BAKKALAUREATSARBEIT

embedded operating systems / tinyOS

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Bakkalaureus der Technischen Informatik

unter der Leitung von

Dipl.-Ing Alexander Kössler
Institut für Technische Informatik 182

durchgeführt von

Harad Glanzer
Matr.-Nr. 0727156
Hardtgasse 25 / 12A, 1190 Wien

Wien, im September 2011 ....

embedded operating systems / tinyOS

Subject of this work is to give an introduction to the TinyOS Embedded Operating System, to explain the internal structure of this OS and to present its main components. Additionally, tinyOS is compared to another embedded OS, MicroC/OS-II. Because tinyOS will be used in future lectures at the TU Wien, there is also a chapter about installing tinyOS from scratch. Also, there is a chapter about new tinyOS modules for the bigAVR6 - developementplatform that have been written by me in the course of my project thesis.

Contents

1. Introduction

This work is about explaining tinyOS. tinyOS is an opensource operating system, designed for use with wireless embedded sensor networks. There are 2 major stable branches, v.1.x and v2.x, which are not compatible to each other. tinyOS 2.x introduced some major improvements, for example the task scheduler was completely redesigned. In it's new version, the whole project us now distributed under the new BSD license too.

Because of tinyOS's component-based architecture, a highlevel programmer does not have to care about microcontroller specifics, as long as the necessary modules are already exisiting. So, implementing new applications or changing exisiting ones is an easy and fast task. There are already existing implementions for a range of popular hardware notes, as for example the mica, iris and teleosa motes.

For developing applications in tinyOS, NesC is used. NesC stands for Network Embedded Systems C, which is very similar to C/C++. Components in NesC are related to objects in C++.

Embedded systems are designed for one or a few specific tasks, perhaps in combination with realtime constraints. Because embedded systems are often battery powered, on of the main requirements is low power consumption, so that high operation times can be achieved - flexibility is not that important.

This work has to be considered as part of another work, where tinyOS was ported to a new platform, the bigAVR6 development platform. So, a big part of this work handles about the bigAVR6 platform an explains some new written softwaremodules for highlevel-use of this platform's peripherals. Also, it is explained how to get a running buildenvironment for writing applications or extending the modules for an exisiting platform from scratch.

1.1. Motivation and Objectives

Ultimate goal of this work is to provide an easy-to-use manual for using tinyOS to the reader, so that a buildenvironement can be set up from scratch step-by-step without any necessary previous knowledge. By comparing tinyOS to other embedded OS FIXME the specific characteristics of tinyOS will get much clearer. Finally, by guidung through the selfwritten software modules for the bigAVR6-board, the reader will get familiar with how to practically extend the tinyOS framework by using as much of the preexisting components and how to properly integrate the selfwritten code into tinyOS.

1.2. Structure of the Thesis

The thesis is structured as follows: Chapter 5 gives an introduction into the basic terms and concepts used throughout the work.

Chapter 2 gives a more detailed overview over tinyOS and explains its internals. It also compares tinyOS to another embedded operation system, MicroC/OS-II.

Chapter 3 introduces the developement platform for which tinyOS was ported to by the author, and explains the supported devices for this platform.

Chapter 4 is a step-by-step howto for setting up a fresh tinyOS - environment.

Finally, the thesis ends with a conclusion in Chapter 10 summarizing the key results of the presented work and giving an outlook on what can be expected from future research in this area.

2. tinyos

2.1. What is tinyOS

tinyOS is a free and open operating system for hardware motes, which is written in nesC. Development started as a collaboration of Berkeley University with Intel Research and Crossbow Technology, a california based company.

A hardware mote is a microcontroller based node in a wireless sensor network, capable of reading sensory information, processing and exchanging of this data with other nodes. Communication typically goes on over wireless networks.

One of tinyOS's most important features is that tinyOS applications are built out of components, which are connected or wired to each other by interfaces.

2.2. History

Developement of tinyOS started in 1999 at Berkeley university. The first supported platform was a mote called 'WeC', as shown in figure 2.1. For communication, this platform is equipped wit an radio device and SPI and UART - interfaces. As cpu an Atmel AVR AT90LS8535 microprocessor, clocked with 4MHz, is used. A major advantage of this mote is that it can be programmed over the wireless interface.

