ContBufferSetup(card, ai_buf_first, ↵
        AI_BUFFER_SIZE, &active_buffer) != 0
    || D2K_AI_ContBufferSetup(card, ai_buf_second, ↵
        AI_BUFFER_SIZE, &active_buffer) != 0)
        error(-1, 0, "Error while enabling double-buffered mode");

    time_stamp();
55  if(D2K_AI_ContReadMultiChannels(card, 4, chans, 0, ↵
        AI_BUFFER_SIZE/4, 40, 0, ASYNCHOP) != 0)
        error(-1, 0, "Error while calling ContReadMultiChannels");

    times[i++] = time_elapsed();
    active_buffer = 0;
60  while(i < BUFFER_SIZE) {
        do {
            usleep(10);
            D2K_AI_AsyncDblBufferHalfReady(card, &is_half_ready, &↵
                is_stopped);
        } while ( !is_half_ready );
65  times[i++] = time_elapsed();

        ai_buf_active = (active_buffer == 0) ? ai_buf_first : ↵
            ai_buf_second;
        active_buffer ^= 1;
    }
70  for(i=0; i<BUFFER_SIZE; i++) {
        printf("%f ", times[i]);
        if(i==0)
            printf("\n");
        else
75  printf("%f\n", times[i] - times[i-1]);
    }

    if(debug)
        fprintf(stderr, "Releasing card and terminating\n");
80  if(D2K_Release_Card(card) < 0)
        error(-1, 0, "Release error");

    return(0);
}
85  void trap_sigint(int signum) {
    if(debug) {
```

```

90     fprintf(stderr, "Received signal: %d\n", signum);
    fprintf(stderr, "Releasing card and terminating\n");
    }
    if(D2K_Release_Card(card) < 0)
        error(-1, 0, "Release error");
    exit(0);
}

```

The idea is to get an idea of how much computation time is left with different buffer sizes. In the listings below the first column contains timestamps of buffer change and the second column shows the delta between two neighbouring timestamps. First the output when running 1 MiB buffer, i.e. 250 KiB per channel:

```

0.000107
0.260964 0.260856
0.521734 0.260770
0.782465 0.260731
5 1.043316 0.260851
1.304144 0.260827
1.565030 0.260886
1.825880 0.260850
2.086659 0.260779
10 2.347466 0.260807

```

And 500 KiB buffer:

