GPS Bycicle Computer

Armin Schlegel,
Christian Eismann

July 24, 2011

Contents

List of Figures

Abstract

This document contains the description and API specification of the project *GPS Bicycle computer*. The target of this project was to create an electronic mobile device that provides several informations on an graphic display. The actual information is determined by analyzing different data sets provided by a GPS receiver. Further more an SD card controller enables data recording on a SD card. An overview of the general functionality is shown in figure 1.

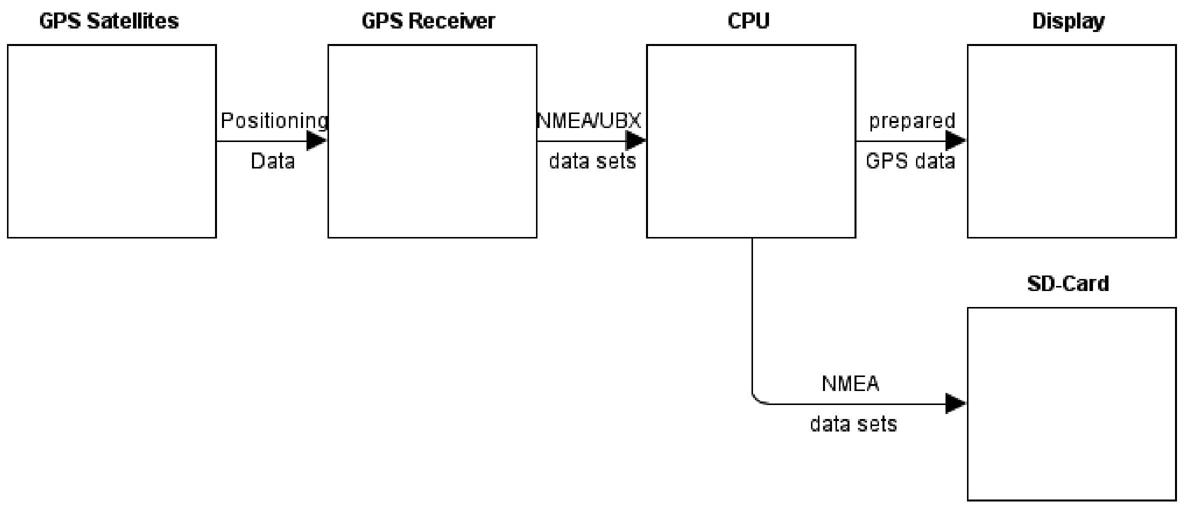


Figure 1: General functionality

1. Introduction

In this chapter the basic features (must-have features) are introduced.

1.1. The main features

- **Receiving and processing of GPS data**
  The information that is provided to the user is determined by analyzing received GPS data sets. Information about latitude/longitude, data time and several more can be provided.
- **Visualization of GPS data on a graphic display**
  The already mentioned data is then presented on a graphic display to the actual user.
- **Storing of received data**
  The data provided by the GPS receiver can be recorded to a SD card. In further processing this can be used, for example, to analyze a road trip with Google Earth.
- **Charging electronics**
  The charging electronics is required for recharging used batteries.

2. The toolchain

In this chapter all tools that have been used during this project are described. Furthermore tools that could have been used alternatively are also noted.

2.1. Hardware development tools

For the hardware development, that means circuit board design and layout, only EAGLE from Cadsoft has been used.

2.2. Software development tools

The main target was to assemble a toolchain containing open-source tools only. Especially for the AVR ATmega series a lot of open source alternatives to the AVR Studio IDE exists. By using these tools a mostly automatic toolchain can be assembled. Another possibility would be using eclipse with a special AVR plugin (`The AVR Eclipse Plugin`).

2.2.1. Oracle Virtualbox

The base for the toolchain used within this project is Linux. However, doing software development on a Linux distribution involves several complications in most cases. An appropriate alternative would be CygWin, an open source terminal for Windows that emulates the Linux API. Nearly all Linux standard tools are supported by CygWin, such as grep, find, gcc, gdb, ... But obviously it is dependent on the Windows version in which it is installed on. For example Windows 7 is not fully supported yet. So for implementing a Linux based toolchain a virtual machine is used, that can be simply copied to any PC for further developing. Virtualbox from Oracle is the best choice in this context, because it is freeware and, in opposite to the VM Player from VMware, virtual machine images can be created. Furthermore it supports access to the file system of the parent OS (e.g. Windows). So it is very easy to share data between both operating systems. The Linux distribution running on the machine is a Debian image (pretty small, but contains all necessary tools). By using the apt-get shell functionality further tool installing is very comfortable. (e.g. for installing grep, simply type: apt-get install grep).

2.2.2. The GNU Compiler Collection (AVR-GCC)

The main component of this toolchain is the AVR-GCC, containing compiler, linker, debugger and so on. The AVR-GCC is a port of the original GNU GCC, so there is a well reviewed documentation available for all tools. Even a ported version of the LIBC is available for AVR microcontroller development. The free compiler suit is one of the biggest advantages of AVR microcontrollers for projects like the GPS bycicle computer. Actually compilers for microcontrollers from other vendors are mostly very expensive (e.g. CodeWarrior from Freescale).

2.2.3. The Programmer: avrdude

To actually program the target avrdude is used. This tool is also open source and freeware. Furthermore, like all other previous mentioned Linux tools, it can be called via Makefiles. Avrdude supports various hardware programmers.

2.2.4. Make

Make is **the** standard tool for managing and building binaries and libraries from source code.