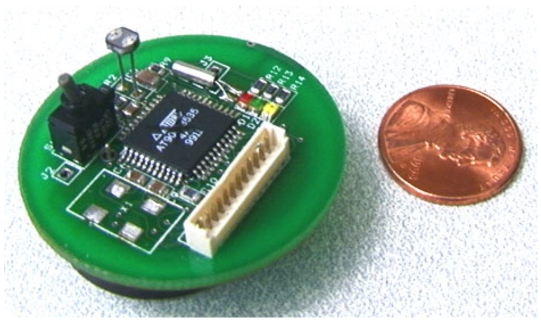


Figure 2.1.: WeC Mote

In the next years, the 'rene' and 'mica' platforms are developed. In the year 2002, work on the nesC programming language began. Until that, tinyOS consisted of a mixture of C files and Perl scripts. In the same year, tinsOS 1.0, the first tinyOS implemented in nesC, is released.

Improvement of tinyOS 1.x went on until february 2006, when tinyOS 2.0 beta1 was released. Version 2.1 was finished in april 2007. Among numerous bugfixes, a cc2420 wireless radio stack implementation was added in this release. tinyOS 2.02 was released some months later, which included an cc2420 stack reimplementation and bugfixes. After that, in august 2008, support for the 'iris' and 'shimmer' platforms was added by distribution of version 2.1. Clearly, bugfixes were included in this release too.

At the time of this writing, the latest tinyOS version is 2.1.1, which was released in April 2010. Most important add-ons are support for the 'mulle', 'epic' and 'shimmer2' - platforms.

2.3. Supported Platforms

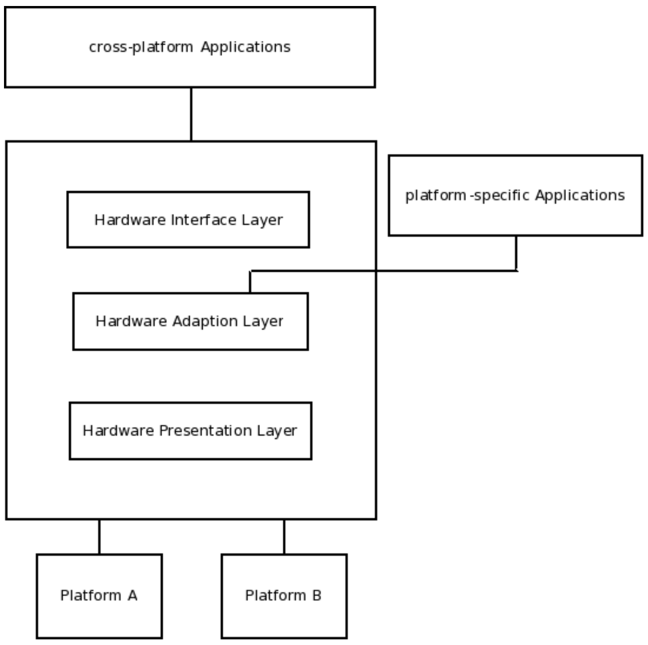
With version 2.1.1, a range of hardware motes are supported out-of-the-box. A brief description of this platforms and its most important communication devices are given in the following list. Obviously, most of this platforms provide interfaces for UART, I2C, SPI or others peripherals, depending of the microcontroller and extension boards used, too. These features are not listed here.

- btnode3: Atmega128L cpu and radio/bluetooth communication devices
- epic: MSP430 cpu and CC2420 radio chip
- eyesIFX: MSP430F149/F1611 and TDA5250 wireless transceiver
- intelmote2: PXA271 XScale cpu and CC2420 radio chip
- mica: Atmega103 and TR1000 radio chip
- mica2: Atmeage128L and Chipcon 868/916 radio chip
- mica2dot: Atmega128 and cc1000 transceiver
- micaz: Atmega128 and cc2420 radio chip
- mulle: Renesas M16C and AT86RF230 transceiver
- sam3s_ek: SAM3S4C chip and cc2520 transceiver
- sam3u_ek: SAM3S4C chip and cc2420 transceiver

- shimmer / shimmer2 / shimmer2r : MSP430 cpu and CC2420 transceiver
- span: MSP430 cpu and CC2420 transceiver
- telosa / telosb: MSP430 cpu and CC2420 transceiver
- tinynode: MSP430 cpu and Semtech SX1211 transceiver
- ucmini: Atmega128RFA1 cpu(low power transceiver cpu integrated)
- z1: MSP430 cpu and CC2420 transceiver