```

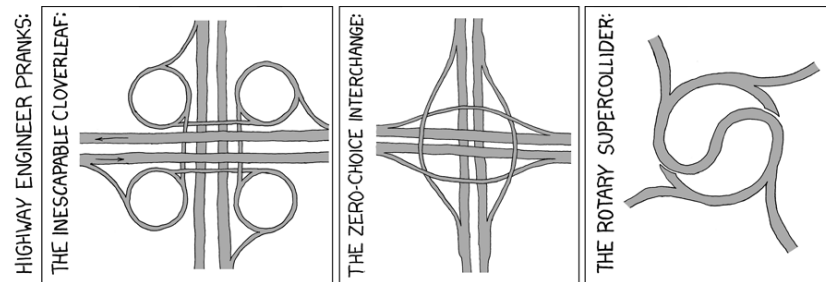
0.000115
0.130523 0.130409
0.260939 0.130416
0.391302 0.130363
5 0.521706 0.130404
0.652088 0.130382
0.782499 0.130410
0.912919 0.130421
1.043338 0.130419
10 1.173744 0.130405

```

Clearly it is a trade-off between how real-time the data is, and how much time is needed for computation. Preliminary tests with our primitive signal finding algorithm, indicated that it was quite fast. Although the complete test has yet to be performed, I would hold that we are able to do signal detection within at least a second of real-time.

Chapter 14

Conclusions



14.1 Things to improve

Here we discuss the problems that have been found in this prototype, and, when applicable, recommendations to alleviate said problems are given.

14.1.1 Hard drive failure

One major goal in the final product should be to limit any moving parts in the final product. During the project we have had a node computer hard disk fail, such incidents will become increasingly likely if such a system became mobile and jolted around. A SSD (solid state disk) would solve such issues, or even better SSD RAID. Of course it should be able to handle the temperature and humidity encountered where the system is to be deployed.

14.1.2 Mini-ITX underpowered

During the tests with the prototype system, it d difficulty starting. The unconfirmed suspicion here is that the 60 W PicoPSU does not supply enough

power to the system, at least not when it is powering up. It is recommended to increase this to e.g. 120 W.

14.1.3 Microphone oversupplied

As has been mentioned elsewhere, there has been a mixup with regards to the datasheet for the microphone that we were using. This has resulted in the microphone being supplied by just below 5.5 V, when its maximum rating is at 3.6 V.

14.2 Preliminary final version specifications

Based on the discussion so far, some preliminary specifications can be advised for the final version of the node computer, see table 14.1.

Component	Specification
Motherboard	VIA or Atom Mini-ITX motherboard
RAM	2 GiB
Hard disk	8 GiB SSD
DC-DC PSU	≥ 120 W
Bandpass filter	Chip based filter
Enclosure	Rugged case with heating elements
OS	Debian Live with hibernate support

Table 14.1: Preliminary final version specifications

14.3 Addressing items 8-16

In the original specification, several more items were mentioned, but have not been properly dealt with yet. The reason for this is that the system is not mature enough to address these issues. Although I remain convinced that the system design will allow these issues to be resolved with relative ease.

14.4 Comparing result to original abstract

In as much as the system design goes, this system is solid. But when thinking about actually solving the issues put forward in the original abstract, this system is probably one or two months from actually working as intended.

14.4.1 What needs to be done

First solve hardware issues. A verified microphone should be acquired, the filter needs some more work regarding the voltage regulator.

When the hardware is working the software algorithms discussed in Chapter 11 on page 65 need to be implemented, these include:

- Signal detection.
- Signal comparison.
- Location calculation.
- Auto-calibration.

14.5 Success or failure

Although many of the goals set forth in the beginning of this project have not been reached, I believe that most of them are much closer. Given more time this system could be completed within one or two months.

The main problem has been my trouble with limiting the scope of the project, too much depth has been tried for in too many places. That coupled with my inexperience with some of these fields has blown the time schedule to smithereens. One positive aspect of having a wide scoped project is that you learn a lot.

So has this project been successful? The decision must be in the eye of the beholder.

Bibliography

- [1] Robert Fisher. In “cvonline: Citation of contents”. Available: <http://homepages.inf.ed.ac.uk/rbf/CVonline/SUPPORT/citations.htm>, Accessed: July 24 2008.
- [2] Melvin E. Page. In “a brief citation guide for internet sources in history and the humanities”. Available: <http://www.h-net.org/~africa/citation.html>, Accessed: July 24 2008.
- [3] Oren Patashnik et al. Bibtex: Your bibtex resource. Available: <http://www.bibtex.org/>, Accessed: July 24 2008.
- [4] Bibtex. Available: <http://en.wikipedia.org/wiki/BibTeX>, Accessed: July 24 2008.
- [5] UK T_EXenthusiasts. Urls in bibtex bibliographies. Available: <http://www.tex.ac.uk/cgi-bin/texfaq2html?label=citeURL>, Accessed: July 24 2008.
- [6] David Joyner and William Stein. Open source mathematical software. *Notices of the American Mathematical Society*, page 1279, November 2007.
- [7] Michael Tiemann et al. About the open source initiative. Available: <http://www.opensource.org/about>, Accessed: July 28 2008.
- [8] Ken Coar et al. The open source definition. Available: <http://www.opensource.org/docs/osd>, Accessed: July 28 2008.
- [9] Daubenton’s bat - *myotis daubentonii*. Available: http://en.wikipedia.org/wiki/Daubenton's_bat, Accessed: August 30 2008.
- [10] Unix on-line man pages. Available: <http://unixhelp.ed.ac.uk/CGI/man-cgi>, Accessed: August 24 2008.

BIBLIOGRAPHY

- [11] Animal echolocation. Available: http://en.wikipedia.org/wiki/Animal_echolocation, Accessed: August 4 2008.
- [12] J. Rydell and J.R. Speakman. Evolution of nocturnality in bats: Potential competitors and predators during their early history. *Biological Journal of the Linnean Society*, 54(2):183–191, 1995.
- [13] Active sonar. Available: http://en.wikipedia.org/wiki/Sonar#Active_sonar, Accessed: August 4 2008.
- [14] A. Surlykke and E.K.V. Kalko. Echolocating Bats Cry Out Loud to Detect Their Prey. *PLoS ONE*, 3(4), 2008.
- [15] M. Weinbeer and E.K.V. Kalko. Ecological niche and phylogeny: the highly complex echolocation behavior of the trawling long-legged bat, *Macrophyllum macrophyllum*. *Behavioral Ecology and Sociobiology*, 61(9):1337–1348, 2007.
- [16] M.K. Obrist, R. Boesch, and P.F. Fluckiger. Variability in echolocation call design of 26 Swiss bat species: consequences, limits and options for automated field identification with a synergetic pattern recognition approach. *Mammalia*, 68(4):307–322, 2004.
- [17] Analog to digital conversion. Available: http://en.wikipedia.org/wiki/Analog-to-digital_converter, Accessed: August 15 2008.
- [18] Sophocles J. Orfanidis. *Introduction to signal processing*. Prentice-Hall, Upper Saddle River, New Jersey, 1996.
- [19] C.L. Phillips and H.T. Nagel. *Digital control system analysis and design*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 3rd edition, 1998.
- [20] Sli (start light ignition) lead acid battery. Available: http://en.wikipedia.org/wiki/Lead_acid_battery, Accessed: August 6 2008.
- [21] Nicd (nickel cadmium) battery. Available: http://en.wikipedia.org/wiki/Nickel-cadmium_battery, Accessed: August 6 2008.
- [22] Nicd (nickel cadmium) battery prices. Available: <http://www.google.com/products?q=nicd>, Accessed: August 6 2008.
- [23] Nimh (nickel metal hydride) battery. Available: http://en.wikipedia.org/wiki/Nickel_metal_hydride_battery, Accessed: August 6 2008.

- [24] Nickel iron battery. Available: http://en.wikipedia.org/wiki/Nickel-iron_battery, Accessed: August 6 2008.
- [25] Be Utility Free Inc. Ad titled: Nickel iron batteries from beutilityfree. Available: <http://www.beutilityfree.com/batterynife/Flyer.pdf>, Accessed: August 6 2008.
- [26] Nizn (nickel zinc) battery. Available: http://en.wikipedia.org/wiki/Nickel-zinc_battery, Accessed: August 6 2008.
- [27] Smd or smt (surface mount devices or technology). Available: http://en.wikipedia.org/wiki/Surface-mount_technology, Accessed: August 9 2008.
- [28] Knowles Acoustics. Acoustic interface design guide. Available: http://www.knowles.com/search/pdf/07-KA-895_designguide_final_v1.pdf, Accessed: August 9 2008.
- [29] Knowles Acoustics. Enhanced rf protected “mini” sisonic microphone specification. Available: <http://www.farnell.com/datasheets/103076.pdf>, Accessed: August 9 2008.
- [30] Knowles Acoustics. Ultrasonic acoustic sensor. Available: <http://www.farnell.com/datasheets/100382.pdf>, Accessed: August 9 2008.
- [31] Anti-aliasing. Available: <http://en.wikipedia.org/wiki/Anti-aliasing>, Accessed: August 11 2008.
- [32] Aliasing. Available: <http://en.wikipedia.org/wiki/Aliasing>, Accessed: August 11 2008.
- [33] Band-pass filter. Available: http://en.wikipedia.org/wiki/Band-pass_filter, Accessed: August 11 2008.
- [34] H. Ebert, T. Larsen, E. Øhlenschläger, S. Dahl-Pedersen, E.A. Jensen, and P. Henningsen. *Elektronik ståbi*, 2003.
- [35] Maxim Integrated Products. Maxim 4th- and 8th-order continuous-time active filters. Available: <http://datasheets.maxim-ic.com/en/ds/MAX274-MAX275.pdf>, Accessed: August 11 2008. Version notes: 19-4191; Rev. 3; 10/96.
- [36] P. Horowitz and W. Hill. *The Art of Electronics*. Cambridge University Press, 1989.

BIBLIOGRAPHY

- [37] Nuhertz Technologies. Filter solutions trial. Version 10.0, available at <http://www.filter-solutions.com/download.html>.
- [38] National Semiconductor. Computer based data acquisition. Available: <http://www.national.com/analog/adc>, Accessed: August 15 2008.
- [39] Amplicon. Computer based data acquisition. Available: <http://www.amplicon.co.uk/MandC/product/PC-Analog-83.cfm>, Accessed: August 15 2008.
- [40] Microsoft Compaq and National Semiconductor. Open host controller interface specification for usb. Available: ftp://ftp.compaq.com/pub/supportinformation/papers/hcir1_0a.pdf, Accessed: August 15 2008.
- [41] Pci express. Available: http://en.wikipedia.org/wiki/PCI_Express#Data_transmission, Accessed: August 15 2008.
- [42] Amplicon. Pci based adc products. Available: <http://www.amplicon.co.uk/MandC/product/PC-Analog-PCI-3027.cfm>, Accessed: August 16 2008.
- [43] Amplicon. Adc product specification, the 9820 series. Available: <http://www.amplicon.co.uk/MandC/product/prod-spec.cfm?type=SPECIFICATIONS&did=2384>, Accessed: August 16 2008.
- [44] Amplicon. Adc product specification, the 9812 series. Available: <http://www.amplicon.co.uk/MandC/product/prod-spec.cfm?type=SPECIFICATIONS&did=2215>, Accessed: August 16 2008.
- [45] Amplicon. Adc product specification, the 2200 series. Available: <http://www.amplicon.co.uk/MandC/product/prod-spec.cfm?type=SPECIFICATIONS&did=2391>, Accessed: August 16 2008.
- [46] Amplicon. Pci express based adc products. Available: <http://www.amplicon.co.uk/MandC/product/PC-Analog-PCI-3030.cfm>, Accessed: August 16 2008.
- [47] Amplicon. Adc product specification, the 2000 series. Available: <http://www.amplicon.co.uk/MandC/product/prod-spec.cfm?type=SPECIFICATIONS&did=2339>, Accessed: August 16 2008.
- [48] Via. New mini-itx mainboard specification white paper. Available: http://www.via.com.tw/en/downloads/whitepapers/initiatives/spearhead/ini_mini-itx.pdf, November 2001.

- [49] Linux Devices. Linux support a requirement in “mini-itx 2.0”. Available: <http://www.linuxdevices.com/news/NS8641332057.html>, Accessed: August 17 2008.
- [50] Via. Via epia n-series nano-itx board. Available: http://www.via.com.tw/en/products/mainboards/motherboards.jsp?motherboard_id=221, March 2004.
- [51] Via. Via pico-itx form factor, “small is beautiful”. Available: http://www.via.com.tw/en/downloads/whitepapers/initiatives/spearhead/pico-itx_form_factor.pdf, April 2007.
- [52] Intel. *Intel® 64 and IA-32 Architectures, Software Developer’s Manual, Volume 1: Basic Architecture*, April 2008.
- [53] Analog Devices. Op275 datasheet - dual bipolar/jfet, audio operational amplifier. Available: http://www.analog.com/static/imported-files/data_sheets/OP275.pdf, February 2004. Rev. C.
- [54] The gEDA gschem information web-page. Available: <http://www.geda-seul.org/tools/gschem/index.html>, Accessed: August 20 2008.
- [55] The gEDA gschem information web-page. Available: <http://www.geda-seul.org/tools/gschem/index.html>, Accessed: August 20 2008.
- [56] St Microelectronics. Tsv911-tsv912-tsv914 datasheet - rail-to-rail input/output 8 mhz operational amplifiers. Available: <http://www.st.com/stonline/products/literature/ds/12584/tsv912.pdf>, February 2008. Rev. 3.
- [57] The gEDA pcb information web-page. Available: <http://www.geda-seul.org/tools/pcb/index.html>, Accessed: August 20 2008.
- [58] D.J. Delorie et al Harry Eaton, Dan McMahon. The official pcb homepage. Available: <http://pcb.sourceforge.net/>, Accessed: August 20 2008.
- [59] Bill Wilson. The gsch2pcb script faq. Available: <http://www.geda-seul.org/wiki/geda:faq-gsch2pcb>, Accessed: August 20 2008.
- [60] Barco Graphics. Gerber rs-274x format - user’s guide. Available: http://www.artwork.com/gerber/274x/rs274xrevd_e.pdf, 1998.
- [61] Linus Torvalds et al. The linux kernel homepage. Available: <http://www.kernel.org>, Accessed: August 24 2008.

BIBLIOGRAPHY

- [62] Linux from scratch homepage. Available: <http://www.linuxfromscratch.org>, Accessed: August 24 2008.
- [63] Damn small linux homepage. Available: <http://www.damnsmalllinux.org>, Accessed: August 24 2008.
- [64] Ubuntu homepage. Available: <http://www.ubuntu.com>, Accessed: August 24 2008.
- [65] Fedora homepage. Available: <http://www.fedoraproject.org>, Accessed: August 24 2008.
- [66] opensuse homepage. Available: <http://www.opensuse.org>, Accessed: August 24 2008.
- [67] Gnu is not unix operating system homepage. Available: <http://www.gnu.org>, Accessed: August 24 2008.