2.2.5. (Sp)lint

Splint is a static code analysis tool for the C Programming Language. In this project it has been used for advanced failure tracking. But obviously this tool reports too many positive faults for continuous usage within a project. Furthermore the tool actually explain the user how to disable certain warnings. Therefore it is even in a professional project not recommendable.
Instead of using splint any further we decided to extend the warning options of the AVR-GCC compiler.

Listing 1: GCC compiler warnings

```
# GCC compiler warnings
CWARN = -ffreestanding -pedantic -Wall -Wextra -Winit-self -Wswitch-default -Wunused-parameter -Wunknown-pragmas -Wstrict-overflow=1 -Warray-bounds -Wfloat-equal -Wdeclaration-after-statement -Wundef -Wno-endif-labels -Wshadow -Wbad-function-cast -Wcast-qual -Wcast-align -Wwrite-strings -Wstrict-prototypes -Wmissing-prototypes -Wmissing-declarations -Wredundant-decls -Wnested-externs -Wvla -Wvolatile-register-var -Wparentheses -g -Os -fno-strict-aliasing
```

3. Hardware development

Before we began with the hardware design it became necessary to analyse the components we needed.

3.1. Requirement analysis and pre-selection of hardware components

Component	**Details**	**Why this component?**	**Comment**
ATMega32-L	8-Bit, **3.3V**, 32kB Flash, 2kB SRAM, 4kB EEPROM	Runs on 3.3V,easy to program via GNU GCC and other GNU Tools	Became obsolete due a lack of RAM
ATMega664-L	Pin compatible to the ATMega32-L, 64kB Flash, **4kB SRAM**	Has 2kB more RAM than the old CPU needed by the application	Used as CPU after lack of RAM
NL-552ETTL	GPS-Receiver, 5V supply, **RS232** communication with **3.3V TTL**	Supports NMEA, supports european GALLILEO sattelites, 3.3V levels makes level shifting unnecessary	
MAX3221	**3.3V**, RS232 driver	e.g. for debugging on a PC	Never used
LM 2940	5V Low Dropout Regulator	Can use min. 5.5V to generate 5.0V	Discharges the batteries to 5.5V

LT1117-3.3	**3.3V** Drouput Regulator	3.3V needed for CPU, SD-Card, Display, RS232 Driver	
EA-DOGL128W, EA-LED68x51-W, EA-TOUCH128-2	**3.3V**, **SPI**, 128x64 b/w, white backgroundlight, touch	Cheap, easy to use via SPI, backlight and touchscreen available	Consumes approx. 80mA
Yamaichi FPS009-2305	Slot for SD-Cards, can be read out if card is present and writeable or locked	It was the cheapest	
LEDs		Some user LEDs	
JTAG		programming and debugging	
ISP		programming	

3.2. Layout and design

Because of the lack of time we jumped over a first prototype board and went directly to the attempt to create the final prototype hardware. We wanted the project to be compact and portable so the first thought was to bring the complete hardware design down to SMD components. In second thoughts to make the device tiny and portable, a compact design is needed. So we decided to put one circuit board on top of another connected via plug connectors.

As the project contains a graphical display its obvious that it has to be placed on top. Because the display and the SD-card communicates with the processor via SPI it is convenient to combine both on one circuit board. It is useful to place the user LEDs on the top too. This board is called the “displayboard” and is connected on top of the “mainboard”.

The “mainboard” contains the rest of the components like CPU, JTAG and ISP connectors, voltage regulators, etc. In order to get the device mobile there are some batteries needed. We chose five AAA 1.2V rechargeable batteries because they are easy to recharge. If all batteries are completely charged (at 1.4V each) the resulting voltage is 7.0V. This is the maximum Voltage the batteries can deliver. At the minimum of 5.5V (1.1V each battery) the 5V Regulator quits and the batteries won’t discharge further. This will prevent the batteries from total discharge that would cause severe damage to the batteries.

To get the device stay longer alive we thought about a opportunity to charge the batteries while driving. For example this could be done hub generator. In order to charge the batteries to the maximum of 7V a regulator is needed that generates 7V out of the hub generator voltage. Therefore a “supplyboard” was built.

The “supplyboard” contains a combined step-up and step-down regulator (SEPIC) in order to generate the needed 7 V out of the voltage coming from the hub dynamo with the range of 4 to 40 Volt AC.

