

# A modular Software Package for the Embedded Systems Engineering Board

BACHELOR THESIS

submitted in partial fulfillment of the requirements for the degree of

**Bachelor of Science (BSc)**

in

**Computer Engineering**

by

**Jürgen Galler**

Registration Number 0825384

to the Faculty of Informatics  
at the Vienna University of Technology

Advisor: Univ.-Ass. Dipl.-Ing. Armin Wasicek

Vienna, TT.MM.JJJJ

\_\_\_\_\_  
(Signature of Author)

\_\_\_\_\_  
(Signature of Advisor)

# Statement

Jürgen Galler  
Linke Brückenstraße 19, 4040 Linz

Hereby I declare that this work has been written autonomously, that all used sources and utilities are denoted accordingly and that these points of the work - including tables, maps and figures - which were taken from other creations or the Internet have been marked as borrowing by quoting the original sources. This document at hand will not be submitted to any other course.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

---

(Ort, Datum)

---

(Unterschrift Verfasserin)

# Abstract

The Embedded Systems Engineering (ESE) laboratory course at TU Vienna uses a custom-made microcontroller board. This board contains four microcontrollers, which have to communicate via a 1-wire bus. Because of old hardware, the actual boards have to be exchanged. The topic of this Bachelor thesis is the implementation of a complete software package for using the new ESE exercise board. The package includes a JTAG programmer, a monitoring software (a logic analyzer) and an application for data exchange between the host computer and the microcontrollers (nodes) on the board.

The package is divided into a firmware for the control device (a LPC1768 microcontroller) on the board and a software for the host computer. The firmware is based on the realtime operating system *FreeRTOS* [1]. The host computer and the board communicate via USB.

This paper provides a complete description of the software package. Furthermore, the thesis contains an evaluation of the software package and examples for using the software. The appendix includes a manual for installing the host software of the monitoring tool.

Die Embedded Systems Engineering (ESE) Laborübung an der TU Wien benutzt ein maßgeschneidertes Microcontroller Hardwareboard. Dieses Board beinhaltet vier Microcontroller, welche über einen 1-wire Bus kommunizieren. Wegen veralteter Hardware müssen die aktuellen Boards ersetzt werden. Diese Bachelorarbeit befasst sich mit der Implementierung eines Softwarepakets für das neue ESE Übungsboard. Das Softwarepaket besteht aus einem JTAG Programmer, einer Monitoring Software (einen Logic Analyzer) und einer Anwendung zum Datenaustausch zwischen Host Computer und den diversen Microcontrollern (Nodes) am Board.

Das Softwarepaket ist in eine Firmware für den Steuerbaustein (ein *LPC1768* Microcontroller) auf dem Board, und eine Software für den Hostrechner unterteilt. Die Firmware basiert auf dem Echtzeit-Betriebssystem *FreeRTOS* [1]. Die Kommunikation zwischen diesen Modulen erfolgt über USB.

Diese Bachelorarbeit beinhaltet eine ausführliche Beschreibung des entwickelten Softwarepakets. Zusätzlich beinhaltet die Arbeit eine Auswertung, sowie einige Beispiele für den Einsatz der Software. Der Anhang beinhaltet eine Anleitung für die Installation der Monitoring Software.

# Contents

<b>List of Figures</b>	<b>v</b>
<b>List of Tables</b>	<b>vi</b>
<b>Listings</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation and Objectives . . . . .	1
1.2 Structure of the Thesis . . . . .	2
<b>2 Related Work</b>	<b>3</b>
2.1 ESE-Board Hardware Design (Staub) . . . . .	3
2.2 JTAGProg (Schmölzer) . . . . .	3
2.3 FreeRTOS . . . . .	4
2.4 libusb . . . . .	4
2.5 lpcusb . . . . .	4
2.6 GTKWave . . . . .	4
<b>3 Design Entry</b>	<b>5</b>
3.1 System structure . . . . .	5
3.2 JTAG programmer . . . . .	5
3.3 Logic analyzer . . . . .	6
3.4 USB-Serial converter . . . . .	7
<b>4 Implementation</b>	<b>8</b>
4.1 USB device setup . . . . .	8
4.2 FreeRTOS configuration . . . . .	11
4.3 JTAG Programmer . . . . .	12
4.4 Logic analyzer . . . . .	12
4.5 USB-Serial converter . . . . .	15
<b>5 Evaluation</b>	<b>17</b>
5.1 Usecases . . . . .	17
5.2 Method of measurement . . . . .	18

5.3	Programming a single node . . . . .	18
5.4	Programming all nodes . . . . .	20
5.5	Monitoring nodes . . . . .	21
5.6	UART data exchange with a single node . . . . .	22
5.7	Analyzing the communication between two nodes . . . . .	22
5.8	CPU utilization . . . . .	23
<b>6</b>	<b>Conclusion</b>	<b>24</b>
6.1	Achieved improvements . . . . .	24
6.2	Outlook . . . . .	24
<b>A</b>	<b>Using the Logic Analyzer</b>	<b>26</b>
A.1	Setup . . . . .	26
A.2	Commandline tool . . . . .	27
A.3	GTKWave . . . . .	29
	<b>Bibliography</b>	<b>31</b>

# List of Figures

3.1	System structure: Integration of this project into the complete system. . . . .	6
4.1	Typical configuration of an USB device (source: [3]) . . . . .	9
4.2	Actual USB configuration of the ESE board (simplified) . . . . .	9
4.3	Waveform related to the example VCD-file . . . . .	16
5.1	Usecase diagram . . . . .	17
5.2	Execution time of programming a single node . . . . .	20
5.3	Execution time of programming all (main) nodes . . . . .	21
A.1	GTKWave: Adding signals to timeline 1 . . . . .	29
A.2	GTKWave: Adding signals to timeline 2 . . . . .	30
A.3	GTKWave: Setting data format (4) and zoom options (5) . . . . .	30

# List of Tables

4.1	Settings for composite devices . . . . .	8
4.2	USB interfaces of the actual setup . . . . .	10
4.3	Settings for composite devices . . . . .	10
4.4	List of USB endpoints . . . . .	11
4.5	Structure of a logic analyzer command . . . . .	13
4.6	List of special functions of the sampled signals . . . . .	14
5.1	System configuration used for measurement . . . . .	18
5.2	Programming a single node: data used for the evaluation . . . . .	18
5.3	Programming all nodes: data used for the evaluation . . . . .	20
5.4	Average CPU utilization . . . . .	23

# Listings

4.1	Using unsupported commands . . . . .	12
4.2	VCD-file . . . . .	15
5.1	Example of using the time-command . . . . .	18
5.2	Programming <i>node 0</i> of the ESE board . . . . .	19
5.3	Programming all (main) nodes of the ESE board . . . . .	20
5.4	LogAn examples . . . . .	22
A.1	Installing the LogAn commandline tool . . . . .	26
A.2	udev rule for the ESE board . . . . .	27
A.3	Installing the LogAn commandline tool . . . . .	27
A.4	LogAn synopsis . . . . .	27
A.5	Using GTKWave . . . . .	29



# Introduction

## 1.1 Motivation and Objectives

Students who study *Computer Engineering* at Vienna University of Technology have to complete the laboratory course Embedded Systems Engineering (ESE). The course is based on a custom-made microcontroller board, containing four networked single microcontrollers. Because the old boards have reached the end of their lifetime, a new hardware board has to be constructed. The new board must include multiple microcontrollers (nodes) which have to be programmable individually. The nodes of the board are Atmel AVR microcontrollers with an additional ZigBee module. The Atmel microcontrollers are equipped with different peripheral hardware, such as a LCD, a fan, and a LED-matrix. The main task for the students of the ESE course is to write a communication protocol for these nodes which is based on a serial communication (UART), where sending and receiving data uses only one wire.