- [68] Debian gnu/linux homepage. Available: <http://www.debian.org>, Accessed: August 24 2008.
- [69] Source code management - revision control. Available: http://en.wikipedia.org/wiki/Source_Code_Management, Accessed: August 25 2008.
- [70] Debian gnu/linux cd/dvd images. Available: <http://www.debian.org/CD>, Accessed: August 25 2008.
- [71] User account at debuntu.org named chantra. How to: Installing debian etch from a running debian based system. Available: <http://www.debuntu.org/2006/05/14/51-how-to-installing-debian-etch-from-a-running-debian-based-system>, Accessed: August 25 2008.
- [72] Ultra Products. Usb 2.0 to ide/sata cable adapter ult40112. Available: http://www.ultraproducts.com/product_details.php?cPath=104&pPath=669&productID=669, Accessed: August 25 2008.
- [73] Iso/iec 8859. Available: http://en.wikipedia.org/wiki/ISO_8859, Accessed: August 25 2008.
- [74] Iso/iec 8859-1. Available: http://en.wikipedia.org/wiki/ISO/IEC_8859-1, Accessed: August 25 2008.

- [75] Utf-8. Available: <http://en.wikipedia.org/wiki/UTF-8>, Accessed: August 25 2008.
- [76] Iso/iec 10646 - universal character set. Available: http://en.wikipedia.org/wiki/ISO_10646, Accessed: August 25 2008.
- [77] Second extended file system. Available: <http://en.wikipedia.org/wiki/Ext2>, Accessed: August 25 2008.
- [78] Gnu grand unified bootloader. Available: http://en.wikipedia.org/wiki/GNU_GRUB, Accessed: August 25 2008.
- [79] The official gnu grub homepage. Available: <http://www.gnu.org/software/grub/>, Accessed: August 25 2008.
- [80] Non disclosure agreement. Available: http://en.wikipedia.org/wiki/Non-disclosure_agreement, Accessed: August 25 2008.
- [81] Adlink Technology Inc., 9F, No. 166, Jian Yi Road, Chungho City, Taipei, 235 Taiwan. *DAQ/PXI-201x/200x - 4-CH, Simultaneous, High Performance, Multi-Function Data Acquisition Card - User's Manual*, 2.00 edition, April 2006. Part No. 50-11020-1030.
- [82] Device manager for the linux 2.6 kernel series. Available: <http://en.wikipedia.org/wiki/Udev>, Accessed: August 25 2008.
- [83] Adlink Technology Inc., 9F, No. 166, Jian Yi Road, Chungho City, Taipei, 235 Taiwan. *D2K-DASK Data Acquisition Software Development Kit For DAQ-2000 Devices - User's Guide*, 1.73 edition, March 2006.
- [84] Adlink Technology Inc., 9F, No. 166, Jian Yi Road, Chungho City, Taipei, 235 Taiwan. *D2K-DASK Data Acquisition Software Development Kit for DAQ-2000 Devices - Function Reference Manual*, 2.00 edition, August 2007. Part No. 50-11225-2000.
- [85] The c programming language. Available: [http://en.wikipedia.org/wiki/C_\(programming_language\)](http://en.wikipedia.org/wiki/C_(programming_language)), Accessed: August 27 2008.
- [86] Brian W. Kernighan and Dennis M. Ritchie. *The C programming language*. Prentice Hall Software Series. Prentice-Hall, Upper Saddle River, NJ 07458, 2nd edition, 1988.
- [87] Matlab homepage. Available: <http://www.mathworks.com>, Accessed: August 27 2008.

BIBLIOGRAPHY

- [88] Matlab. Available: <http://en.wikipedia.org/wiki/MATLAB>, Accessed: August 27 2008.
- [89] Octave homepage. Available: <http://www.gnu.org/software/octave/>, Accessed: August 27 2008.
- [90] Gnu octave. Available: http://en.wikipedia.org/wiki/GNU_Octave, Accessed: August 27 2008.
- [91] Pdl homepage - the perl data language. Available: <http://pdl.perl.org/>, Accessed: August 27 2008.
- [92] Perl data language. Available: http://en.wikipedia.org/wiki/Perl_Data_Language, Accessed: August 27 2008.
- [93] Perl homepage. Available: <http://www.perl.org>, Accessed: August 27 2008.
- [94] T. Chen, Y. Wang, B. Fang, and J. Zheng. Detecting Lasting and Abrupt Bursts in Data Streams Using Two-Layered Wavelet Tree. *Proceedings of the Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services*, 2006.
- [95] Network time protocol. Available: <http://www.ntp.org>, Accessed: August 28 2008.
- [96] Network time protocol. Available: http://en.wikipedia.org/wiki/Network_Time_Protocol, Accessed: August 28 2008.
- [97] Doppler shift. Available: <http://imagine.gsfc.nasa.gov/YBA/M31-velocity/Doppler-shift-2.html>, Accessed: August 28 2008.
- [98] Doppler effect. Available: http://en.wikipedia.org/wiki/Doppler_effect, Accessed: August 28 2008.
- [99] H. Schau and A. Robinson. Passive source localization employing intersecting spherical surfaces from time-of-arrival differences. *Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing]*, *IEEE Transactions on*, 35(8):1223–1225, 1987.
- [100] BT Fang. Simple solutions for hyperbolic and related position fixes. *Aerospace and Electronic Systems, IEEE Transactions on*, 26(5):748–753, 1990.

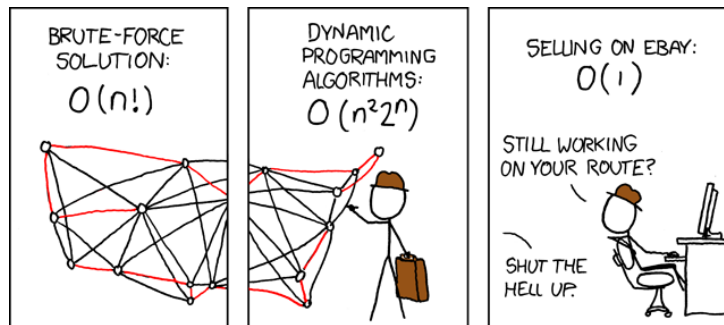
- [101] YT Chan and KC Ho. A simple and efficient estimator for hyperbolic location. *Signal Processing, IEEE Transactions on [see also Acoustics, Speech, and Signal Processing, IEEE Transactions on]*, 42(8):1905–1915, 1994.
- [102] Free Software Foundation. The free software definition. Available: <http://www.gnu.org/philosophy/free-sw.html>, Accessed: August 19 2008.
- [103] Gerber file. Available: http://en.wikipedia.org/wiki/Gerber_File, Accessed: July 24 2008.
- [104] Unix philosophy. Available: http://en.wikipedia.org/wiki/Unix_philosophy, Accessed: August 19 2008.
- [105] Travelling salesman problem. Available: en.wikipedia.org/wiki/Traveling_salesman_problem, Accessed: July 24 2008.
- [106] Ales V. Hvezda. geda/gaf symbol creation document. Available: <http://geda.seul.org/wiki/geda:scg>, July 6th, 2004.
- [107] m4 (computer language). Available: [http://en.wikipedia.org/wiki/M4_\(computer_language\)](http://en.wikipedia.org/wiki/M4_(computer_language)), Accessed: August 20 2008.
- [108] Harry Eaton. Pcb manual - an interactive printed circuit board layout system for x11. Available: <http://pcb.sourceforge.net/pcb-20080202/pcb.pdf>, February 2008.
- [109] Stephen Meier and Stuart D. Brorson. Footprint creation for the open-source layout program "pcb". Available: http://www.brorson.com/gEDA/land_patterns_20050129.pdf, January 2005.
- [110] ON Semiconductor. Datasheet - mc78m00, mc78m00a, ncv78m00 series - 500 ma positive voltage regulators. Available: <http://www.onsemi.com/pub/Collateral/MC78M00-D.PDF>, February 2008. Rev. 2.0.
- [111] make (software). Available: [http://en.wikipedia.org/wiki/Make_\(software\)](http://en.wikipedia.org/wiki/Make_(software)), Accessed: August 22 2008.
- [112] Paul D. Smith Richard M. Stallman, Roland McGrath. *GNU Make*, April 2006.
- [113] Scheme (programming language). Available: [http://en.wikipedia.org/wiki/Scheme_\(programming_language\)](http://en.wikipedia.org/wiki/Scheme_(programming_language)), Accessed: August 22 2008.

BIBLIOGRAPHY

- [114] Lisp (programming language). Available: http://en.wikipedia.org/wiki/Lisp_programming_language, Accessed: August 22 2008.
- [115] Chet Ramey and Brian Fox. *Bash reference manual*, 3.2 edition, September 2006.
- [116] Ed (text editor). Available: [http://en.wikipedia.org/wiki/Ed_\(text_editor\)](http://en.wikipedia.org/wiki/Ed_(text_editor)), Accessed: August 22 2008.
- [117] Patrick J. LoPresti. Ed, man! !man ed. Available: <http://www.gnu.org/fun/jokes/ed.msg.html>, July 1991.
- [118] Colin Kelley et al. Thomas Williams. *GNUplot - an interactive plotting program*, 4.2 edition, March 2007.
- [119] Perl. Available: <http://en.wikipedia.org/wiki/Perl>, Accessed: August 27 2008.
- [120] Gentoo linux homepage. Available: <http://www.gentoo.org>, Accessed: August 27 2008.
- [121] Gentoo linux. Available: http://en.wikipedia.org/wiki/Gentoo_Linux, Accessed: August 27 2008.

Appendix A

Open source software choices



Here some of the interesting aspects of solving relevant problems using open source software are described. Of course the aspects which have direct bearing on the project are described where appropriate.

By open source software we mean free software, as in speech, not as in beer [102]. See further discussion in section “Proprietary vs. open source solutions” in the Preface.

A.1 PCB production

Producing PCBs efficiently is a delicate process, unless you have equipment that only needs data files to produce PCBs. The equipment available to this project was a UV-machine, to create the wiring on a photo sensitive layer, and a bubble etch machine which etches all copper that was not exposed to the UV-light. It is a straightforward process to create two layer PCBs.

There are several open source packages that support PCB design and production, by production we mean being able to export the design in the RS-274-X file format, also known as Extended Gerber file format [60, 103], which can be used to fabricate the PCB. Those examined during the design

APPENDIX A. OPEN SOURCE SOFTWARE CHOICES

phase in this project were Kicad* and gEDA†. The findings will be described below.

The criteria looked for in this process were:

- Integration between the phases.
- Optimised rats nest generation from the netlist.
- Relatively easy multi-layer design.
- Auto-routing capability, would be nice.

A.1.1 The Kicad design tool

This was the first design tool that was tried, and showed great promise in the beginning. This project is an attempt at creating an integrated tool which contains schematics editor, netlist exporter and PCB creator. These tools are individual programs which agree on a shared protocol for communication, and thus you can select components in the schematic and have them selected in the PCB design program. Importantly it is also able to export the PCB design in the Gerber format.

As the first prototype was not satisfactory, seen on figures A.1 on the next page and A.2 on page 100, many changes were needed to improve it.

Given the many difficulties with using this program collection, another alternative was chosen for the next prototype. The major difficulties included:

- Many program-crashes.
- It is difficult to draw two sided PCBs.
- No way of creating keybindings.

Auto-routing was not found to be very efficient.

A.1.2 The gEDA design tool

The gEDA program collection, solves most problems one might have when creating PCBs. In this project, mostly the schematics and PCB design programs were used, `gschem` and `pcb` respectively.

*Official web page: http://www.lis.inpg.fr/realise_au_lis/kicad/ and official wiki web page: http://kicad.sourceforge.net/wiki/index.php/Main_Page

†<http://geda.seul.org/>

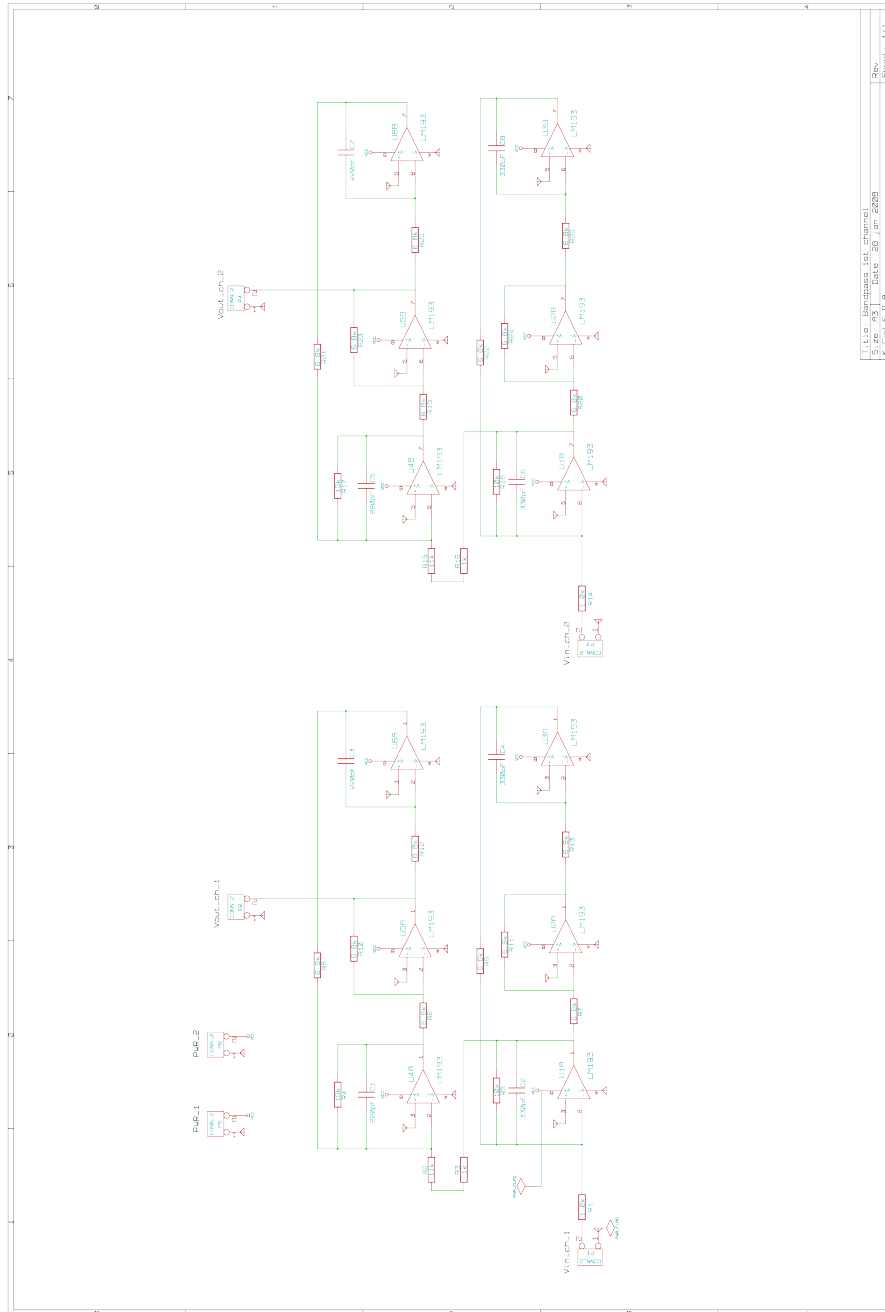


Figure A.1: First filter prototype, schematic created with Kicad

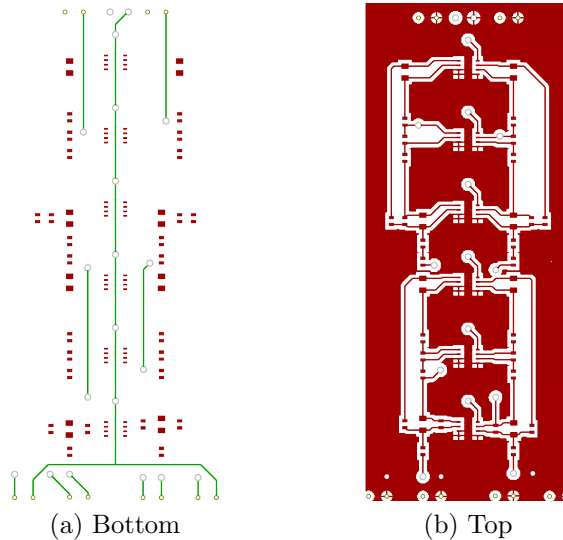


Figure A.2: First PCB filter prototype, design created with Kicad

The **gEDA** collection consists of several separately developed tools, and as such are not as well integrated as with Kicad. But on the other hand **gEDA** follows the Unix Philosophy better see [104] or according to Douglas McIlroy [‡]:

- Write programs that do one thing and do it well.
- Write programs to work together.
- Write programs to handle text streams, because that is a universal interface.

Such development rules allow us to use drop-in replacement if we so choose, also all intermediate data is text-based, and as such can be manipulated with the common tools available.

Beneficial as the above arguments may seem, they do result in loss of integration, meaning that there is no runtime communication between the schematics creator and the PCB creator, but in my opinion the benefits outweigh the drawbacks.

As the output from each tool is well defined, and that efforts have been made to support each others formats, the design process is surprisingly easy, once you get used to it.

[‡]<http://www.cs.dartmouth.edu/~doug/>

Though the PCB design part of **gEDA** includes both auto-placement and auto-routing, these were not found to be more efficient than doing it by hand, nor more optimised.

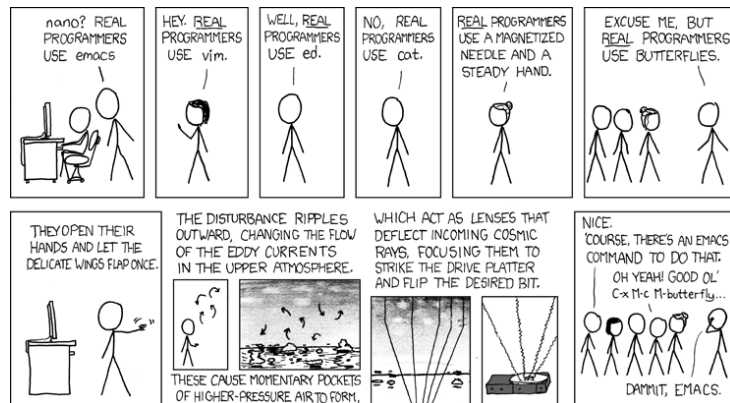
A.1.3 Concluding notes on design tools

There are many other open source design tools out there, but there was not time to test more of them. I do believe that Kicad and **gEDA** are some of the foremost tools available today, even though they both need more work.

Auto-routing is a very similar problem to the travelling-salesman problem [105], and as such is a NP-complete problem. So optimal solutions were not expected, somewhat better solutions than those found were however expected.

Appendix B

Source files for PCB production



For completeness, the text files created to complete the PCB design phase, are listed here with short description. References to interpreting the file formats are listed where relevant.

B.1 Symbol files for `gschem`

`gschem` is the schematics editor which has been included in the `gEDA` package, even though it is an independent project in its own right. Many of the `gEDA` programs use the same file format, e.g. the symbol file format used in this section is shared amongst some of the programs. Specification of this format can be found here [106].

There are several different ways to create symbols for `gschem`. Because the symbols I needed to create were fairly simple, I copied existing symbols and adapted them to my requirements.

B.1.1 Knowles SMD microphone

The SMD microphone had no existing symbol, so it needed to be created. I started with the regular microphone symbol, and modified it to fit the SMD microphone. This means some pins were added and correctly numbered and correct device name inserted.

The interpreted symbol is shown on figure B.1, and its listing can be seen below.

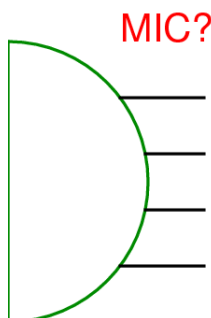


Figure B.1: gschem SMD microphone symbol