2.4. tinyOS hardware abstraction

When looking at the supported hardware platforms, one can see that many platforms use the same cpu and/or communication hardware. To avoid rewriting of code, hardware abstraction is used. By introducing hardware abstraction, it is easier to port applications from one platform to another, and application development itself gets easier too. On the other hand, abstraction means generalisation, which is problematic because hardware motes only have very limited resources and strict energy-efficiency requirements. So, tinyOS uses a 3-level *Hardware Abstraction Architecture* to provide a flexible and performant framework to build applications on, as shown in figure 2.2.


cross-platform Applications
Hardware Interface Layer
platform-specific Applications
Hardware Adaption Layer
Hardware Presentation Layer
Platform A
Platform B
Figure 2.2.: Hardware Abstraction Architecture

In contrast to other embedded OS that use only 2 layer abstraction, the third tinyOS layer provides more flexibility. For maximum performance, a *Platform-Specific-Application* can directly hook into the *Hardware Adaption Layer*, circumventing the *Hardware Interface Layer*.

2.5. tinyOS internals

2.5.1. Basic - Scheduler

By default, tinyOS 2.x uses a **non-preemptive FIFO** scheduler with a maximum of 255 parameterless tasks waiting for execution. A task can only be scheduled once at a time, if periodic execution is needed the task has to re-post itself just before finishing. The scheduler itself consists of an interminable for-loop, that pops executes one task after the other, in sequence as this tasks got pushed. If no tasks are waiting for execution, the scheduler enters sleepmode immediatly.

Because the scheduler of tinyOS is implemented as a component it is possible to replace this default FIFO-scheduler with a selfwritten one. See [Lev03] for details how to implement a selfwritten scheduler.

2.5.2. Microcontroller Power Management

To reduce power consumption, a microcontroller should always run in the lowest power state possible. As mentioned above, tinyOS enters a low power mode if the task queue is empty. Normally, microcontrollers support a full range of low power modes, the ATmega128 - for example - supports up to 6 different power saving modes. To decide what mode fits best, tinyOS uses the control- and statusregisters to find the proper lowpower-mode.

For example, on a ATmega128-based platform, the cpu-specific powersaving-mode *IDLE* is to be choosen if timer, SPI, UART or I2C are in use. If the ADC-submodule is working, another mode, *ADC Noise Reduction*, is entered. If none of this modules are active, the cpu is set to *POWER DOWN*.

3. bigAVR6

3.1. The Hardware

3.2. Supported Modules

3.2.1. GLCD

3.2.2. MMC

3.2.3. Ethernetboard

3.2.4. LCD 2x16

4. buildenv

4.1. Prerequisites

5. Concepts

Usually, this chapter is necessary in order to give an overview on the *terms* and *concepts* that are required to understand your work.

5.1. Writing Style

Usually you should not use the first person singular (I) in your text, write *we* instead. As a general recommendation, use the first person sparsely, sometimes it can be replaced by a phrase like *This work presents...*.

The indefinite article **a** is used as **an** before a vowel sound - for example **an** apple, **an** hour, **an** unusual thing, **an** FPGA (becourse the acronym is pronouned Ef-Pee-Gee-A), **an** HIL. Before a consonant sound represented by a vowel letter **a** is usual – for example **a** one, **a** unique thing, **a** historic chance[1].

5.2. Acronyms

Explain acronyms at their first occurrence in the text. In order to achieve this consistently, we recommend to use the `acronym` package.

A new acronym is then declared by writing `\newacro{acronym}{expanded name}`. Use the macro `\ac{acronym}` as a placeholder for the acronym in the text.

See file `acronym.tex` for further examples and explanations.

5.3. Figures

A Figure should always be referenced in the text, as it is the case with Figure 5.1.

[1]According to Merriam Webster, both **a** and **an** can be used in writing before unstressed or weakly stressed syllables with initial h, thus you could also write **an** historic chance.

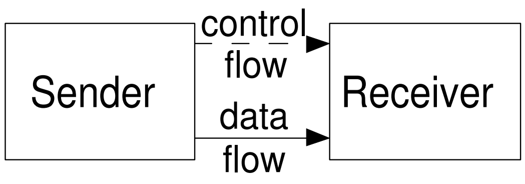


Figure 5.1.: Example figure

This template can be compiled with the `latex` command or the `pdflatex` command. While `latex` creates an intermediate file format (.dvi) that can be further processed into a `.ps` or `.pdf` file, the `pdflatex` command directly creates a `.pdf` file.