The programming of the nodes is supported by a dedicated powerful Cortex-M3 microcontroller (NXP LPC1768). For this purpose the modular XML-based JTAG Programmer from *Martin Schmölzer* [7] was ported to the LPC1768. The firmware uses the embedded realtime operating system *FreeRTOS* [1]. This decision has the advantage that the design becomes more modular. To enhance functionality two other features were added. On the one hand an embedded logic analyzer was implemented, to provide better methods for debugging. On the other hand the firmware includes an USB-Serial converter to enable communication between a personal computer and the specific nodes.

The contribution of this work is the implementation of the firmware which runs on the LPC1768 and its counterparts on the PC.

## 1.2 Structure of the Thesis

Chapter 2 introduces projects which are related to this thesis, like the ESE Hardware Design [9], and the modular XML-based JTAG Programmer [7].

The system design is illustrated in Chapter 3. In addition the connection between the hardware design and the software package is explained on a high level of abstraction. Furthermore the composition of the project into the three main functionalities - JTAG programmer, logic analyzer, USB-Serial converter - is presented.

Chapter 4 gives a deep insight into the actual implementation. After pointing out the USB device setup, the configuration of FreeRTOS is described. Subsequently the specialities of the ported programmer, the logic analyzer (host software and firmware) and the USB-Serial converter are explained.

Chapter 5 provides some usecases for the software package. In this context, a comparison between the time of programming of the old and the new ESE board is made.

Chapter 6 gives a brief summary of this work and indicates the advantages of the chosen system structure. Furthermore it points out the possibility to enhance functionality of the firmware and the host software by upgrading respectively adding new applications.

Appendix A contains a manual for the logic analyzer. After explaining the installation of the host software, the commandline tool is explained. Furthermore it is illustrated how to use GTKWave to display the generated VCD files.

## Related Work

This chapter holds a list of theses and projects which are related to this work, either because of direct interaction or because of the use of their output.

### 2.1 ESE-Board Hardware Design (Staub)

In the scope of Arvid Staub's bachelor thesis, he created the hardware for the actual ESE exercise board [9]. The board consists of four main nodes (ATmega 1281/128a microcontrollers), a ZigBee module and various peripheral hardware, for example a LCD display. Also on the board there is the NXP LPC1768 microcontroller which is used in this project. With the combination of Arvid's and this work, the Embedded Systems Engineering course gets a new exercise board which displaces the old one. Since the board has more features than the old one (ZigBee module, graphical LCD, two instead of one bus, etc.), it enables new exercises for the ESE course.

### 2.2 JTAGProg (Schmölzer)

In the context of his bachelor thesis, Martin Schmölzer has implemented a modular XML-based JTAG programmer (*JTAGProg*) [7]. It consists of a commandline program and a firmware for ATmega128 microcontrollers. Since the whole programming algorithms are encoded in XML files and not in the firmware, all types of JTAG devices can be programmed, even FPGAs.

Schmölzer's work was used for the new ESE board by porting his firmware to the NXP LPC1768 microcontroller. Since there are no changes in the communication protocol, the commandline tool has not been touched. This has the advantage, that both versions of the programmer can be used with the same commandline tool.

## 2.3 FreeRTOS

*FreeRTOS* [1] is a realtime operating system for a various kind of microcontrollers. It provides functionalities for running several tasks with different priorities. Furthermore it implements techniques for mutual exclusions, queues and other realtime related functionalities. *FreeRTOS* is open source and free to use.

## 2.4 libusb

The *libusb* library [5] is an open source host stack for USB communication which runs in user-mode. This stack provides an easy way for communicating with USB devices without the need of a special driver. *libusb* was developed for Linux but is also available for Microsoft Windows.

## 2.5 lpcusb

The *lpcusb* library [8] is an USB core stack for the built-in USB device of NXP microcontrollers. It provides procedures for interfacing the hardware and for USB enumeration and configuration. *lpcusb* is shipped with FreeRTOS and is free to use. Typical applications are USB virtual COM ports, USB mass storages, USB HID devices, etc.

## 2.6 GTKWave

*GTKWave* [2] is an open source tool for visualizing waveforms, which supports various fileformats like VCD/EVCD. *GTKWave* handles digital as well as analog signals. There are many possible representations of signals, such as binary, hexadecimal, real number, ASCII, etc. *GTKWave* was developed for Linux/GTK+ but is also available for other operating systems such as Microsoft Windows.

## Design Entry

This chapter gives an overview of the software concept and points out the environment of the project.

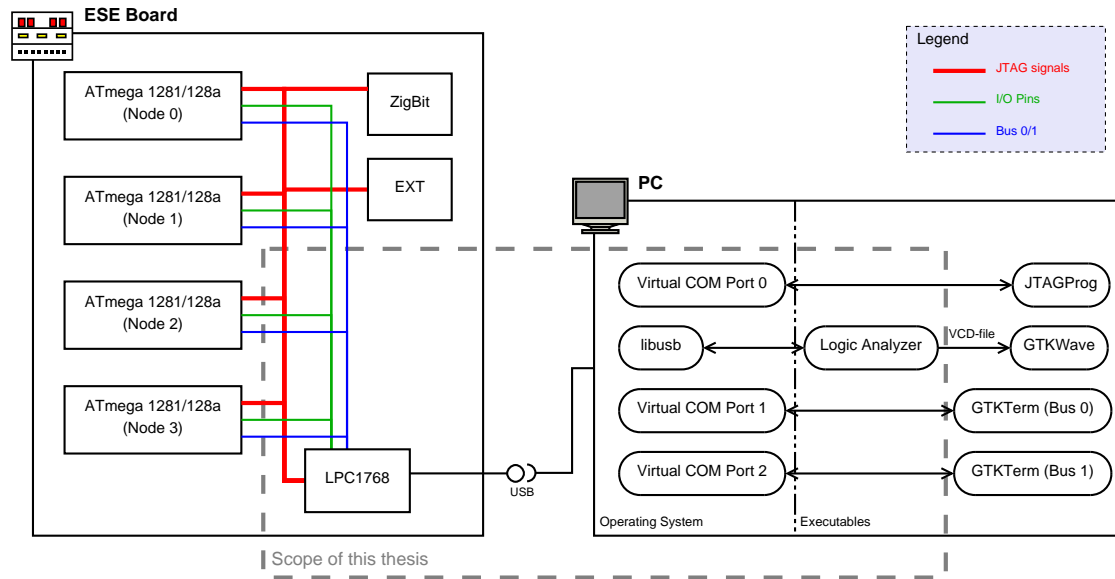
### 3.1 System structure

Figure 3.1 illustrates the system model of the new ESE board. The software package consists of a set of programs which run on a host computer with a Linux operating system installed and the firmware of the ARM Cortex-M3 microcontroller (LPC1768). While the LPC1768 is connected directly to the other components of the ESE board, the communication with the computer is done by USB (see chapter 4). USB has not only the advantage of plug&play and speed, but it is also possible to use multiple connections over just one physical USB port. With the help of these multiple connections it is possible to isolate the three applications - programmer, logic analyzer and USB-Serial converter - from each other.

### 3.2 JTAG programmer

The JTAG programmer consists of a Linux commandline tool and a firmware part. The firmware was written for ATmega microcontrollers. To use the programmer for the ESE board, the firmware was ported to the LPC1768. The algorithms for programming JTAG devices are encoded in XML-files, which are interpreted by the host program. The firmware part does not contain any information about the programming algorithms. It only receives commands to execute from the host program. This has the advantage, that almost all devices which support JTAG can be programmed. The only thing to do is writing a dedicated (XML-) configuration file for the device.