```

v 20050820 1
A 0 500 500 270 180 3 0 0 0 -1 -1
T 400 1200 5 10 0 0 0 0 1
device=SMD MICROPHONE
5 L 0 1000 0 0 3 0 0 0 -1 -1
T 400 1000 8 10 1 1 0 0 1
refdes=MIC?
T 400 1850 5 10 0 0 0 0 1
description=microphone
10 T 400 1650 5 10 0 0 0 0 1
numslots=0
T 400 1450 5 10 0 0 0 0 1
symversion=0.1
P 700 800 400 800 1 0 0
15 {
T 500 750 5 10 0 1 0 2 1
pinseq=1
T 350 800 5 10 0 1 0 8 1
pintype=pas
20 T 500 850 5 10 0 1 0 0 1
pinnumber=1
T 350 800 5 10 0 1 0 6 1
pinlabel=1
}
25 P 700 600 490 600 1 0 0

```

APPENDIX B. SOURCE FILES FOR PCB PRODUCTION

```
{
T 500 750 5 10 0 1 0 2 1
pinseq=2
T 350 800 5 10 0 1 0 8 1
30 pintype=pas
T 500 850 5 10 0 1 0 0 1
pinnumber=2
T 350 800 5 10 0 1 0 6 1
pinlabel=2
35 }
P 700 400 490 400 1 0 0
{
T 500 750 5 10 0 1 0 2 1
pinseq=3
40 T 350 800 5 10 0 1 0 8 1
pintype=pas
T 500 850 5 10 0 1 0 0 1
pinnumber=3
T 350 800 5 10 0 1 0 6 1
45 pinlabel=3
}
P 700 200 400 200 1 0 0
{
T 500 150 5 10 0 1 0 2 1
50 pinseq=4
T 350 200 5 10 0 1 0 8 1
pintype=pas
T 500 250 5 10 0 1 0 0 1
pinnumber=4
55 T 350 200 5 10 0 1 0 6 1
pinlabel=4
}
```

B.1.2 SMD voltage regulator

When no suitable symbol was found for the voltage regulator needed for the circuit, I decided to create my own. There may exist other symbols but as the process proved to be relatively easy when creating the SMD microphone symbol, this seemed like the shortest path to go.

The result can be seen in the listing below, figure B.2 on the facing page shows the resulting symbol.

```
v 20031231 1
T 400 600 9 8 1 0 0 0 1
IN
T 1148 600 9 8 1 0 0 0 1
5 OUT
```

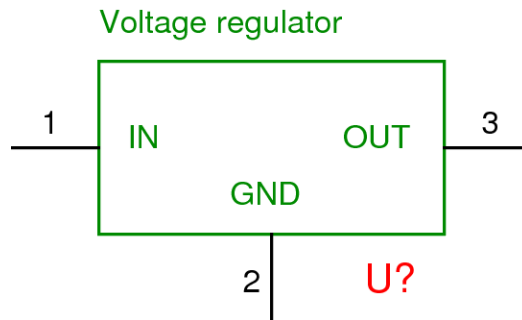


Figure B.2: gschem voltage regulator symbol

```

T 300 1000 9 8 1 0 0 0 1
Voltage regulator
B 300 300 1200 600 3 0 0 0 -1 -1 0 -1 -1 -1 -1 -1
T 1600 1300 5 10 0 0 0 0 1
10 device=7805
T 756 401 9 8 1 0 0 0 1
GND
P 300 600 0 600 1 0 1
{
15 T 100 650 5 8 1 1 0 0 1
pinnumber=1
T 100 650 5 8 0 0 0 0 1
pinseq=1
}
20 P 900 0 900 300 1 0 0
{
T 800 100 5 8 1 1 0 0 1
pinnumber=2
T 800 100 5 8 0 0 0 0 1
25 pinseq=2
}
P 1500 600 1800 600 1 0 1
{
T 1630 650 5 8 1 1 0 0 1
30 pinnumber=3
T 1630 650 5 8 0 0 0 0 1
pinseq=3
}
T 1400 100 8 10 1 1 0 6 1
35 refdes=U?
T 1600 1100 5 10 0 0 0 0 1
pins=3

```

B.2 Custom PCB footprints

The `pcb` program has a long history and was first released in 1990 for the Atari computer. The age and the fact that much backwards compatibility has been a goal, the footprint library is not the most user friendly place to be. That aside there are a host of different components, and the library is very flexible, i.e. it supports usage of the `m4` macro language [107] *. `m4` pre-processes the footprint files and replaces any macros found with a new string or, if the macro was procedural, with the result of the procedure. The procedural macros enable you to say things like:

```
CONNECTOR 5
```

or

```
CONNECTOR 5 5
```

which either gives you a 5 by 1 connector footprint or a 5 by 5 connector footprint respectively - this can be extended arbitrarily. Many other such functions exist.

As with the `gschem` program, there are several different ways to create new components, specifically there are the GUI approaches which uses a built-in component of the program to do so. As may be surmised from the above paragraphs, I chose to do this by hand, which was not that difficult, when you first get the hang of it. I used this guide [109] which also explains the file format.

B.2.1 Knowles SMD microphone

The `pcb` file format is somewhat easier to handle than `gschem`, and seems to make more sense, except for the pad description code. Anyway the SMD microphone consists of four connectors spread out evenly (see datasheet [30]). The result can be seen on figure B.3 on the next page, and is described by the listing below.