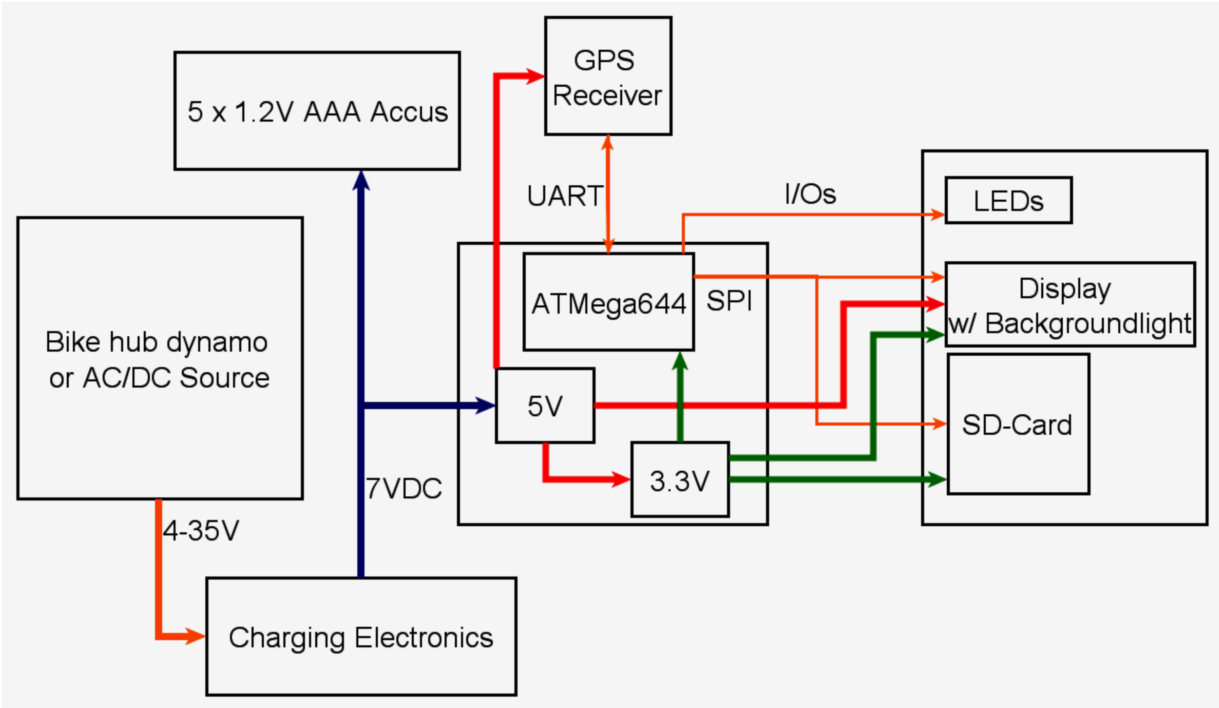


Figure 2: Hardware Block Diagram

Within figure 2 the hardware components of the entire product and the relations between them is shown.