The programmer is responsible for programming the ATmega microcontrollers and the Zig-Bee module. In addition there exists a connector (*EXT*) on the board, where it is possible to attach further JTAG devices respectively a chain of JTAG devices. All devices share the same



**Figure 3.1:** System structure: Integration of this project into the complete system.

*Test Clock Input (TCK)*, *Test Data Input (TDI)* and *Test Data Output (TDO)* signals, but have dedicated *Test Mode Select (TMS)* wires. To select one device, the firmware changes the pin where the TMS signal is attached. This is different from the original firmware where a PLD was required for multiplexing of the TMS signal. The optional *Test Reset Input (TRST)* is not used by the programmer [7] (p. 17).

While the original programmer used only UART communication (which was then translated to USB communication by a FTDI FT232L USB-to-serial device), the ported firmware uses the built-in USB-device of the LPC1768 for communication with the commandline tool. To stay compatible to the old programmer, the USB communication emulates a virtual COM port. This has the advantage, that it was not necessary to change the commandline tool (*jtagprog*).

### 3.3 Logic analyzer

The logic analyzer is also divided into a commandline tool (*LogAn*) for Linux systems and a firmware part. The commandline tool uses libusb to interface the USB device. When the user executes the commandline tool, it sends a message including all set options to the firmware via USB. Subsequently the firmware waits for the (optional) trigger and then samples the wires with the given accuracy. When the sample buffer is full, the firmware transmits the samples back to the host program, where a VCD-file (see section 4.4) is generated. This file can then be displayed by a waveform viewer like GTKWave [2].

The green wires in figure 3.1 represent the dedicated I/O pins of the ATmega microcontrollers, which are sampled by the logic analyzer. On the one hand it is possible to sample 8 pins per node, and on the other hand it is possible to sample special functions of the nodes (like PWM,

bus, etc. - see table 4.6). The user can adjust the amount of observed nodes, the sample rate, the trigger and he can decide if he wants to use a pretrigger.

### **3.4 USB-Serial converter**

The blue wires in figure 3.1 represent the two *I-wire-busses* available on the ESE board, which are used for the communication between the four main nodes. To enable communication between the computer and the nodes, the two busses are connected to the LPC1768, which acts as a USB-Serial converter. The USB device setup of the LPC1768 emulates two virtual COM ports on the computer. These ports can then be used like any other serial port by a various kind of terminal programs, like GTKTerm.

# Implementation

This chapter explains the actual implementation. After introducing the USB device setup and the FreeRTOS configuration, specialities of the implementation are pointed out.

## 4.1 USB device setup

The USB standard allows a very wide range of device setups. Figure 4.1 shows the hierarchy of a typical configuration of an USB device and Figure 4.2 illustrates the actual configuration of the ESE board. On the top of the configuration there is a *device descriptor*, which specifies vendor id, product id, serial number and other device specific settings. Important are the device class, device subclass and the device protocol. One of the characteristics of USB is that all transfers are packet transfers. There is no possibility to use (real) stream transfers.

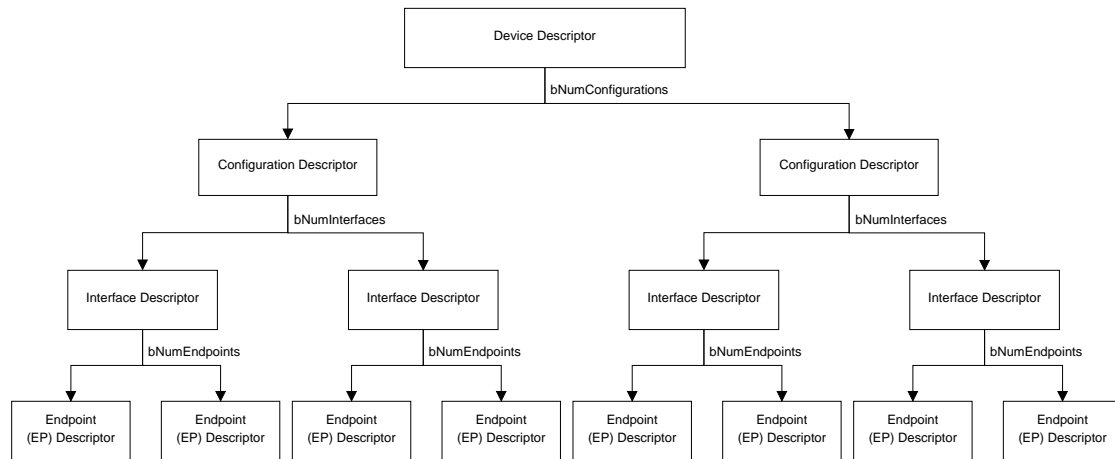
USB devices can i.e. be human interface devices (HID) (mice, keyboards, joysticks, etc.), mass storage devices or application specific devices. In the case of this project more than one application uses the same USB connection, so a composite device is used. Table 4.1 shows the necessary settings.

Setting	Value
device class	0xEF
device subclass	0x02
device protocol	0x01

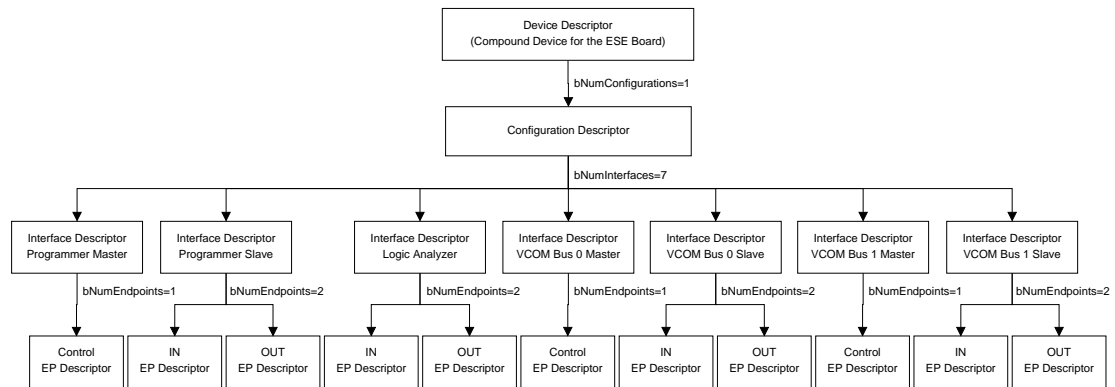
**Table 4.1:** Settings for composite devices

On the next level there is an arbitrary number of *configuration descriptors*. The USB standard allows to use more than one configuration, so that it is possible to switch the device configuration during operation. Since in case of this project there exists only one device configuration, only one configuration descriptor is used. This descriptor specifies the amount of interfaces. In case of virtual serial ports it includes an *interface association descriptor (IAD)* for each port.





**Figure 4.1:** Typical configuration of a USB device (source: [3])



**Figure 4.2:** Actual USB configuration of the ESE board (simplified)

This (sub-) descriptor specifies the interfaces used by the (virtual) serial port and the associated class. This is important, since it tells the USB stack at the host computer to use the associated driver for the port. In case of Linux this driver is called abstract control model (ACM). In the actual configuration there exist three IADs, one for each virtual serial port (programmer, bus 0 and bus 1).

The configuration is continued by various interfaces. For each application there exists at least one interface. Table 4.2 lists the interfaces for the actual setup. As the table shows, for virtual serial ports (VCOM) there are two interfaces, one for control transfers (used to set baudrate, parity, etc.) and one for data transfers. Table 4.3 outlines important settings for ACM applications.