```

5 Element(0x00 "SPM smd microphone" "" "" 0 0 20 -100 0 100 0x00 ←
  )
  (
    Pad( 5 45 5 52 80 40 0 "Output" "1" 0x00000100)
    Pad( 5 192 5 199 80 40 0 "Ground" "2" 0x00000100)
    Pad(125 192 125 199 80 40 0 "Ground" "3" 0x00000100)
    Pad(125 45 125 52 80 40 0 "Power" "4" 0x00000100)
  )

```

*The usage of `m4` with `pcb` is documented in the `pcb` manual [108]

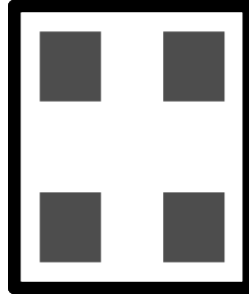


Figure B.3: Footprint for the Knowles microphone

```

10 ElementLine(-70 -35 -70 279 10)
    ElementLine(-70 279 195 279 10)
    ElementLine(195 279 195 -35 10)
    ElementLine(195 -35 -70 -35 10)
)

```

B.2.2 Voltage regulator

Here we document the footprint created for the positive voltage regulator, used with the filter. The ON Semiconductor component MC7809CDTG [110] was chosen for this task. The footprint was created based on the physical dimensions listed in the datasheet. The result can be seen on figure B.4, and is described by the listing shown below.

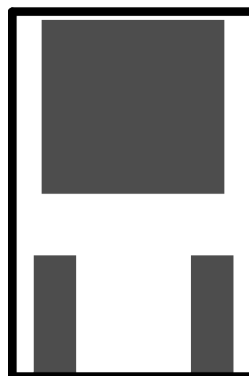


Figure B.4: Footprint for the Knowles microphone

```

Element(0x00 "SMD Voltage Regulator" "" "" 0 0 20 -100 0 100 0 ↵
x00)

```

APPENDIX B. SOURCE FILES FOR PCB PRODUCTION

```
(
  Pad(137 112 147 112 205 40 0 "Ground" "2" 0x00000100)
  Pad( 50 312  50 402  50 40 0 "V_in" "1" 0x00000100)
5  Pad(235 312 235 402  50 40 0 "V_out" "3" 0x00000100)

  ElementLine( 0 0 285 0 10)
  ElementLine(285 0 285 430 10)
  ElementLine(285 430 0 430 10)
10 ElementLine( 0 430 0 0 10)
)
```

B.3 Generating printouts for PCB

When creating a PCB in a bubble-etch bath, much information must be readily accessible, and thus must be exported from the computer. Specifically printouts of the following are needed:

- schematic of the circuit, to make soldering easier.
- component layout, also to make soldering easier.
- PCB solder side and mirrored PCB component side, printed on transparencies to be used when subjecting the photosensitive layer to UV-light.

It has been endeavoured to make this process as automated as possible, and to this end a *Makefile* has been created, which is interpreted and executed by the **make** program [111] (GNU make in our case [112]). The *Makefile* can be seen in the listing below.

```
# Which files to generate
TARGET      = bandpass
TARGETSCH   = $(TARGET)-sch.ps power_adaptation-sch.ps
5 TARGETPCB  = $(TARGET)-pcb.ps
TARGETLAYOUT = $(TARGET)-pcb.pdf
TARGETBOM   = $(TARGET)-pcb.bom
TARGETS     = $(TARGETSCH) $(TARGETPCB) $(TARGETLAYOUT) $ ←
              (TARGETBOM)

10 # gschem options
GSCHEM      = gschem
GSCHEMOPTS  = -q -p -s ../src/scripts/print.scm

# pcb options
```



```

15 PCB          = pcb
   PCB_OPTS    = -x ps --psfile /dev/stdout --mirror
   PCB_QUIRK    = tail -n +3

   # component layout options
20 PCB_LAYOUT_OPTS = -x eps --eps-file /dev/stdout
   PCB_LAYOUT_OPTS += --as-shown --only-visible
   PCB_LAYOUT_QUIRK = tail -n +3
   REMOVELAYERS    = ../src/scripts/remove_pcb_layers.sh
   NUMBER_OF_LAYERS = 4

25
   # eps fit options
   EPS_FIT         = epsffit
   EPS_FIT_OPTS    = -m -c -s
   EPS_FIT_A4      = 20 20 575 822      # A4 with 20 pt margin

30
   # epstopdf
   EPS_TO_PDF      = epstopdf

   # LP options
35 PRINTER = e2

all: $(TARGETS)
print: print_sch print_pcb print_layout print_bom

40
print_sch: $(TARGET_SCH)
    lp -d $(PRINTER) $(TARGET_SCH)

print_pcb: $(TARGET_PCB)
45    lp -d $(PRINTER) -P 1 $(TARGET_PCB)
    lp -d $(PRINTER) -P 2 $(TARGET_PCB)

print_layout: $(TARGET_LAYOUT)
50    lp -d $(PRINTER) $(TARGET_LAYOUT)

print_bom: $(TARGET_BOM)
    lp -d $(PRINTER) $(TARGET_BOM) $(TARGET_BOM:.bom=.xy)

# The schematic
55 %sch.ps: %.sch
    @echo "==== Exporting $< -> $@ ==="
    $(GSCHEM) $(GSCHEM_OPTS) -o $@ $<

# Mirrored component-side and normal solder-side
60 %pcb.ps: %.pcb
    @echo "==== Exporting $< -> $@ ==="
    $(PCB) $(PCB_OPTS) $< | $(PCB_QUIRK) \
        | psselect -q -p2-3 > $@

```

APPENDIX B. SOURCE FILES FOR PCB PRODUCTION

```
65 # Component layout
%-pcb.pdf: %-pcb.eps
    @echo "==== Exporting $< -> $@ ===="
    $(EPS_TO_PDF) --outfile=$@ $<

70 %-pcb.eps: %-pcb
    $(PCB) $(PCBLAYOUT_OPTS) $< \
        | $(PCBLAYOUT_QUIRK) > $@.tmp \
        && $(REMOVELAYERS) -n $(NUMBER_OF_LAYERS) $@.tmp \
        && $(EPS_FIT) $(EPS_FIT_OPTS) $(EPS_FIT_A4) $@.tmp $@ \
75    && $(RM) $@.tmp

%-pcb.bom: %-pcb
    $(PCB) -x bom --bomfile $@ --xyfile $(@:.bom=.xy) $<

80 clean:
    $(RM) $(TARGETS)
```

Running **make** on this *makefile* results in the listing in figure B.5 on the next page, the output has been rotated to improve readability.

To automatically generate these printouts, both **gschem** and **pcb** need to work as filters, and as might be surmised from the extent of the above file, they are not optimised for this task. Although they are designed to support it. Some of the more interesting workarounds employed will be described briefly in the sections below.

```

===== Exporting bandpass.sch -> bandpass-sch.ps =====
gschem -q -p -s ../src/scripts/print.scm -o bandpass-sch.ps bandpass.sch
===== Exporting power-adaptation.sch -> power-adaptation-sch.ps =====
gschem -q -p -s ../src/scripts/print.scm -o power-adaptation-sch.ps power-adaptation.sch
===== Exporting bandpass.pcb -> bandpass-pcb.ps =====
pcb -x ps --psfile /dev/stdout --mirror bandpass.pcb | tail -n +3 | psselect -q -p2-3 > bandpass-pcb.ps
pcb -x eps --epsfile /dev/stdout --as-shown --only-visible bandpass.pcb | tail -n +3 > bandpass-pcb.eps.tmp \
    && ../src/scripts/remove-pcb-layers.sh -n 4 bandpass-pcb.eps.tmp \
    && epsffit -m -c -s 20 20 575 822 bandpass-pcb.eps.tmp bandpass-pcb.eps \
    && rm -f bandpass-pcb.eps.tmp
===== Exporting bandpass-pcb.eps -> bandpass-pcb.pdf =====
epstopdf --outfile=bandpass-pcb.pdf bandpass-pcb.eps
pcb -x bom --bomfile bandpass-pcb.bom --xyfile bandpass-pcb.xy bandpass.pcb
Looking for default-font in /usr/share/pcb
Found default-font in /usr/share/pcb
rm bandpass-pcb.eps

```

Figure B.5: Output from running make in *filter/* directory

B.3.1 Printout quirks with gschem

`gschem` does support batch processing, but only by supplying an extra Scheme [113] script, Scheme is a Lisp [114] dialect. As can be seen in the *Makefile* listing above, a file named *print.scm* is supplied to the `gschem` program. This file contains the following, see `gschem(1)` for more on the options supplied to `gschem` in the *Makefile*.