Note that with `latex` the `\includegraphics` accepts only .eps files, while with `pdflatex` accepts `.pdf`, `.png`, or `.jpg`. Luckily, the file extension can be omitted in order that `\includegraphics{pics/example}` will look for file with name `example.eps` in `latex` mode and for a file with name `example.pdf`, `example.png`, or `example.jpg` in `pdflatex` mode. If you already have an `.eps` file, you may create a respective `.pdf` file with the commandline conversion tool `epstopdf`.

5.4. Citations and References

Whenever you refer to previously published work, you should set a reference to acknowledge the work you build upon. For example this is a reference to two bachelor's theses [Kra03, Wei05]. If you literally cite a part of someone else's work, mark the respective sentence by quotes and italic letters and add the page number, where is text can be found:

"*An intelligent or* smart *transducer is the integration of an analog or digital sensor or actuator element, a processing unit, and a communication interface. In case of a sensor, the smart transducer transforms the raw sensor signal to a standardized digital representation, checks and calibrates the signal, and transmits this digital signal to its users via a standardized communication protocol.*" [EP05, p. 175]

5.5. Spellchecking

Do not use your advisor as your spell checker. Instead, run an electronic spell checker over your document before submitting the document to your advisor.

5.6. References with Bibtex

Bibtex is an additional program to LaTeX that creates a list of your cited references in a chapter named *Bibliography*. In order to use Bibtex, you must maintain a database of all references in so-called *bibfiles* (file extension `.bib`).

The *bibfiles* contain entries of several types, the most needed types are `book`, `inproceedings`, `article`, `techreport`, `mastersthesis`, and `phdthesis`. In the following we list the templates for these types, whereas each asterisk (*) should be replaced by the respective data, if this is not available, the element should be left out. The case of the element names does not matter to Bibtex, however in the examples we have used UPPERCASE for the obligatory fields and lowercase for the optional fields. To see some complete examples, have a look into the file `bibfile.bib`. For more information, read [Pat88].

5.6.1. Some BibteX Examples

```
@BOOK{*,
  AUTHOR =       {*},
  editor =       {*},
  TITLE =        {*},
  PUBLISHER =    {*},
  YEAR =         {*},
  volume =       {*},
  number =       {*},
  series =       {*},
  address =      {*},
  edition =      {*},
  month =        {*},
  note =         {*}
}

@INPROCEEDINGS{*,
  AUTHOR =       {*},
  TITLE =        {*},
  BOOKTITLE =    {*},
  YEAR =         {*},
  editor =       {*},
  volume =       {*},
  number =       {*},
  series =       {*},
  pages =        {*},
  address =      {*},
  month =        {*},
  organization = {*},
  publisher =    {*},
  note =         {*}
}

@ARTICLE{*,
  AUTHOR =       {*},
  TITLE =        {*},
  JOURNAL =      {*},
  YEAR =         {*},
  volume =       {*},
  number =       {*},
  pages =        {*},
  month =        {*},
  note =         {*}
}

@TECHREPORT{*,
  AUTHOR =       {*},
  TITLE =        {*},
  INSTITUTION =  {*},
  YEAR =         {*},
  type =         {*},
  number =       {*},
  address =      {*},
  month =        {*},
  note =         {*}
}

@MASTERSTHESIS{*,
  AUTHOR =       {*},
  TITLE =        {*},
  SCHOOL =       {*},
  YEAR =         {*},
  type =         {*},
  address =      {*},
  month =        {*},
  note =         {*}
}

@PHDTHESIS{*,
  AUTHOR =       {*},
  TITLE =        {*},
  SCHOOL =       {*},
  YEAR =         {*},
  type =         {*},
  address =      {*},
  month =        {*},
  note =         {*},
  abstract =     {*},
  keywords =     {*},
  source =       {*},
}
```

6. Related Work

This chapter should give an overview over existing work that is related to your work. Instead of “Related Work”, this chapter can also be named specifically to the topic of the thesis.

For example in Bernhard Weirich’s Bachelor’s thesis, there are two chapters on related work named “Time-Driven Algorithms” and “Event-Driven Algorithms” [Wei05].

Each related approach should be described by a section of about 100-500 words.

6.1. Types of Bachelor’s Theses

If you write a plain report on some implementation, you might have no chapter on related works.