On the bottom level of the configuration there are the endpoint descriptors. An endpoint is used for one direction of data transfer per application, so for bidirectional communication two endpoints are necessary. There are four types of endpoints (see [3] chapter 4):

Interface Nr.	Application	Interface type
0	Programmer	ACM control class interface (master)
1	Programmer	ACM data class interface (slave)
2	Logic analyzer	Data interface
3	VCOM bus0	ACM control class interface (master)
4	VCOM bus0	ACM data class interface (slave)
5	VCOM bus1	ACM control class interface (master)
6	VCOM bus1	ACM data class interface (slave)

**Table 4.2:** USB interfaces of the actual setup

- *Control transfers* are used for sending commands or status information. Typical applications for control transfers are setting up devices.
- *Interrupt transfers* are based on interrupt requests and should only be used for a small amount of data. Using interrupt transfers, errors can be detected and faulty transmissions are retried in the next USB cycle. In addition the latency is guaranteed.
- *Bulk transfers* are used for a large amount of data. Using bulk transfers, errors can be detected (by CRC) and the delivery is guaranteed. But the bandwidth can't be guaranteed. Bulk transfers are only available in USB full and highspeed mode (USB 1.1 and 2.0).
- *Isochronous transfers* are used for continuous/periodic information such as audio/video streaming. Using Isochronous transfers, errors can be detected (by CRC) but the delivery is *not* guaranteed. In addition the bandwidth is guaranteed and the latency is bound. Isochronous transfers are only available in USB full and highspeed mode (USB 1.1 and 2.0).

Setting	Value
interface class	0x02
interface subclass	0x02
interface protocol	0x01

**Table 4.3:** Settings for composite devices

For the ESE board it would make sense to use only bulk transfers, since they are fast and the delivery is guaranteed. The problem of this approach is that the LPC1768 has a fixed assignment of endpoints (see [6] p. 212f). This assignment allows only six bulk endpoints, but the actual configuration needs eight endpoints. The problem was solved by using two interrupt endpoints for the two virtual serial ports for bus 0 and 1. Table 4.4 lists all used endpoints with their direction and application. The lower half-byte encodes the endpoint number ( $0x0 - 0xF$ ) and the higher half byte encodes the direction of the transfer ( $0x0$  for OUT transfers and  $0x8$  for IN transfers).

Endpoint	Type	Direction	Application
0x81	Control	PC ← ESE board	Programmer
0x08	Bulk	PC → ESE board	Programmer
0x8B	Bulk	PC ← ESE board	Programmer
0x01	Bulk	PC → ESE board	Logic analyzer
0x85	Bulk	PC ← ESE board	Logic analyzer
0x84	Control	PC ← ESE board	VCOM Bus 0
0x07	Interrupt	PC → ESE board	VCOM Bus 0
0x8E	Bulk	PC ← ESE board	VCOM Bus 0
0x8A	Control	PC ← ESE board	VCOM Bus 1
0x0D	Interrupt	PC → ESE board	VCOM Bus 1
0x8F	Bulk	PC ← ESE board	VCOM Bus 1

**Table 4.4:** List of USB endpoints

## 4.2 FreeRTOS configuration

As mentioned before the firmware of the LPC1768 is based on the embedded realtime operating system FreeRTOS. This operating system is comparatively small in code size but one of the most powerful ones for embedded devices. To use the operating system, it has to be configured. The file *FreeRTOSConfig.h* contains the whole setup, where settings like the clock rate, the size of the stack, the mode of task preemption and the enabled API functions can be adjusted.

The first thing which has to be done by the firmware is to initialize the hardware. After that, the tasks must be started. Following configuration is chosen for the ESE board:

- The programming routines are executed when the host computer sends a request to the firmware. All characters received from the *jtagprog* commandline tool are parsed to interpret the JTAG commands. Furthermore a FreeRTOS software timer is used to detect timeouts.
- The logic analyzer part of the firmware waits for valid commands from the host computer. After receiving such a command it executes the initialization of a new sampling procedure and waits for its completion before it sends back the samples to the *LogAn* commandline tool. The actual sampling is done by a hardware timer in the according interrupt routine to be as accurate as possible.
- The functionality of the two USB-Serial converters is implemented directly in USB- and UART-interrupts, to enhance performance.

The specialities of the applications are described in the following sections.

## 4.3 JTAG Programmer

As mentioned in section 3.2, Martin Schmölzer wrote the original firmware of the JTAG programmer for Atmel AVR microcontrollers. In the scope of this thesis Schmölzer's code was ported to the LPC1768. The whole programming algorithm of the original firmware was integrated into the FreeRTOS configuration, while the communication with the host computer was changed from UART to native USB. For a maximum of compatibility the device emulates a virtual serial port on the host computer. This has the advantage, that there was no need to adjust the commandline tool of the Programmer to USB communication, so both - the old and the new programmer hardware - work with the same commandline tool. In addition there are no drawbacks of a virtual serial port compared to communication via a dedicated USB host stack (like libusb [5]).

### ARM and EEPROM

In Schmölzer's firmware some settings, like the firmware version are stored in the AVR's internal EEPROM. Since ARMs do not have any EEPROM, this feature is deactivated in the actual implementation for the ESE board. The settings can be changed by altering preprocessor macros (*#define*) of the firmware code. In future versions these settings can be stored either in a dedicated external EEPROM, or directly in the flash memory of the LPC1768. The method for programming the internal flash memory is called In-Application Programming (IAP, see [6]).

Since the commandline tool was not altered for the new firmware, commands like changing the firmware version lead to an error message. Listing 4.1 shows an example for such a message.

**Listing 4.1:** Using unsupported commands

---

```
1 > jtagprog -p /dev/ttyACM0 --set-hwversion 3.0.0
:: Checking link .... Found device:
   ESE Target, hardware version 2.0.0, firmware version 2.0.0
Error: Setting the hardware version failed (got bad response
      from programmer)
```

---

## 4.4 Logic analyzer

For the logic analyzer there exists a dedicated FreeRTOS task, which is responsible for receiving commands from the host computer. The structure of such a command is given in Table 4.5. After successfully parsing a command, a hardware timer is started to sample the dedicated IO pins of the four main nodes (*node 0 - 3*). After filling a buffer with the samples, the firmware dumps the obtained data to the host program, which generates a VCD-file for viewing of the waveforms.

With the chosen command structure the user is able to enable only the nodes he wants to investigate. Enabling less nodes enhances the number of samples. In other words if only one

Byte	Description
0	enabled Nodes and Triggers
1	Trigger pattern for node 0
2	Trigger pattern for node 1
3	Trigger pattern for node 2
4	Trigger pattern for node 3
5	Trigger mask for node 0
6	Trigger mask for node 1
7	Trigger mask for node 2
8	Trigger mask for node 3
9	Sampling interval
10	Sampling interval
11	Sampling interval
12	Sampling interval
13	Pretrigger
14	Pretrigger
15	Special flags

**Table 4.5:** Structure of a logic analyzer command

node is activated it is possible to investigate the IO pins four times longer than if all four nodes are activated.

The sampling interval in microseconds is an indicator for the accuracy. According to the *Nyquist–Shannon sampling theorem* (see [4] chapter 3.3 p. 94), the sampling rate has to be at least twice as high as the maximum frequency of changing on the signal wires. This means that the user must set the right sampling rate if he wants to gain correct results. For example if the sampled signals change at most every millisecond, the sampling interval must be at least  $500\mu s$ .

The pretrigger allows to record an amount of samples before the trigger gets fired. The maximum amount of pretrigger values depends on the number of enabled nodes and the buffer size.