```
5 ; You need call this after you call any rc file function
   (gschem-use-rc-values)

   ; Filename is specified on the command line
   (gschem-postscript "dummyfilename")

   (gschem-exit)
```

B.3.2 Printout quirks with pcb

Although `pcb` was designed to support running it as a filter, its insistence on printing approximately fourteen pages on each run is rather annoying. And thus workarounds were needed for many aspects of the printout generation. These are rather serious faults in the program, and should be improved, given the time I will do so myself.

No separation of *stdout* and *stdin*

As `pcb` prints status messages as well as the printout onto the *stdout* stream, some of the output needs to be ignored, thus the **PCB_QUIRK** which removes status messages from the output.

Copious output

There is no way to select only certain printouts, so we select them ourselves with `psselect`, we only need page two and three.

Error in component printout

The needed layers are not readily visible in the standard component printout, and thus some layer-colours must be changed. No matter how much tweaking was done, `pcb` refused to change the colours of the printed layers of the component printout. This is probably a bug. The workaround here was to create Bash script [115] which in turn generates an Ed script [116,117]. Not the most intuitive solution, but not boring either. The

script `remove_pcb_layers.sh` is listed below, and the resulting component layout is seen on figure B.6 on the following page.

```
#!/bin/bash

# Author: Thor Andreassen <ta@imada.sdu.dk>
# Created: 2008-03-09

5 # Description: This script removes layers from a pcb
# exported as eps by geda pcb. This was written because geda
# pcb does not seem to be consistent in its outputs.
#
10 # Basically this script works by removing (turning white)
# those layers we are not interested in, e.g. solder tracks.
# The script depends on the ED editor.
#
# Currently we can only remove layers from top down, this is
15 # sufficient for my needs, but can easily be adapted to be
# more flexible. This functionality, IMHO, should be in geda
# pcb and I will submit a patch if time permits.

PROGNAME=$(basename $0)
20 ED_SCRIPT_REPLACE='/setrgbcolor\ns/[0-9.]* /1 /g\n'
ED_SCRIPT_WRITE='wq\n'
LAYERS=1

if [ "$1" = "-n" ]; then
25 shift; LAYERS=$1; shift
fi

if [ -z "$1" ]; then
30 echo "usage: $PROGNAME [-n LAYERS] FILE"
fi

# We first generate the ed script, and then pipe it over to ed
{
35 for i in $(seq $LAYERS); do
echo -ne "$ED_SCRIPT_REPLACE"
done;
echo -ne "$ED_SCRIPT_WRITE"
} | ed -s "$1" | tail -n +$(($LAYERS + 1))
```

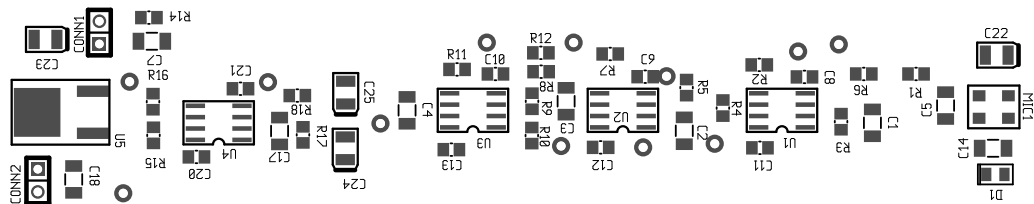
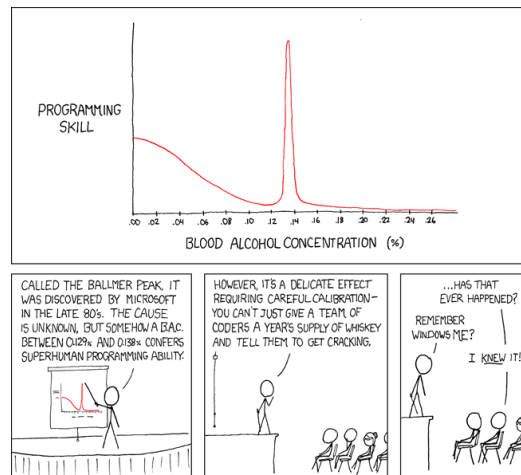


Figure B.6: Component layout of the PCB

Appendix C

A/D-board source code



So as not to clutter the chapters too much, only the most relevant A/D-board source code listings are included directly in the chapter, the rest is listed here.

C.1 Sampling programs

To maximize reuse of code a small support library has been developed, this library is listed in the next subsection. After that the sampling programs are listed, and an attempt at creating an oscilloscope like sampling and plotting. See header of listing for description.

It should be noted that some of the output from these programs is intended to be interpreted and rendered by `gnuplot` [118], hence the `plot '-'`, terminating the sequence with a single `e`.

APPENDIX C. A/D-BOARD SOURCE CODE

Also it should be noted that the double-buffered sampling program has yet to be integrated with the newly created support library.

C.1.1 Support library

Common inclusions: *common.h*

```
5  /*
   * Author: Thor Andreassen <ta@imada.sdu.dk>
   *
   * Description: Common header files and defines.
   */

   #ifndef __COMMON_H__
   #define __COMMON_H__

10  #include <error.h>
   #include <stdlib.h>
   #include <stdio.h>

   typedef enum { false, true } bool;

15  #endif /* ifndef __COMMON_H__ */
```

Initialise and release DAQ device: *daq2010_utils.h* and *daq2010_utils.c*

```
5  /*
   * Author: Thor Andreassen <ta@imada.sdu.dk>
   *
   * Description: DAQ2010 related functions.
   */

   #ifndef __DAQ2010_UTILS_H__
   #define __DAQ2010_UTILS_H__

10  #include <my_utils/common.h>
   #include <my_utils/signals.h>
   #include <daq2010/d2kdask.h>

   I16 daq_card;

15  void init_dac(void * buffer, U32 write_count, U16 *buf_id);
   void init_single_shot_adc(void);
   void init_buffered_dac(void);
   void init_double_buffered_adc(void);
```



```

20 void init_ao_buffer(void * buffer, U32 write_count, U16 * ↵
    buf_id);
void init_ai_double_buffer(U32 buffer_size, I16 *buf1, I16 * ↵
    buf2, U16 *buf_id1, U16 *buf_id2);

void dac_write(U16 ao_buf_id, U32 write_count, U32 ↵
    sample_update_rate);

25 void register_daq2010(void);
void release_daq2010(void);

#endif /* ifndef __DAQ2010_UTILS_H__ */

```

```

/*
 * Author: Thor Andreassen <ta@imada.sdu.dk>
 *
 * Description: Functions to register, release and configure ↵
    DAQ cards.
5 */

#include <my_utils/daq2010_utils.h>

/* Configure - start */
10 void init_dac(void * buffer, U32 write_count, U16 *buf_id) {
    init_buffered_dac();
    init_sigint(daq_card);
    init_ao_buffer(buffer, write_count, buf_id);
}

15 void init_single_shot_adc(void) {
    if(D2K_AI_CH_Config(daq_card, 1, AD_B_10_V) != NoError)
        error(-1, 0, "Error while configuring single shot ADC");
}

20 void init_double_buffered_adc(void) {
    if(D2K_AI_CH_Config(daq_card, -1, AD_B_10_V) != 0
        || D2K_AI_Config(daq_card, 0, 0, 0, 0, 0, 1) != 0)
        error(-1, 0, "Error while configuring channels");
25 }

void init_buffered_dac(void) {
    if(D2K_AO_CH_Config(daq_card, 1, DAQ2K_DA_BiPolar, ↵
        DAQ2K_DA_Int_REF, 10.0) != NoError
        || D2K_AO_Config(daq_card, DAQ2K_DA_WRSRC_Int, ↵
            DAQ2K_DA_TRGSRC_SOFT | DAQ2K_DA_TRGMOD_POST, 1, 0, ↵
            0, false) != NoError)
30 error(-1, 0, "Error while configuring AO channel");
}

```

APPENDIX C. A/D-BOARD SOURCE CODE

```
void init_ao_buffer(void *buffer, U32 write_count, U16 *buf_id ↵
) {
    if(D2K_AO_ContBufferSetup(daq_card, buffer, write_count, ↵
        buf_id) != NoError)
35     error(-1, 0, "Error while enabling AO buffered mode");
}

void init_ai_double_buffer(U32 buffer_size, I16 *buf1, I16 * ↵
    buf2, U16 *buf_id1, U16 *buf_id2) {
    if(D2K_AI_AsyncDblBufferMode (daq_card, TRUE) != NoError
40     || D2K_AI_ContBufferSetup(daq_card, buf1, buffer_size, ↵
        buf_id1) != NoError
        || D2K_AI_ContBufferSetup(daq_card, buf2, buffer_size, ↵
            buf_id2) != NoError)
        error(-1, 0, "Error while enabling double-buffered mode");
}
/* Configure - end */
45
/* Actions - start */
void dac_write(U16 ao_buf_id, U32 write_count, U32 ↵
    sample_update_rate) {
    U32 ao_count;

50     if(D2K_AO_ContWriteChannel(daq_card, 1, ao_buf_id, ↵
        write_count, 1,
        sample_update_rate, 0, ASYNCHOP) != NoError)
        error(-1, 0, "Error while DAC write");
    D2K_AO_AsyncClear(daq_card, &ao_count, DAQ2K_DA_TerminateIC) ↵
        ;
}
55 /* Actions - end */

/* Register and Release */
void register_daq2010(void) {
#ifdef DEBUG
60     fprintf(stderr, "Registering card\n");
#endif

    if( (daq_card = D2K_Register_Card(DAQ_2010, 0)) < 0)
        error(-1, 0, "Registering card error");
65 }

void release_daq2010(void) {
#ifdef DEBUG
70     fprintf(stderr, "Releasing card and terminating\n");
#endif

    if(D2K_Release_Card(daq_card) < 0)
        error(-1, 0, "Release error");
}
```

```
}

```

Signal handling: *signal.h* and *signal.c*

```

/*
 * Author: Thor Andreassen <ta@imada.sdu.dk>
 *
 * Description: Signal related functions.
5 */

#ifndef __SIGNALS_H__
#define __SIGNALS_H__

10 #include <signal.h>
#include <daq2010/d2kdask.h>
#include <my_utils/common.h>

void init_sigint(I16 new_card);
15 void trap_sigint(int signum);

#endif /* ifndef __SIGNALS_H__ */

```

```

/*
 * Author: Thor Andreassen <ta@imada.sdu.dk>
 *
 * Description: Initialise signals to the defined trap.
5 */

#include <my_utils/signals.h>

10 I16 card;

void init_sigint(I16 new_card) {
    if( signal(SIGINT, &trap_sigint) == SIG_ERR)
        error(-1, 0, "Unable to register signal handler");
15     card = new_card;
}

void trap_sigint(int signum) {
20     fprintf(stderr, "Received signal: %d\n", signum);
    fprintf(stderr, "Releasing card and terminating\n");

    if(D2K_Release_Card(card) < 0)
25     error(-1, 0, "Release error");
}

```

APPENDIX C. A/D-BOARD SOURCE CODE