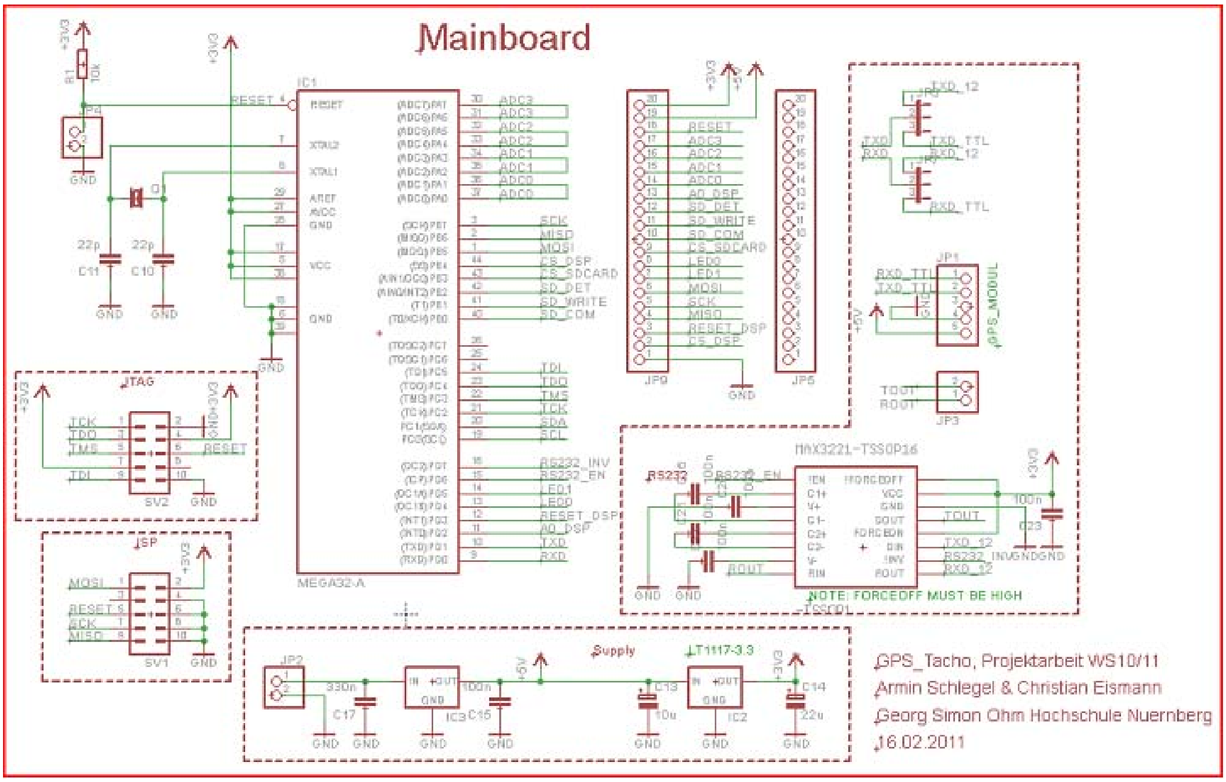


Figure 3: Mainboard schematics

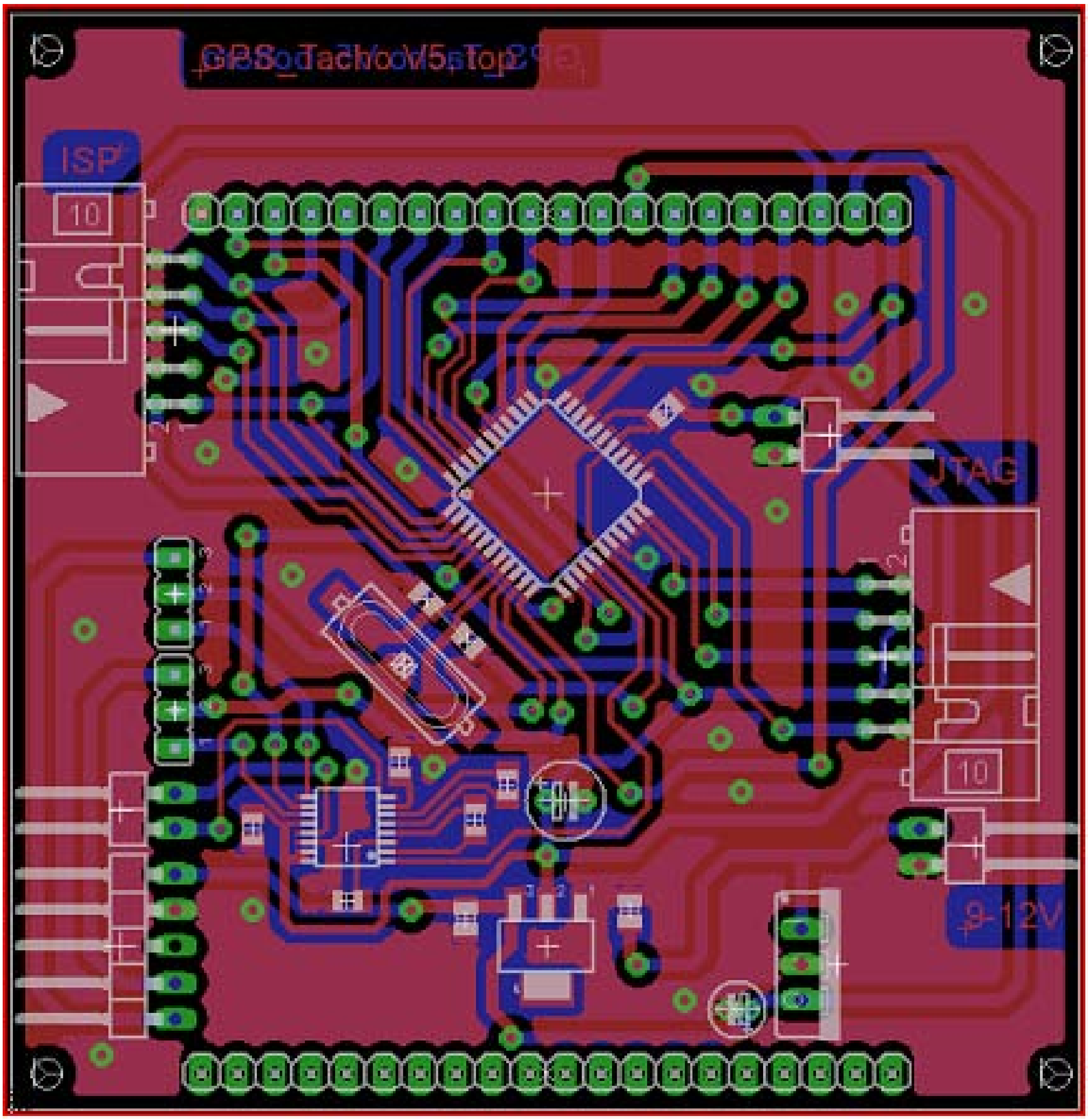


Figure 4: General functionality

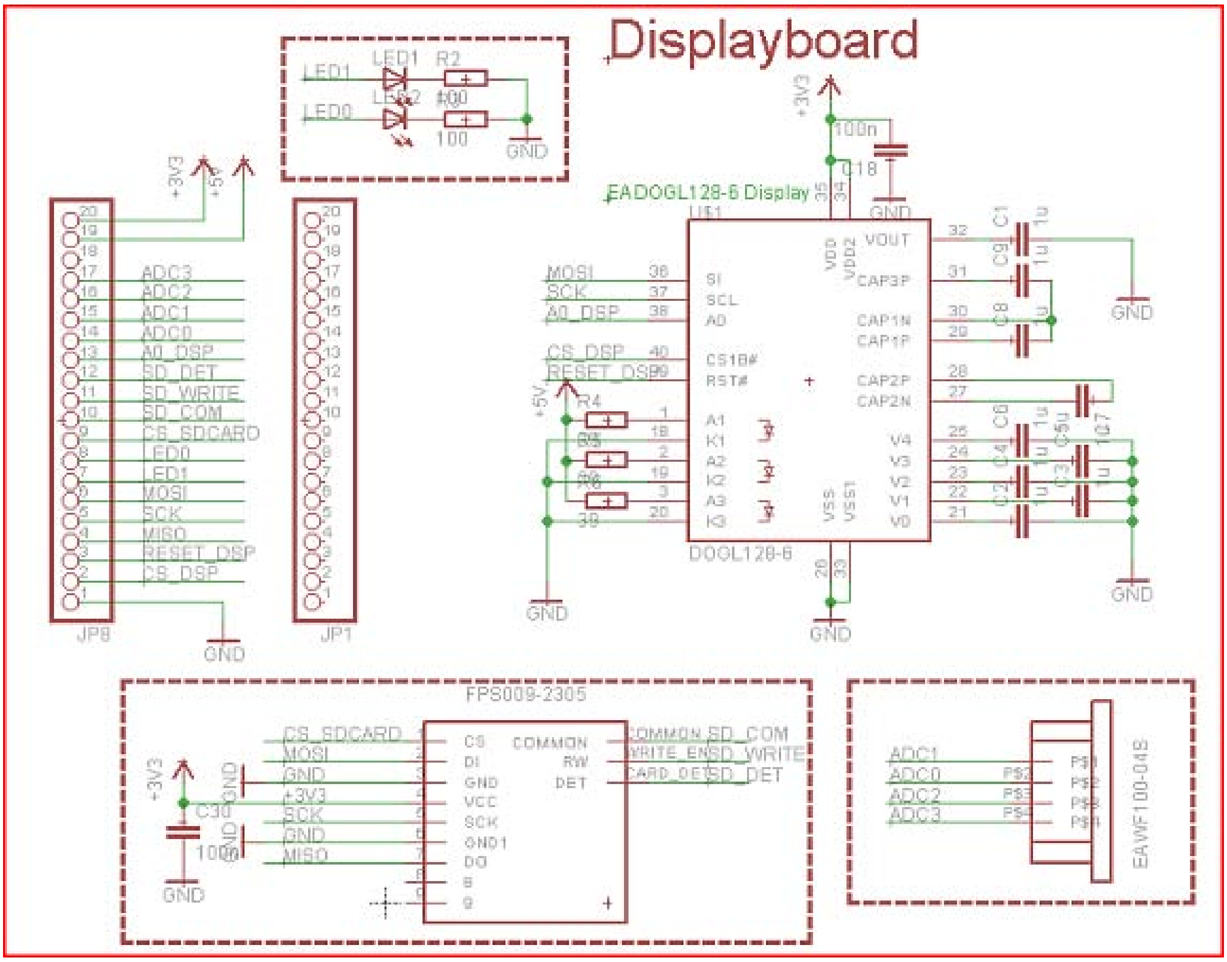


Figure 5: Displayboard schematics