The last byte contains special flags for the logic analyzer. While in the current implementation there exists only one special flag for enabling special functions, it is possible to extend the functionality in future versions. The special functions flag indicates if the user wants to sample the dedicated IO pins or special signals from the nodes like PWM, bus 0/1, etc. This is done by a further output signal of the LPC1768, which is used to multiplex the signals. A complete list of the assignment of special functions can be found in table 4.6.

Bit	Node 0	Node 1	Node 2	Node 3
7	Bus 0 TX <sup>1</sup>	Bus 0 TX <sup>1</sup>	Bus 0 TX <sup>1</sup>	Bus 0 TX <sup>1</sup>
6	Bus 0 RX <sup>1</sup>	Bus 0 RX <sup>1</sup>	Bus 0 RX <sup>1</sup>	Bus 0 RX <sup>1</sup>
5	Bus 1 TX <sup>2</sup>	Bus 1 TX <sup>2</sup>	Bus 1 TX <sup>2</sup>	Bus 1 TX <sup>2</sup>
4	Bus 1 RX <sup>2</sup>	Bus 1 RX <sup>2</sup>	Bus 1 RX <sup>2</sup>	Bus 1 RX <sup>2</sup>
3	Pin 1.3	Pin 1.3	Pin 1.3	Pin 1.3
2	Bus 1	HEAT REQ	LCD DDIR	LEDMATRIX SCL
1	Bus 0	FAN PWM	LCD TEAR	LEDMATRIX SD
0	Clock IRQ	FAN SENSE	LCD BLIGHT	LEDMATRIX LATCH

**Table 4.6:** List of special functions of the sampled signals

In the current version the host program generates a VCD-file of the samples, but the host program is written modular, so it is simple to add more output methods in the future. These can be i.e. other fileformats, databases or streams to wave-viewing programs like GTKWave.

For further information about using the logic analyzer see appendix A.

### Value Change Dump (VCD)

Listing 4.2 shows a minimal version of a generated VCD file. In the header the timescale is set to  $500\mu s$ , and two nodes (*node 0 and 1*) are activated. After that, all recognized changes of the signals are stored to the according timestamp (i.e. timestamp 1000 is equal to  $500ms$  after the start of the sampling phase). Figure 4.4 shows the related screenshot of the waveforms (viewed by GTKWave). The screenshot also illustrates some different data format representations of the signals. While *node 0* is represented as a hexadecimal value, *node 1* is represented as binary.

---

<sup>1</sup>Bus 0 from the view of the node.

<sup>2</sup>Bus 1 from the view of the node.

**Listing 4.2:** VCD-file

---

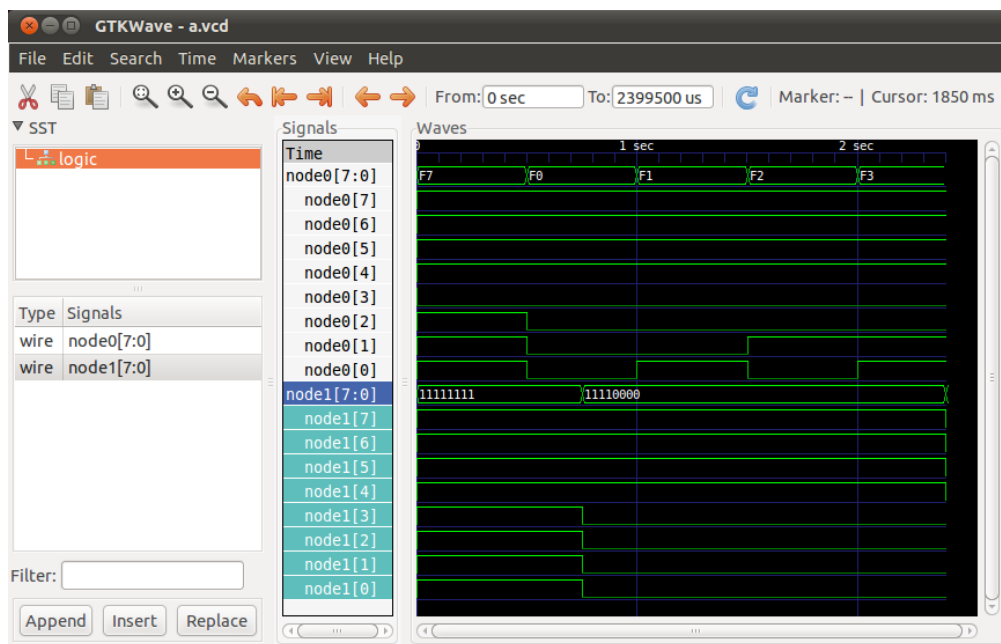
```
1 $comment Generated by the ESE Logic Analyzer. $end
  $timescale 500us $end
  $scope module logic $end
  $var wire 8 a node0 $end
  $var wire 8 b node1 $end
6 $upscope $end
  $enddefinitions $end
  #0
    b11110111 a
    b11111111 b
11 #1000
    b11110000 a
    #1500
    b11110000 b
    #2000
16 b11110001 a
    #3000
    b11110010 a
    #4000
    b11110011 a
21 #4799
    b11110011 a
    b00000000 b
```

---

## 4.5 USB-Serial converter

The third application running on the LPC1768 consists of two USB-Serial converters to forward data from the two busses to the PC. The converter emulates a virtual serial port on the host computer. On Linux systems these serial ports are located in */dev/ttyACMx* (where x is the number of the emulated port). While the first virtual serial port of the ESE board is dedicated to the *JTAG programmer*, the second and the third are assigned to the converters for *bus 0* and *bus 1*.

Both converters allow full justification of the UART communication through the ACM host driver. The user can use an arbitrary terminal program (like GTKTerm) to transmit data to or receive data from the four main nodes (the four ATmega microcontrollers). Changes of baudrate, parity and number of databits made on the host computer are simultaneously forwarded to the dedicated UART-device of the LPC1768, so there is no limitation of the UART settings. The settings made by the user are transmitted by a special *control interface* to the converter, where the UART communication is adjusted.



**Figure 4.3:** Waveform related to the example VCD-file

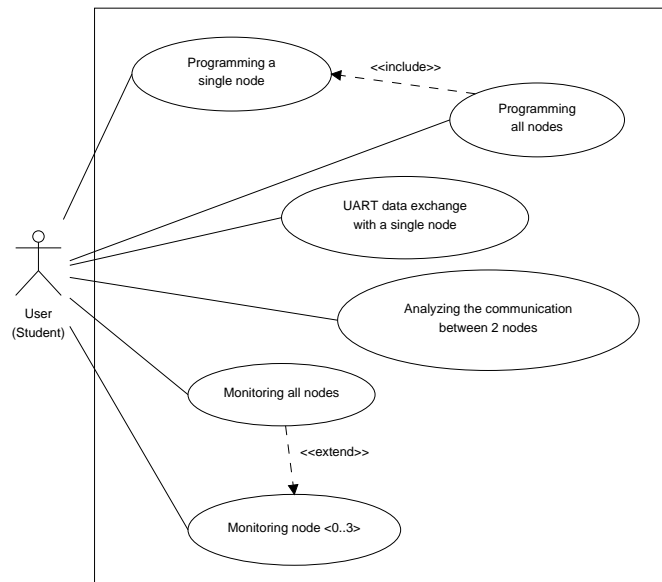


## Evaluation

This chapter includes some examples for using the software package in the ESE laboratory course. Furthermore the chapter contains a comparison between the programming time of the old and the new ESE board.

### 5.1 Usecases

Figure 5.1 illustrates some usecases for the new ESE exercise board.