7. Design Approach

If you have derived some new concepts on your own, this is the place to present them. You can use generic names for this chapter like “Design Approach” or “System Architecture” or chose name accordingly to its contents (for example “Automatic Text Generator Algorithm”).

7.1. Types of Bachelor’s Theses

If you write a Bachelor’s thesis in form of a survey, you might have several chapters on existing work from others, but no chapter as described here.

8. Implementation

Call this “Implementation” or “Case Study”, here you will describe you actual hands-on part of your work.

8.1. Types of Bachelor’s Theses

If you write a Bachelor’s thesis in form of a survey, you might have several chapters on existing work in the literature, but no chapter as described here.

9. Results and Discussion

Name this chapter "Results and Discussion", "Experimental Results", "Evaluation" or "Experiments and Evaluation". First present your measurements here, usually in form of graphs and tables. Second, discuss it. Explain for example missing or misplaced data points[1].

If this chapter grows too large, you might split it into two separate chapters, for example "Results" and "Discussion".

9.1. Tables

Sensor	Mean squared error (cm^2)	Mean absolute error (cm)	Estimated variance (cm^2)	Respective confidence
IR 1 ($d \leq 80$ cm)	228.38	5.97	212.98	4
IR 1 (hybrid)	860.87	14.35	686.08	2
IR 1 ($d > 110$ cm)	1880.50	28.27	1078.30	2
IR 2 ($d \leq 80$ cm)	242.04	7.25	233.15	4
IR 2 (hybrid)	226.04	7.15	206.56	4
IR 2 ($d > 110$ cm)	162.13	6.24	127.92	5
IR 3 ($d \leq 80$ cm)	108.45	7.35	104.82	5
IR 3 (hybrid)	795.57	18.59	538.91	3
IR 3 ($d > 110$ cm)	1945.08	37.65	505.68	3

Table 9.1.: Quality of calibrated infrared sensor data (from [Elm02])

Table 9.1 shows an example of a table set in LaTeX. To be honest, making tables in LaTeX is a complicated and sumptuous task. A good tutorial about setting tables in LaTeX is presented bei Axel Reichert in [Rei99] (in German language).

9.2. Types of Bachelor's Theses

If you write a Bachelor's thesis in form of a survey, you might have several chapters on existing work in the literature, but no chapter as described here.

[1]By the way, do not refer to colored elements in a figure or graph, assume the user prints out your thesis in black and white

10. Conclusion

The chapter “Conclusion”, sometimes also named “Summary” should contain two things:

Main contribution(s) of this work – Why is the world a better one now ;-)

When you write the conclusion, assume that some quick readers might not have gone through the whole thesis but are merely peeking into the conclusion, therefore avoid complicated nomenclature here.

Outlook – what could be the next steps or possible extensions for this research?

Bibliography

[Elm02] W. Elmenreich. *Sensor Fusion in Time-Triggered Systems*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002.

[EP05] W. Elmenreich and S. Pitzek. Smart transducers – principles, communications, and configuration. In J. A. T. Machado W. Elmenreich and I. J. Rudas, editors, *Intelligent Systems at the Service of Mankind*, volume II, pages 175–186. Ubooks, December 2005.

[Kra03] H. Kraut. Signalverarbeitung mittels eines Neuronalen Netzwerkes für einen Smart Sensor. Bachelor's thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2003.

[Lev03] Philip Levis. Scheduler and Tasks. Bachelor's thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2003.

[Pat88] O. Patashnik. *BibTeXing*, 1988.

[Rei99] A. Reichert. *Satz von Tabellen*, 1999.

[Wei05] B. Weirich. Wireless real-time communication. Bachelor's thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2005.

A. Setup Guide

You have implemented some system others will use? They will surely acknowledge a short setup guide here. Describe the system requirements and setup steps in a precise way here, like for example:

A.1. System Requirements

In order to use the *thesis template*, you need to install LaTeX, furthermore an editor supporting LaTeX is recommended.

A.1.1. Required Software for Windows

For compiling this template, we used the actual version of MikTeX, which is a an up-to-date implementation of TeX and LaTeX for all current variants of Windows on x86 systems. MikTeX is freely available at `http://www.miktex.org`.

As an editor, we recommend the free *TeXnicCenter* (available at `http://www.toolscenter.org`). Both, MikTeX and TeXnicCenter are published under the Gnu Public License (GPL).