```
    exit(0);  
}
```

Timer configuration: *timer.h* and *timer.c*

```
/*  
 * Author:  Thor Andreassen <ta@imada.sdu.dk>  
 *  
 * Description: Timer related functions, created an easy ↵  
               interface to nano  
5  *               sleep functionality.  
 */  
  
#ifndef __TIMER_H__  
#define __TIMER_H__  
10 #include <time.h>  
  
#define ts2dbl(timeValStruct) \  
    ((double) timeValStruct.tv_sec + timeValStruct.tv_nsec * 1E ↵  
    -9)  
15  
void    time_stamp(void);  
double  time_elapsed(void);  
void    my_sleep(double secs);    /* with nanosecond precision ↵  
    */  
20 #endif /* ifndef __TIMER_H__ */
```

```
/*  
 * Author:  Thor Andreassen <ta@imada.sdu.dk>  
 *  
 * Description: Timer related functions, created an easy ↵  
               interface to nano  
5  *               sleep functionality.  
 */  
  
#include <my_utils/timer.h>  
  
10 double time_stamp_dbl;  
struct timespec ts;  
  
void time_stamp(void) {  
    clock_gettime(CLOCK_REALTIME, &ts);  
15    time_stamp_dbl = ts2dbl(ts);  
}
```

```

double time_elapsed(void) {
    clock_gettime(CLOCK_REALTIME, &ts);
20   return(ts2dbl(ts) - time_stamp_dbl);
}

void my_sleep(double secs) {
    ts.tv_sec  = (long) secs;
25   ts.tv_nsec = (long) 1E9 * (secs - ts.tv_sec);

    nanosleep(&ts, NULL);
}

```

C.1.2 Software triggered sampling

```

/*
 * Author:  Thor Andreassen <ta@imada.sdu.dk>
 *
 * Description: Software triggered sampling.
5  */

#include <stdio.h>
#include <my_utils/daq2010_utils.h>
#include <my_utils/timer.h>
10

int main( void )
{
    F64 sample1, sample2, sample3;
    int i = 0;
15

    init_single_shot_adc();

    printf("set xrange [0:1000]\n");
    printf("set yrange [-10:10]\n");
20

    while(1) {
        puts("plot '-' with lines");
        while(i++ < 1000) {
            D2K_AI_VReadChannel(0, 1, &sample1);
            D2K_AI_VReadChannel(0, 2, &sample2);
            D2K_AI_VReadChannel(0, 3, &sample3);
25             printf("%10f %10f %10f\n", sample1, sample2, sample3);
        }
        puts("e");
        fflush(stdout);
30         i = 0;
    }
}

```

APPENDIX C. A/D-BOARD SOURCE CODE

```
35     release_adc();  
  
     return(0);  
}
```

C.1.3 Double buffered sampling

```
/*  
 * Author: Thor Andreassen <ta@imada.sdu.dk>  
 *  
 * Description: Double-buffered sampling  
5 */  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <error.h>  
10 #include <signal.h>          /* sigint */  
#include <daq2010/d2kdask.h>  
#include <my_utils/timer.h>  
  
/*#define ALBUFFER_SIZE 1048576 */  
15 #define ALBUFFER_SIZE 4096  
#define SAMPLE_RATE 1e5  
#define NUMBER_OF_SAMPLES_PER_SIGNAL 1000  
#define SAMPLE_RATE_MAGIC_NUMBER (8e7 / SAMPLE_RATE)  
#define SIGNAL_THRESHOLD 50  
20  
  
enum { false, true } bool;  
  
char debug = true;  
I16 card;  
25  
  
void trap_sigint(int signum);  
  
int main( void )  
{  
30     I16 ai_buf_first[ALBUFFER_SIZE], ai_buf_second[↵  
        ALBUFFER_SIZE];  
     I16 * ai_buf_active = NULL;  
     U16 chans[4] = {0,1,2,3};  
     U16 buf_id;  
     BOOLEAN is_half_ready, is_stopped;  
35     U32 ai_iterator;  
     int i;  
     BOOLEAN signal_print = false;  
  
     if( signal(SIGINT, &trap_sigint) == SIG_ERR)
```

```

40     error(-1, 0, "Unable to register signal handler");

    if(debug) fprintf(stderr, "Registering card\n");

    if( (card = D2K_Register_Card(DAQ_2010, 0)) < 0)
45         error(-1, 0, "Registering card error");

    if(D2K_AI_CH_Config(card, -1, AD_B_10_V) != 0
        || D2K_AI_Config(card, 0, 0, 0, 0, 0, 1) != 0)
        error(-1, 0, "Error while configuring channels");

50     if(D2K_AI_AsyncDblBufferMode (card, TRUE) != 0
        || D2K_AI_ContBufferSetup(card, ai_buf_first,  ←
            AIBUFFER.SIZE, &buf_id) != 0
        || D2K_AI_ContBufferSetup(card, ai_buf_second,  ←
            AIBUFFER.SIZE, &buf_id) != 0)
        error(-1, 0, "Error while enabling double-buffered mode");

55     time_stamp();
    if(D2K_AI_ContReadMultiChannels(card, 4, chans, 0,  ←
        AIBUFFER.SIZE/4, SAMPLERATEMAGICNUMBER, 0, ASYNCHOP  ←
        ) != 0)
        error(-1, 0, "Error while calling ContReadMultiChannels");

60     while(1) {
        ai_iterator = 0;

        do {
            if(ai_buf_active != NULL && ai_iterator + 3 <  ←
                AIBUFFER.SIZE) {
65                ai_buf_active[ai_iterator + 1] >>= 2;    /* Remove DI  ←
                    values */
                ai_buf_active[ai_iterator + 2] >>= 2;    /* Remove DI  ←
                    values */
                ai_buf_active[ai_iterator + 3] >>= 2;    /* Remove DI  ←
                    values */

                if(!signal_print
70                    && (abs(ai_buf_active[ai_iterator + 1]) >  ←
                        SIGNAL_THRESHOLD
                    || abs(ai_buf_active[ai_iterator + 2]) >  ←
                        SIGNAL_THRESHOLD
                    || abs(ai_buf_active[ai_iterator + 3]) >  ←
                        SIGNAL_THRESHOLD)) {
                    signal_print = true;
                    i = 0;
75                }
            }
        } while(1);
    }

```

APPENDIX C. A/D-BOARD SOURCE CODE

```
        if( signal_print && i < NUMBER_OF_SAMPLES_PER_SIGNAL ) {  
            {  
                printf("%d %d %d\n", ai_buf_active[ai_iterator + 1],  
                    ai_buf_active[ai_iterator + 2], ai_buf_active[  
                    ai_iterator + 3]);  
            }  
80         if(++i == NUMBER_OF_SAMPLES_PER_SIGNAL) {  
            /* signal separator */  
            puts("#####");  
            signal_print = false;  
        }  
85         fflush(stdout);  
    }  
  
    ai_iterator += 4;  
    }  
90    D2K_AI_AsyncDblBufferHalfReady(card, &is_half_ready, &  
        is_stopped);  
    } while ( !is_half_ready );  
  
    /* fprintf(stderr, "%d\n", i); */  
    /* fwrite(ai_buf_active, sizeof(I16), ALBUFFER_SIZE,   
        stdout); */  
95    ai_buf_active = (ai_buf_active == ai_buf_first) ?  
        ai_buf_second : ai_buf_first;  
    }  
  
    if(debug)  
        fprintf(stderr, "Releasing card and terminating\n");  
100    if(D2K_Release_Card(card) < 0)  
        error(-1, 0, "Release error");  
  
    return(0);  
    }  
105  
  
void trap_sigint(int signum) {  
    if(debug) {  
        fprintf(stderr, "Received signal: %d\n", signum);  
        fprintf(stderr, "Releasing card and terminating\n");  
110    }  
    if(D2K_Release_Card(card) < 0)  
        error(-1, 0, "Release error");  
    exit(0);  
}
```

C.1.4 Simple oscilloscope implementation

```

#include <stdio.h>
#include <stdlib.h>
#include <error.h>
#include <signal.h>      /* sigint */
5 #include <time.h>      /* clock_gettime */
#include <daq2010/d2kdask.h>
#include <my_utils/timer.h>

#define __USE_XOPEN
10 #include <math.h>      /* abs */

enum { false, true } bool;

char debug = true;
15 I16 card;

void trap_sigint(int signum);

#define PLOT_LENGTH      2E3
20 #define SIGNAL_THRESHOLD 300
#define ALBUFFER_SIZE    262144
#define FOURTEEN_BIT_MIN (-8192)
#define FOURTEEN_BIT_MAX 8191

25 int main( void )
{
    int ai_iterator;
    short min = FOURTEEN_BIT_MAX;
    short max = FOURTEEN_BIT_MIN;
30    short middle = 0;
    short limit_y1 = FOURTEEN_BIT_MIN;
    short limit_y2 = FOURTEEN_BIT_MAX;

    /* double-buffered AI variables - start */
35    I16 ai_buf_first[ALBUFFER_SIZE], ai_buf_second[ ←
        ALBUFFER_SIZE];
    I16 * ai_buf_active = NULL;
    U16 chans[4] = {0,1,2,3};
    BOOLEAN is_ai_half_ready, is_ai_stopped, start_plotting;
    U16 buf_id;
40    /* double-buffered AI variables - end */

    if( signal(SIGINT, &trap_sigint) == SIG_ERR)
        error(-1, 0, "Unable to register signal handler");

    if(debug) fprintf(stderr, "Registering card\n");
45    if( (card = D2K_Register_Card(DAQ_2010, 0)) < 0)
        error(-1, 0, "Registering card error");

```

APPENDIX C. A/D-BOARD SOURCE CODE

```
50  /* AI config - start */
    if(D2K_AI_CH.Config(card, -1, AD_B_10_V) != 0
       || D2K_AI.Config(card, 0, 0, 0, 0, 0, 1) != 0)
        error(-1, 0, "Error while AI configuring channels");

55  if(D2K_AI_AsyncDblBufferMode (card, TRUE) != NoError
     || D2K_AI_ContBufferSetup(card, ai_buf_first,  ←
        AI_BUFFER_SIZE, &buf_id) != NoError
     || D2K_AI_ContBufferSetup(card, ai_buf_second,  ←
        AI_BUFFER_SIZE, &buf_id) != NoError)
        error(-1, 0, "Error while enabling AI double-buffered mode ←
            ");
    /* AI config - end */