**Figure 5.1:** Usecase diagram

## 5.2 Method of measurement

For evaluating the programmer (sections 5.3 and 5.4), following setup was used:

Computer	Dell Studio 1555
CPU	Intel Core 2 Duo P8600 2.4 GHz
Type	64 Bit
RAM	8 GB
OS	Ubuntu 10.10

**Table 5.1:** System configuration used for measurement

To measure the programming time, the linux command *time* was used. Listing 5.1 shows an example.

**Listing 5.1:** Example of using the time-command

---

```
> time ls
2 [...]
real    0m0.007 s
user    0m0.000 s
sys     0m0.004 s
```

---

The value *real* contains the actual execution time of the command, *user* and *sys* are the total numbers of CPU-seconds that the process spent in user respectively kernel mode. The last two values were not used in the evaluation.

## 5.3 Programming a single node

Listing 5.2 shows an example for programming *node 0* of the ESE board. To select an arbitrary node, the user has to use the '*-m<0..5>*' command. IDs 0 to 3 represent the four Atmega microcontrollers, ID 4 the ZigBee module and ID 5 the external JTAG chain. For a full list of features and a *HOWTO* for installing/using the commandline tool see [7].

Figure 5.2 illustrates the average execution time of several images. Table 5.2 contains the data used for the diagram.

image size	old board	new board
5.4kB	2.99s	0.98s
6.6kB	3.04s	2.04s
11.6kB	3.30s	2.03s
56kB	4.85s	4.84s

**Table 5.2:** Programming a single node: data used for the evaluation

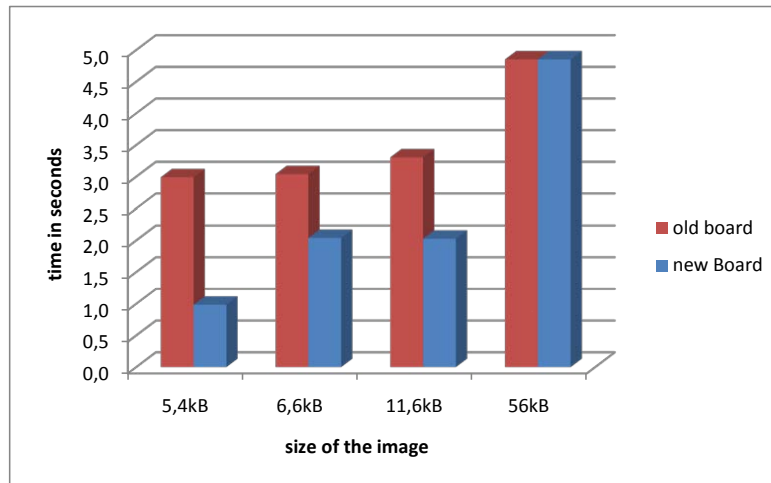
**Listing 5.2:** Programming *node 0* of the ESE board

---

```
> jtagprog -p /dev/ttyACM0 -m0 --infile=main.elf erase prog-
flash-pageload prog-eprom-bytewise
:: Checking link.... Found device:
   ESE Target, hardware version 2.0.0, firmware version 2.0.0
:: Selected node 0.
5 :: Starting blind interrogation of attached chain... found 1
   devices.
   Warning: Argument -n or --dev-number not specified,
           defaulting to 1.
:: Topology of JTAG chain is:
   Selected device is marked with "SEL:". Other devices are put
   into BYPASS mode
[SEL:atmega128]
10 :: Using configuration for >> Atmel AVR 8-bit microcontroller
   ATmega128(L) <<
   [Operation 'erase'] Performing Chip-Erase:
       Erase complete.
   [Operation 'prog-flash-pageload'] Programming flash memory:
       Processing 1 BFD sections in input image: [.text]
15   Programming complete.
   [Operation 'prog-eprom-bytewise'] Programming EEPROM
       memory:
       Processing 0 BFD sections in input image: []
       Programming complete.
```

---

As the evaluation shows, for small images the programming mechanism of the new board is about 2 – 3 times faster than the mechanism of the old board. For larger images, the execution time is almost the same. This can be explained by the static overhead of the old programming mechanism. Static overhead is for example the execution of the *JTAGZeusProg* command for multiplexing the JTAG signals. This means, that the execution time for flashing images up to a size of about 15kB is almost constant on the old board. For larger images, the time of transfer is the dominant factor of the programming process, so there is no observable difference between the two programmers.



**Figure 5.2:** Execution time of programming a single node

## 5.4 Programming all nodes

Programming all (main) nodes includes the case of programming a single node. For flashing all four Atmega microcontrollers, the *jtagprog* command has to be executed four times:

**Listing 5.3:** Programming all (main) nodes of the ESE board

---

```

> jtagprog -p /dev/ttyACM0 -m0 --infile=node0.elf erase prog-
  flash-pageload prog-EEPROM-bytewise
2 > jtagprog -p /dev/ttyACM0 -m1 --infile=node1.elf erase prog-
  flash-pageload prog-EEPROM-bytewise
> jtagprog -p /dev/ttyACM0 -m2 --infile=node2.elf erase prog-
  flash-pageload prog-EEPROM-bytewise
> jtagprog -p /dev/ttyACM0 -m3 --infile=node3.elf erase prog-
  flash-pageload prog-EEPROM-bytewise

```

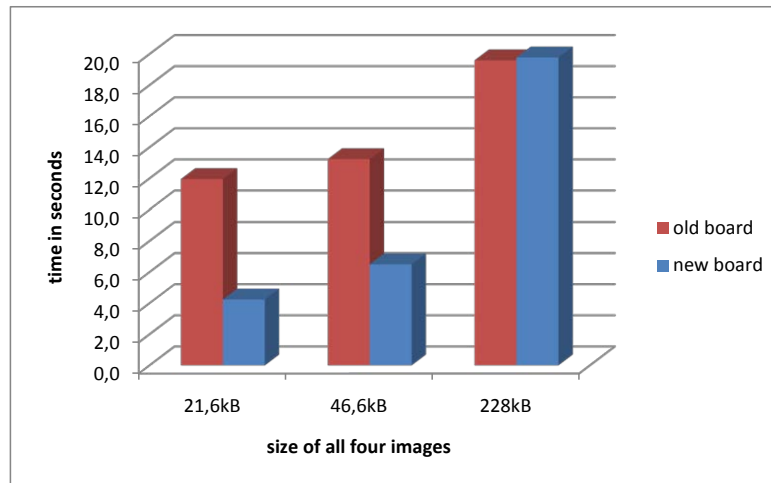
---

Following evaluation points out the difference between the old and the new board when all Atmega microcontrollers are programmed.

node 0	node 1	node 2	node 3	sum	old board	new board
5.4kB	5.4kB	5.4kB	5.4kB	21.6kB	11.96s	4.23s
11.6kB	11.6kB	11.6kB	11.6kB	46.6kB	13.24s	6.50s
56kB	59.4kB	55.5kB	57.1kB	228kB	19.59s	19.78s

**Table 5.3:** Programming all nodes: data used for the evaluation

The evaluation confirms the results of the previous section. The programming time of small



**Figure 5.3:** Execution time of programming all (main) nodes

images is twice as fast on the new board in comparison to the old board. For larger images both programming mechanisms need almost the same time.

## 5.5 Monitoring nodes

Following examples show how to use the logic analyzer. The intention of giving these examples is how the logic analyzer has to be configured with several options. They should not be seen as examples for using in real world scenarios. In most cases the trigger is not set, which means that running these examples leads to starting sampling immediately. In a real debug session, the user has to decide how many nodes he wants to investigate, the sampling interval (=accuracy), which trigger to use and how many samples he wants to store before the trigger is fired (pretrigger).