TeXnicCenter comes with an integrated spell checker, otherwise you are recommended to install the Windows version of *aspell*, an open source spell checker under the GPL, which is available at `http://aspell.net/win32/`.

If also want to do grammar checking, try Queequeg (`http://queequeg.sourceforge.net/index-e.html`) or see guides like the one at `http://www.physics.usyd.edu.au/guides/spell-grammer-latex.html`.

A.1.2. Required Software for Linux and BSDs

The standard distributions for Linux already come with a LaTeX system (typically `tetex`).

As an editor, we recommend the Kile editor (available at `http://kile.sourceforge.net/` under GPL).

As spell checker we recommend *aspell*, an open source spell checker that replaces the older *ispell* checker. *aspell* is included in most distributions, otherwise it can be downloaded from `http://www.gnu.org/software/aspell/`. If also want to do grammar checking, try Queequeg (`http://queequeg.sourceforge.net/index-e.html`) or see guides like the one at `http://www.physics.usyd.edu.au/guides/spell-grammer-latex.html`.

A.1.3. Required Software for Apple Mac OS X

The *darwin ports* (`http://darwinports.opendarwin.org/`) provide a port of *teTeX* that can be installed under Apple Mac OS X.

As an editor, we recommend TeXShop (available at `http://www.uoregon.edu/~koch/texshop/` under GPL).

As spell and grammar checker we recommend Excalibur (`http://www.eg.bucknell.edu/~excalibr/`).

A.2. Installing the Thesis Template

The thesis template comes in a zip-archive. Simply extract the archive into a directory of your choice and start working.

A.3. Types of Bachelor's Theses

Of course, not all Bachelor's theses require a setup guide.

B. User Guide

You have implemented some system others will use? If you were in their place what kind of documentation would you like to have in order to start working?

For complex programs, a tutorial that guides the user through a typical task showing screenshots of the intermediate states is desirable.

B.1. Types of Bachelor's Theses

Of course, not all Bachelor's theses require a user guide.

B.2. How to use the Thesis Template

B.2.1. Files to Edit

You should edit the following files (unless you remove some of them, see Section B.2.2):

- `title.tex`
- `abstract.tex`
- `acronyms.tex`
- `introduction.tex`
- `concepts.tex`
- `relatedwork.tex`
- `designapproach.tex`
- `implementation.tex`
- `results.tex`
- `conclusion.tex`
- `setupguide.tex`
- `userguide.tex`

B.2.2. Removing Chapters

Open the file `thesis.tex` and remove (or insert a comment) at the lines with the include-commands, for example:

Before	After
`\include{concepts}`	`\include{concepts}`
`\cleardoublepage`	`\cleardoublepage`
`\include{relatedwork}`	`%\include{relatedwork}`
`\cleardoublepage`	`%\cleardoublepage`
`\include{designapproach}`	`\include{designapproach}`
`\cleardoublepage`	`\cleardoublepage`

Table B.1.: Removing the Chapter `Related Work`

B.2.3. Adding Chapters

Open the file `thesis.tex` and add an `\include{`*filename*`}` command (followed by a `\cleardoublepage`) at the respective line. Then create a new file *filename*`.tex` in the same directory and write the chapter's contents into it.

B.2.4. Adding References

Whenever you `\cite` something, there must be a respective entry in any of the included .bib files. To add such an entry, open the respective bibfile (e. g., `bibfile.bib`) with a text editor and add the entry according to the bibfile syntax. The reference will be added after running latex/bibtex/latex on your project.

B.2.5. Removing References

Remove all `\cite` commands to this reference in the text and the citation will not be listed in the bibliography after running latex/bibtex/latex on your project. There is no need to remove the reference from the .bib file.

B.2.6. Changing the .bib Files to be used

In the file `thesis.tex` there is a line with a `\bibliography` command, that lists all used .bib files without file extensions separated by commas.

B.2.7. Troubleshooting

If you make something wrong, you should get an error at compile time, except for citation problems, like:

Adding or removing references does not work: Perhaps there is a spelling error in some .bib file, this causes the program `bibtex` to abort execution leaving the old bibliography unchanged.

Changing a reference does not work: Some systems only perform a new bibtex run if there are missing references, delete the `.aux` file in order to overcome this problem.

References show a question mark: Bibtex could not find a matching entry in the bibfile for this reference, check the label name.