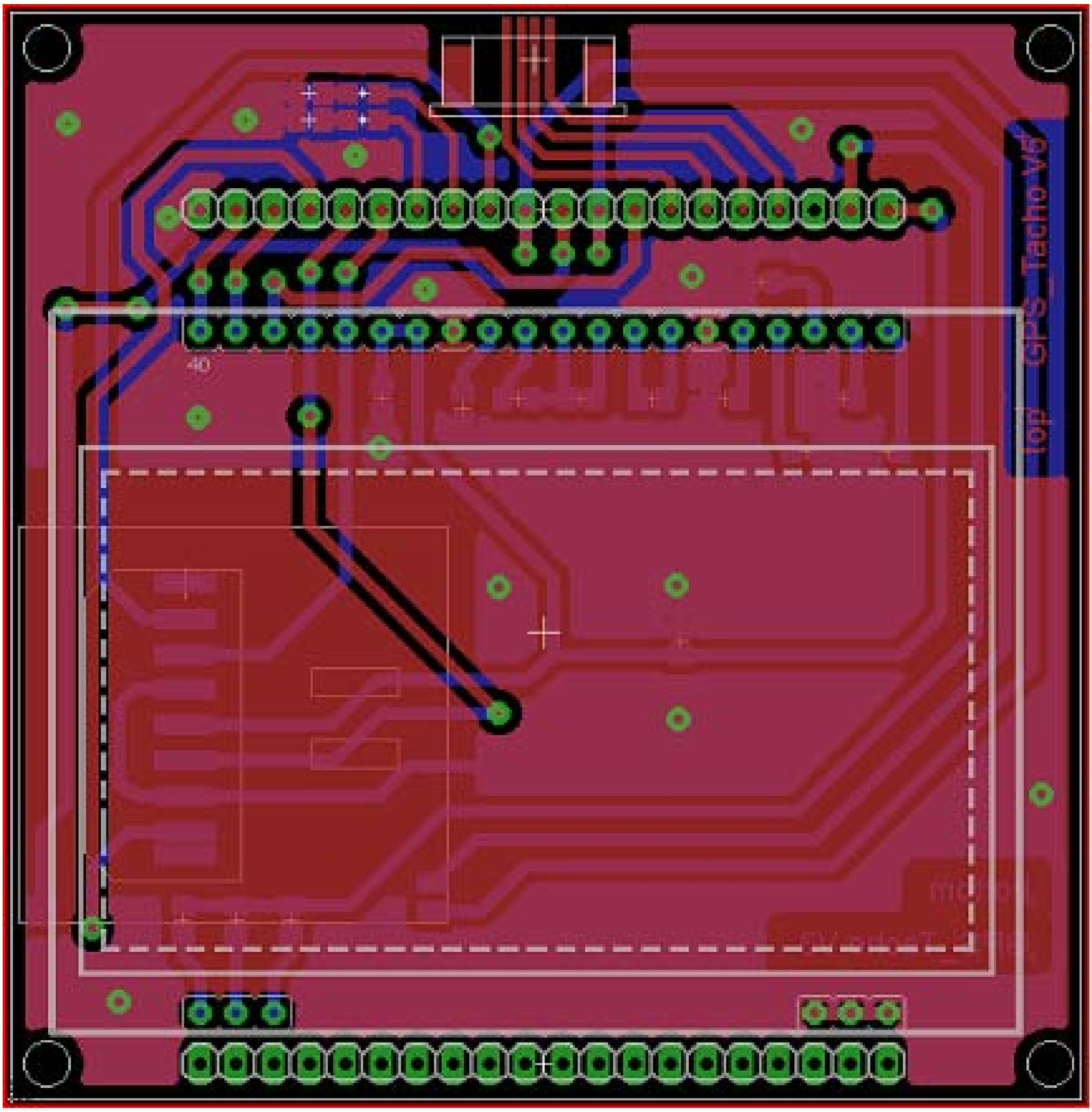


Figure 6: Displayboard layout

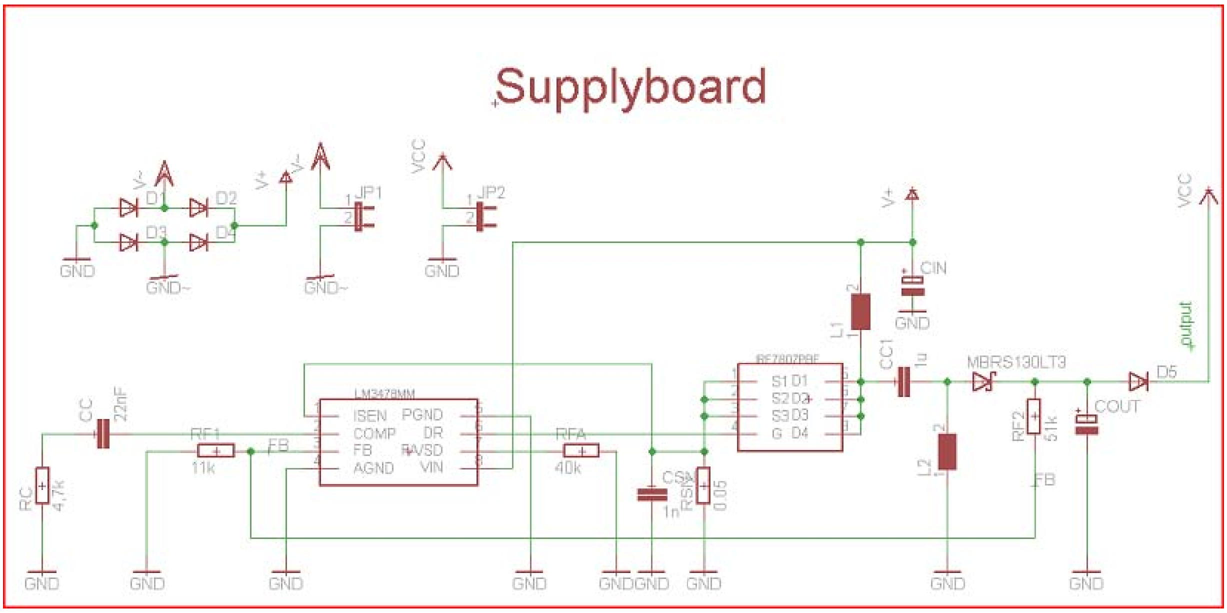


Figure 7: Supplyboard schematics

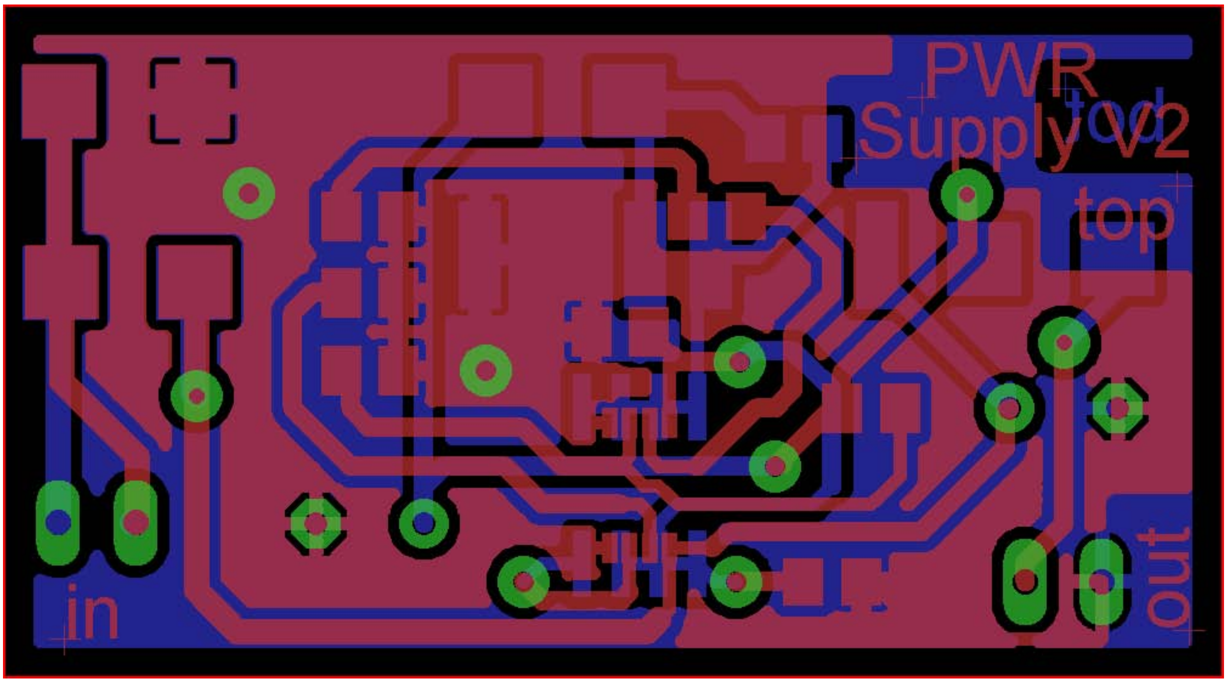


Figure 8: Supplyboard layout

4. Software development

In this chapter the software development which was done during this project is described. In addition to that all API functions are mentioned in the following sections.