---

**Listing 5.4: LogAn examples**

---

```
1 # Enable trigger on node 2 and sample node 0 and 1 (interval 10
   us)
> LogAn -o test.vcd -n0 -n1 --trigger2 111100XX -i 10

# Sampling with 100 samples as pretrigger (interval 1ms)
> LogAn -o test.vcd -n0 -p 100 -i 1ms

6 # Use bus/device option (interval 500us)
> LogAn -d 002/015 -o test.vcd -i 500us

# Sampling special functions
11 > LogAn -o test.vcd -n0 -f
```

---

For a list of all special functions see table 4.6.

## 5.6 UART data exchange with a single node

To exchange data with a single node, the user can decide which bus he wants to use. Both busses (*bus 0* and *bus 1*) are forwarded to the pc by a virtual serial port. As mentioned before, bus 0 is mapped to the second port emulated by the ESE board and bus 1 is mapped to the third one, while the programmer is mapped to the first virtual serial port. To communicate with an arbitrary node, a terminal program like *gtkterm* or *minicom* can be used. Important for successful communication is the setting of the line coding. Baudrate, number of data- and stop bits and the mode of parity checking must be the same at the host computer and the dedicated node.

## 5.7 Analyzing the communication between two nodes

To analyze the communication between two nodes there are two possible scenarios: Either the communication is based on standard UART protocol, or the communication protocol is a dedicated 1-wire network protocol. In the first case, the user can monitor the communication by using the virtual serial port used for communication as described above. For the second case it is possible to use the logic analyzer by adding the option *'-f'* for enabling sampling special functions. According to table 4.6, bit 4 – 7 contain the node view of both busses and bit 1 and 2 of node 0 contain the view of the control device on the bus.

## 5.8 CPU utilization

To measure CPU utilization during programming, monitoring and data exchange, the firmware was expanded. To quantify the CPU utilization, the idle hook functionality of FreeRTOS was used. The function *vApplicationIdleHook* gets called every time the scheduler executes the idle task. The function increases an integer variable. An additional task, which is executed in equidistant time intervals, sends the actual value of the variable to the computer. Finally the variable is set back to zero.

The arithmetic average of this variable ( $V_0$ ) for the time the CPU is in idle mode is mapped to 0%. To calculate the actual CPU utilization, following formula has to be applied ( $V$  is the arithmetic mean of the variable when executing some operations):

$$U_{CPU} = \frac{V_0 - V}{V_0}$$

According to this method, the average CPU utilization for all three applications was calculated. Table 5.4 contains the gained results.

programming	monitoring	uart communication
18.53%	40.02%	39.38%

**Table 5.4:** Average CPU utilization

## Conclusion

This section points out the enhancements affecting the ESE laboratory course and the possibilities for future upgrades of the firmware and the host software.

### 6.1 Achieved improvements

The new exercise board enables a wider range of exercises for the ESE laboratory course. Contemporary technologies (like the graphical LCD or the ZigBee module) and comfortable features (like the logic analyzer) adapt the course to recent trends.

The use of the LPC1768 enables native USB communication and therefore reduces complexity of the hardware design. The embedded logic analyzer allows easy debugging of code. The user can decide if he wants to observe normal IO pins or special functions, which is helpful for implementing communication protocols or device drivers for the LCD, the LED-matrix or the onboard fan. The included USB-Serial converter for the two 1-wire-busses allows communication between the host computer and each of the nodes.

### 6.2 Outlook

Both, the firmware and the software for the host computer are written modular, so future add-ons can be written, enhancing the features of the board. The commandline tool of the logic analyzer generates VCD-files of the recorded samples, which then can be observed by waveform viewers like GTKWave. With the help of abstract C++ classes and inheritance it is possible to extend the commandline tool for further output formats. It is even possible writing an output module for storing the samples in a database, or writing a graphical user interface for directly viewing waveforms. Another upgrade of the logic analyzer could be adding the detection of rising and/or falling edge triggers. But this has to be done carefully, because this detection is quite complex and the interrupt routine for sampling the IO pins should retain short to preserve accuracy.



As mentioned before, the ability for dynamically changing the version of the JTAG programmer is disabled in the ported version of the firmware. Another future update may be adding this feature by using In-Application Programming (IAP, see [6]), which stores data directly in the internal flash memory of the LPC1768.

Furthermore it is possible to extend the abilities of the board by new applications. Since the firmware is based on an embedded operating system, it is not complex to add new tasks, achieving new functions. For example some of the dedicated IO pins of the logic analyzer could be used in an alternative way for these new functions.

## Using the Logic Analyzer

The appendix contains a user manual for the logic analyzer. After a step-by-step guide for installing the commandline tool (*LogAn*), possible arguments and settings are explained. The appendix ends with usage information for *GTKWave*.

### A.1 Setup

Following commands have to be executed from the *LogAn*-directory to install the commandline tool of the logic analyzer:

**Listing A.1:** Installing the LogAn commandline tool

---

```
# configure the makefile
> ./configure

4 # compile source
> make

# install binary, udev-rule and man-page
> make install
```

---

Attention: To use the commandline tool, the libusb library [5] must be installed.

## udev rule

In order grant read/write access for the ESE board to non root-users, the setup tool adds following udev rule to */etc/udev/rules.d/e.se.rules*

---

**Listing A.2:** udev rule for the ESE board

---

```
SUBSYSTEM=="usb", ENV{DEVTYPE}=="usb_device",SYSFS{idVendor}=="  
ffff", SYSFS{idProduct}=="1234", MODE="0666"
```

---

## Uninstalling

To uninstall the commandline tool following command has to be executed:

---

**Listing A.3:** Installing the LogAn commandline tool

---

```
# uninstall binary, udev-rule and man-page  
> make uninstall
```

---

## A.2 Commandline tool

### Synopsis

Listing A.4 shows the usage of the LogAn commandline tool. The only mandatory option is '-o' to specify the output file.

---

**Listing A.4:** LogAn synopsis

---

```
LogAn -o FILE [options]
```

---

## Options

For using the logic analyzer several options are provided. Following list describes all options that can be used:

- *-o, - -outfile <output file>:*  
Specifies the output file. The output format is VCD (Value Change Dump)
- *-h, - -help:*  
Displays usage information.
- *-d, - -device <bus/device>:*  
Specifies the bus and device to which the ESE board is attached (example: 002/015)
- *-n, - -node <0..3>:*  
Enables the sampling of the signals of the selected node. If not set at least once, all nodes are activated.
- *- -trigger<0..3> <00000000..11111111 - X for don't care>:*  
Sets the trigger for the specified node.
- *-p, - -pretrigger <number of samples>:*  
Sets the number of samples to store before the trigger gets fired.
- *-i, - -interval <interval>[mlu]:*  
Specifies the sampling interval in  $\mu s$  (default) or *ms*.
- *-f, - -special-functions:*  
Samples special functions instead of IO pins. Special functions are for example: Fan PWM, LCD Backlight PWM, Bus0/1, etc.
- *-v:*  
Increases the verbosity level by one. Multiple occurrences of this option add up to a maximum verbosity level of three. In verbose mode, LogAn prints debug output to stdout, depending on the verbosity level:
  - Level 1: Output USB connection information
  - Level 2: Additionally print basic communication information
  - Level 3: Print even more information
- *- -version:*  
Displays the program version.