60  if(D2K_AI_ContReadMultiChannels(card, 4, chans, 0,  ←
     AI_BUFFER_SIZE/4, 80, 0, ASYNCHOP) != 0)
        error(-1, 0, "Error while enabling AI acquisition");

    while(1) {
65      /* next plot preamble */
        if(ai_buf_active != NULL) {

            if( min < limit_y1 + SIGNAL_THRESHOLD )
                limit_y1 -= SIGNAL_THRESHOLD;

70          if( min > limit_y1 + 2 * SIGNAL_THRESHOLD )
                limit_y1 += SIGNAL_THRESHOLD;

            if( max > limit_y2 - SIGNAL_THRESHOLD )
                limit_y2 += SIGNAL_THRESHOLD;

75          if( max < limit_y2 - 2 * SIGNAL_THRESHOLD )
                limit_y2 -= SIGNAL_THRESHOLD;

            printf("set xrange [ 0 : %d ]\n", (int) PLOT_LENGTH -  ←
                1000);
            printf("set yrange [ %d : %d ]\n", limit_y1, limit_y2);
            printf("plot '-' with lines, '-' with lines, '-' with  ←
                lines\n");
        }

80      /* initialise loop variables */
        ai_iterator = 1;

        middle = (max + min) / 2;
        min = FOURTEEN_BIT_MAX;
        max = FOURTEEN_BIT_MIN;

90
```

```

start_plotting = false;

is_ai_half_ready = false;
95
while(ai_iterator < PLOTLENGTH * 4 && !is_ai_half_ready) ←
{
    if(ai_buf_active != NULL) {
        printf("%d\n", ai_buf_active[ai_iterator] >> 2);
        /* printf("%d\n", ai_buf_active[ai_iterator+1] >> 2); ←
        */
100    /* printf("%d\n", ai_buf_active[ai_iterator+2] >> 2); ←
        */

        if( min > ai_buf_active[ai_iterator] >> 2 )
            min = ai_buf_active[ai_iterator] >> 2;

105        if( max < ai_buf_active[ai_iterator] >> 2 )
            max = ai_buf_active[ai_iterator] >> 2;

        ai_iterator += 4;
    }
110    D2K_AI_AsyncDblBufferHalfReady(card, &is_ai_half_ready, ←
        &is_ai_stopped);
}

if(ai_buf_active != NULL)
    printf("e\n");
115
ai_iterator = 2;

while(ai_iterator < PLOTLENGTH * 4 && !is_ai_half_ready) ←
{
    if(ai_buf_active != NULL) {
120        printf("%d\n", ai_buf_active[ai_iterator] >> 2);

        if( min > ai_buf_active[ai_iterator] >> 2 )
            min = ai_buf_active[ai_iterator] >> 2;

125        if( max < ai_buf_active[ai_iterator] >> 2 )
            max = ai_buf_active[ai_iterator] >> 2;

        ai_iterator += 4;
    }
130    D2K_AI_AsyncDblBufferHalfReady(card, &is_ai_half_ready, ←
        &is_ai_stopped);
}

if(ai_buf_active != NULL)
    printf("e\n");

```

APPENDIX C. A/D-BOARD SOURCE CODE

```
135     ai_iterator = 3;

    while(ai_iterator < PLOTLENGTH * 4 && !is_ai_half_ready)  ↵
    {
        if(ai_buf_active != NULL ) {
140             printf("%d\n", ai_buf_active[ai_iterator] >> 2);

            if( min > ai_buf_active[ai_iterator] >> 2 )
                min = ai_buf_active[ai_iterator] >> 2;

145             if( max < ai_buf_active[ai_iterator] >> 2 )
                max = ai_buf_active[ai_iterator] >> 2;

            ai_iterator += 4;
        }
150     D2K_AI_AsyncDblBufferHalfReady(card, &is_ai_half_ready,  ↵
        &is_ai_stopped);
    }

    if(ai_buf_active != NULL)
        printf("e\n");
155     ai_buf_active = (ai_buf_active == ai_buf_first) ?  ↵
        ai_buf_second : ai_buf_first;
    }

    if(debug)
160         fprintf(stderr, "Releasing card and terminating\n");
    if(D2K_Release_Card(card) < 0)
        error(-1, 0, "Release error");

    return(0);
165 }

void trap_sigint(int signum) {
    if(debug) {
170         fprintf(stderr, "Received signal: %d\n", signum);
        fprintf(stderr, "Releasing card and terminating\n");
    }

    if(D2K_Release_Card(card) < 0)
        error(-1, 0, "Release error");
175     exit(0);
}
```

C.1.5 Makefiles

There have been written a couple of makefiles to compile the library and the sampling programs. It may be noticed that there are an overwhelming amount of **C_FLAGS**, these are a cross-section of what is used to compile the `perl` [93, 119] code base, the *gentoo* Linux distribution [120, 121] and personal taste. This should catch most compile- and some aesthetic errors.

Support library

This makefile compiles all c-files in the directory into *libmyutils.so* shared object file and, if requested, copies it to */usr/local/lib* and runs `ldconfig`. Below is the makefile file listing and below that the output listing with results from running `make install` in the *utils/* directory.

Makefile Note that we define **_POSIX_C_SOURCE=200112L** which requires the compiled code to be *POSIX.1* compliant.

```

# Author:  Thor  Andreassen  <ta@imada.sdu.dk>
#
# Description:  Definitions of the rules to compile and link
#               the program. We also include rules to create
5 #               tags.

MY_LIBRARY = libmyutils.so
DESTINATION = /usr/local/lib

10 CC          = gcc
CFLAGS       = -Wall -g -ansi -pedantic -pedantic-errors \
               -Wextra -Wshadow -Wpointer-arith  ←
               \
               -Wbad-function-cast -Winline -Wcast- ←
               qual \
               -Wcast-align -Wwrite-strings  ←
               \
15               -Wstrict-prototypes -Wmissing-prototypes \
               -Wnested-externs -D_POSIX_C_SOURCE=200112L \
               -I../include
LDFLAGS      = -lm -lrt -lpthread -lpci_dask2k

20 C_FILES    = $(wildcard *.c)
OBJS         = $(C_FILES:.c=.o)

.PHONY: all install clean

25 all: $(MY_LIBRARY)
```

APPENDIX C. A/D-BOARD SOURCE CODE

```
$(MY_LIBRARY): $(OBJS)
    $(CC) -shared -o $(MY_LIBRARY) $(OBJS)

30 install: $(MY_LIBRARY)
    sudo install --owner=root --group=staff --mode=0755 \
        $(MY_LIBRARY) $(DESTINATION)
    sudo ldconfig

35 clean:
    $(RM) $(OBJS) $(MY_LIBRARY)
```

Resulting output The output that results from running `make` on the above makefile.

```
gcc -Wall -g -ansi -pedantic -pedantic-errors -Wextra -↵
Wshadow -Wpointer-arith -Wbad-function-cast -Winline -↵
Wcast-qual -Wcast-align -Wwrite-strings -Wstrict-↵
prototypes -Wmissing-prototypes -Wnested-externs -↵
D_POSIX_C_SOURCE=200112L -I../include -c -o ↵
daq2010_utils.o daq2010_utils.c
gcc -Wall -g -ansi -pedantic -pedantic-errors -Wextra -↵
Wshadow -Wpointer-arith -Wbad-function-cast -Winline -↵
Wcast-qual -Wcast-align -Wwrite-strings -Wstrict-↵
prototypes -Wmissing-prototypes -Wnested-externs -↵
D_POSIX_C_SOURCE=200112L -I../include -c -o signals.o ↵
signals.c
gcc -Wall -g -ansi -pedantic -pedantic-errors -Wextra -↵
Wshadow -Wpointer-arith -Wbad-function-cast -Winline -↵
Wcast-qual -Wcast-align -Wwrite-strings -Wstrict-↵
prototypes -Wmissing-prototypes -Wnested-externs -↵
D_POSIX_C_SOURCE=200112L -I../include -c -o timer.o ↵
timer.c
gcc -shared -o libmyutils.so daq2010_utils.o signals.o timer.o ↵
o
5 sudo install --owner=root --group=staff --mode=0755 libmyutils ↵
.so /usr/local/lib
sudo ldconfig
```

Sampling programs

The sampling programs share one makefile, and have an individual makefile as well. As it is now this separation is not needed, but makes the compile process more flexible if individual compile options need to be added later. Below is listed the common and the individual makefiles, and below that we list the resulting output from running `make` in one of the sampling program directories.

Common makefile The makefile common to all sampling programs, and the oscilloscope program.

```

CC      = gcc
CFLAGS  = -Wall -g -ansi -pedantic -pedantic-errors \
          -Wextra -Wshadow -Wpointer-arith ←
          \
          -Wbad-function-cast -Winline -Wcast-qual ←
          \
          -Wcast-align -Wwrite-strings ←
          \
          -Wstrict-prototypes -Wmissing-prototypes \
          -Wnested-externs -D_POSIX_C_SOURCE=200112L \
          -I../.. /include
LDLAGS  = -lm -lrt -lpthread -lpci_dask2k -lmyutils

```

Individual makefile This makefile is currently used by all sampling programs, this may change in the future.

```

# Author:  Thor Andreassen <ta@imada.sdu.dk>
#
# Description: Compile all c files with the options found in
#              common.mk.
5
include ../common.mk

# File definitions
C_FILES = $(wildcard *.c)
10 OBJS   = $(C_FILES:.c=.o)

.PHONY:

sample: $(OBJS)
15
clean:
    $(RM) $(OBJS)

```

Resulting output Output when running make in a sample program directory, this includes variables from both the above listed makefiles.

```

gcc -Wall -g -ansi -pedantic -pedantic-errors -Wextra - ←
Wshadow -Wpointer-arith -Wbad-function-cast -Winline - ←
Wcast-qual -Wcast-align -Wwrite-strings -Wstrict- ←
prototypes -Wmissing-prototypes -Wnested-externs - ←
D_POSIX_C_SOURCE=200112L -I../.. /include -c -o sample.o ←
sample.c

```

APPENDIX C. A/D-BOARD SOURCE CODE

```
gcc -lm -lrt -lpthread -lpci_dask2k -lmyutils sample.o -o ↔  
sample
```