4.1. Software Pre-Analysis

For transmitting GPS data from the receiver to any microcontroller (or any other processing unit) the ``National Marine Electronics Association" (NMEA) protocol is used. NMEA is an ASCII based protocol, used for transmitting character frames from one participant to another. The general structure of a NMEA frame is shown in figure 9.

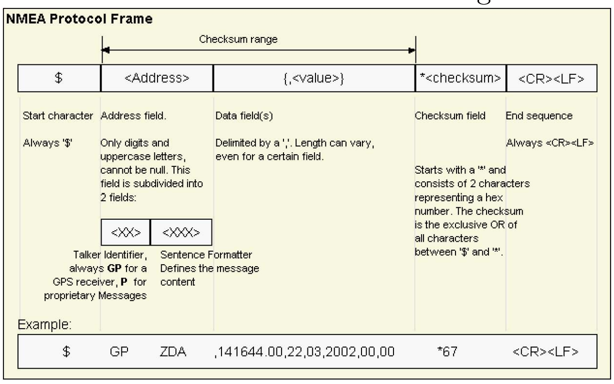


Figure 9: NMEA protocol structure

For the configuration of the GPS receiver UBX (acronym for U-Blox) protocol messages are used. The proprietary UBX format is shown in figure 10.

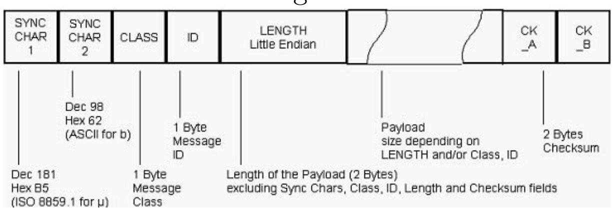


Figure 10: UBX protocol structure

For further detailed information about these protocols please refer to the according data sheet of the U-Blox 5.

4.2. General design

General spoken a project providing a more or less complex user interaction via a graphical touch display usually requires the implementation of an Operating System. But obviously this work would be too time consuming for such a small project like this. So we decided to abandon on extended touch screen functionality in a first step. As consequence the general workflow is kept as simple as possible.

4.3. Program workflow

After initializing every component like spi, display driver, uart and others the central part of the program control is an infnite while loop within the main routine. The actual application is triggered on every `SIG_USART_RECV` interrupt request. This IRQ indicates that the GPS receiver has sent new data to the microcontroller via UART. Figure 8 visualizes the basic functionality: most of the runtime the microcontroller is waiting for incoming data (`SIG_USART_RECV`). Every time the interrupt is thrown one byte has been read from the UART interface as long as all needed GPS blocks are completely received. This is monitored with the aid of the identifier tags of the NMEA tags like `$GPRMC` or `$GMVTG`. The received data is stored within a separate buffer (`char uart_str[]`) and then split up to its separate components (`RMC, VTG and GGA` data sets). If all conditions for recording are met (at least 3 sattelites, SD card present and writeable) the user can start a record via the touchscreen. If the user set the device to record the whole `uart_str[]` buffer is stored on a SD Card. In order to display the correct data, some calculations are done. Those calculations where for example to convert the `NMEA` format of longitude and latitude in a human readable format or correct the view of the time. After all the wanted information is being displayed.

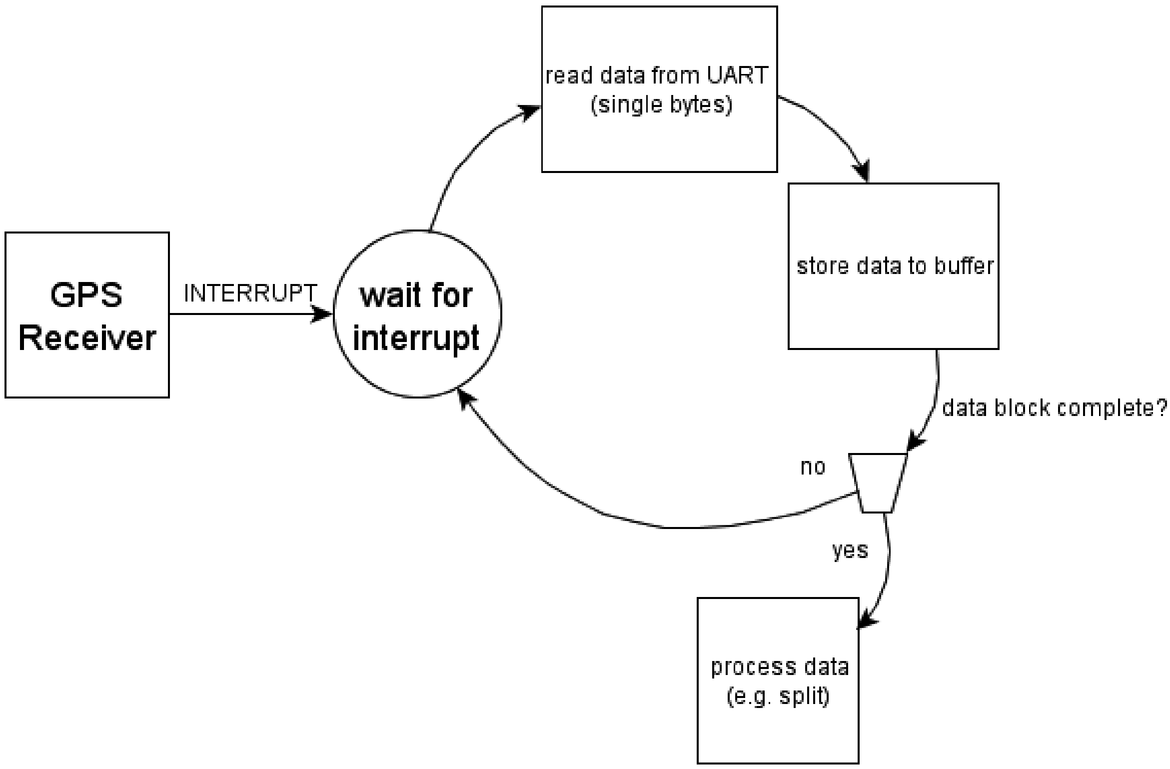


Figure 11: General workflow overview

4.4. Modules

This section provides an overview about the functionality within the separate software modules. The API function documentation can be found in the delivered tar-ball (generated by using doxygen).