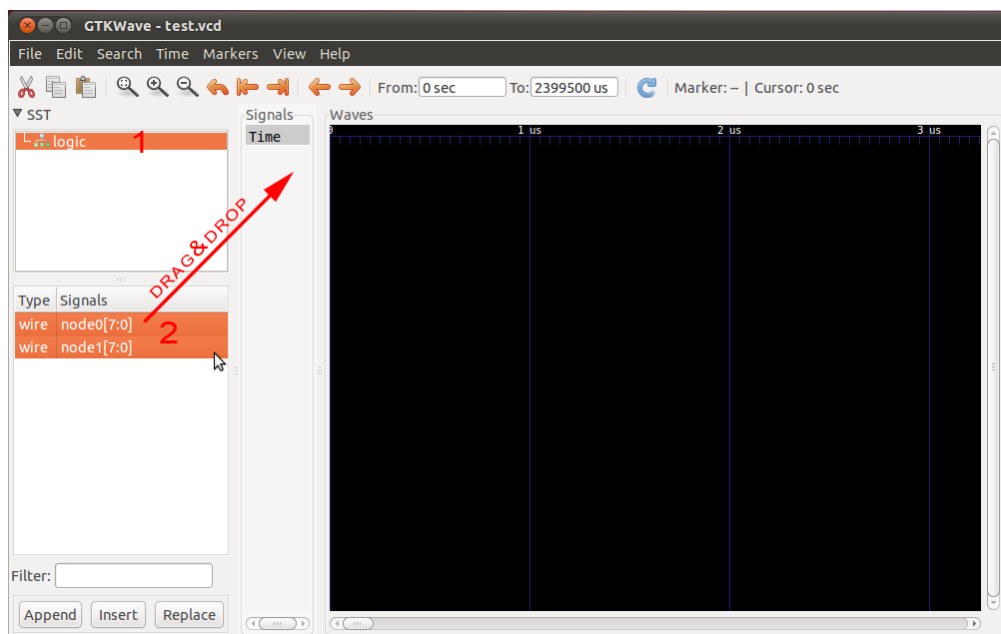
## A.3 GTKWave

After a successful run of the logic analyzer, the commandline tool generates a VCD file. To visualize this file, GTKWave can be used (see listing A.5). After the waveform viewer has started, the user can move signals to the timeline (see figure A.1 and A.2).

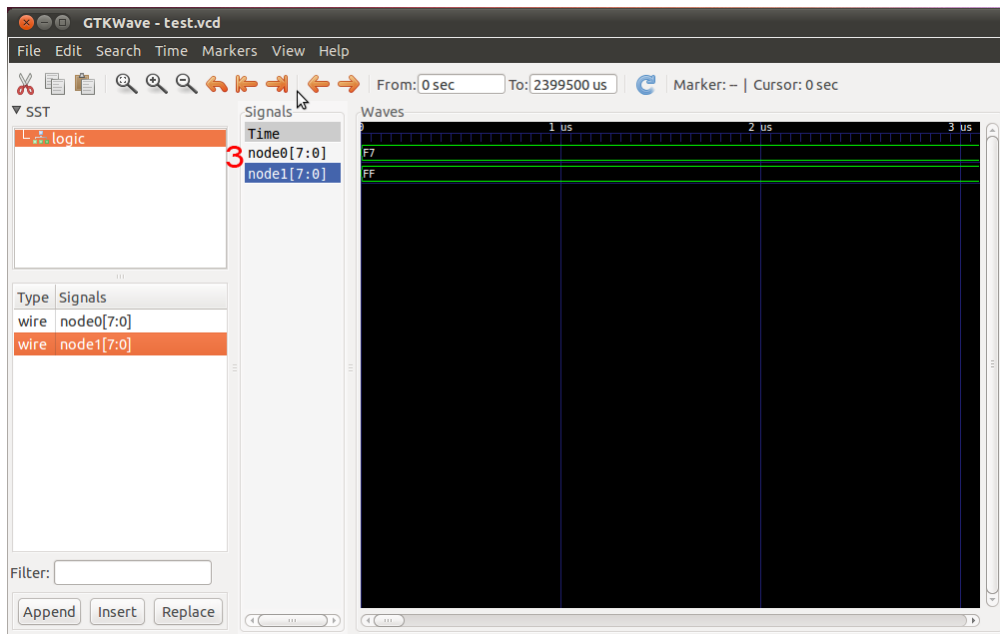
### Listing A.5: Using GTKWave

```
> gtkwave test.vcd
```

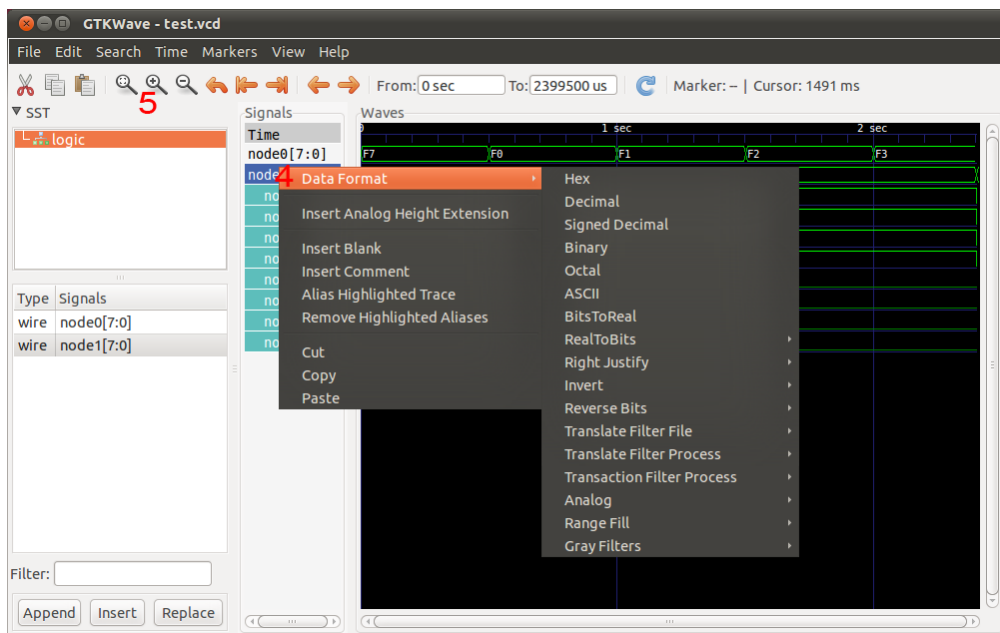
Following screenshots illustrate the workflow of using GTKWave.




**Figure A.1:** GTKWave: Adding signals to timeline 1



**Figure A.2:** GTKWave: Adding signals to timeline 2



**Figure A.3:** GTKWave: Setting data format (4) and zoom options (5)

If the VCD file is changed, the view can be refreshed by the shortcut  $[Ctrl] + [Shift] + [R]$  or pressing the reload button .

# Bibliography

- [1] Richard Barry. Freertos. <http://www.freertos.org/>. Accessed: 2011-06-04.
- [2] Tony Bybell. Gtkwave. <http://gtkwave.sourceforge.net/>. Accessed: 2011-06-04.
- [3] Craig Peacock (Beyond Logic). Usb in a nutshell. <http://www.beyondlogic.org/usbnutshell/usb1.shtml>. Accessed: 2011-06-04.
- [4] M. H. Rashid Fang Lin Luo, Hong Ye. *Digital power electronics and applications*. Elsevier Academic Press, 2005.
- [5] libusb. libusb. <http://www.libusb.org/>. Accessed: 2011-06-04.
- [6] NXP. *LPC1768 User Manual*. [http://www.nxp.com/documents/user\\_manual/UM10360.pdf](http://www.nxp.com/documents/user_manual/UM10360.pdf), 2010.
- [7] Martin Schmölzer. A modular, xml-based jtag programmer for embedded devices, 2008.
- [8] Bertrik Sikken. lpcusb. <http://sourceforge.net/projects/lpcusb/develop>. Accessed: 2011-06-04.
- [9] Arvid Staub. A teaching platform for embedded systems engineering, 2011.