4.4.1. Display

As described in the data sheet of the graphic display, the data transmission between the display and the CPU is unidirectional. In consequence to this, the current set pixels have to be stored in CPUs SRAM to avoid loosing the actual frame information.

The *Display* module provides the following features:

- **Sending data to the display**
  This functionality is mandatory for the general handling of the graphic display. The implemented methods provides an interface for sending commands (e.g. turn display on) and data (e.g. an entire frame) via `SPI`.

- **Initialization of the graphic display**
  The actual hardware initialization. As described in the data sheet of the EA DOGL 128L ([EADOGL08]) different preferences have to be configured before using the display, such as display contrast, starting line, etc.

- **Writing text**
  The byte representation of every single character is stored within the `Font5x7[]` array (in EEPROM). To write text to the display the index of the required character is determined. Taken the data from that array and sending it to the display by using the `display_putpixel()` function results in a printed character on the graphical display.

- **Painting pictures**
  For painting a picture e.g. a bitmap file has to be converted to a representing byte array. The provided tool “GrafikKonverterV1_1” provides this functionality.

As an example for the internal pixel representation the following listing shows the sending of an entire frame to the display controller.

Listing 2: Sending a frame to the display controller

```c
/**
 * Send a new frame to th ST-7565R display RAM
 *
 * A frame consists of 8 Pages. Each page consists of 8x128bit storage.
 *    0                                              127
 *    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-...-+-+-+
 * D0 | | | | | | | | | | | | | | | | | | | ... | | |
 *    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-...-+-+-+
 * D1 | | | | | | | | | | | | | | | | | | | ... | | |
 *    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-...-+-+-+
 * D2 | | | | | | | | | | | | | | | | | | | ... | | |
 *    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-...-+-+-+
 * ...
 *    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-...-+-+-+
 * D7 | | | | | | | | | | | | | | | | | | | ... | | |
 *    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-...-+-+-+
 * The ST-7565R display controller provides a 64 * 132(!) bit RAM. But
 * the actual display resolution of the EA DOGL128-6 is 128x64 pixel.
 * Therefore only 128 columns are used within the internal RAM of the
 * display controller.
 * The virtual data representation is realized in linear order here.
 *    |- disp_ram[0]                 disp_ram[1023] -|
 *    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-...-+-+-+
 *    | | | | | | | | | | | | | | | | | | | ... | | |
 *    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-...-+-+-+
 * To avoid a waste of memory the data structure is organized as follows:
 * ARRAY_ELEMENT = 8bit * COLUMN + PAGE --> maximum array size of
 * 1024 elements
 *
 * Every 8 bit value stored in disp_ram array represents a single column
 * of one page.
 */
void display_send_frame()
{
   uint8_t page;
   uint8_t column;

   for (page = 0U; page < 8U; page++) {
      display_go_to(0U, page);
      for (column = 0U; column < 128U; column++)
         display_send_data(disp_ram[page + (column << 3U)]);
   }
}
```

4.4.2. GPS

Within this module the actual resolving of GPS data sets is realized. Furthermore the storage of GPS data to the SD card is implemented. The central functionality of the GPS module is the actual splitting of GPS data sets into separate tokens. This is done in function `gps_split_data()`. First the type of the current received data set is determined. For each different type a separate splitting function is implemented (`gps_split_gga(), gps_split_rmc(), gps_split_vtg()`. Within these functions a data set is split into its tokens, separated by "," or "*". In a next step these tokens are stored in the according data buffer (`rmc[], gga[] or vtg[]`).

As shown in listing 2 every single data component of a RMC, GGA or VTG data set is separated by a single comma. Only the checksum at the end of the NMEA frame is separated by a "*".

Listing 3: Example of NMEA data sets

```
$GPRMC,125449.00,A,4928.05209,N,01105.07684,E,3.153,252.34,260311,,,A*61
$GPVTG,252.34,T,,M,3.153,N,5.840,K,A*32
$GPGGA,125449.00,4928.05209,N,01105.07684,E,1,03,2.82,349.7,M,47.0,M,,*53
$GPRMC,125450.00,A,4928.05204,N,01105.07576,E,2.242,,260311,,,A*74
$GPVTG,,T,,M,2.242,N,4.152,K,A*27
$GPGGA,125450.00,4928.05204,N,01105.07576,E,1,03,2.82,349.7,M,47.0,M,,*58
```

The static PUBX messages used for initializing the GPS receiver are stored within EEPROM to avoid a waste of RAM memory.

Listing 4: Static GPS initialization PUBX commands

```
/** The byte code for setting the baudrate of the GPS receiver */
static char EEMEM baud [] = {0xB5,0x62,0x06,0x00,0x14,0x00,0x01,0x00,
                             0x00,0x00,0xD0,0x08,0x00,0x00,0x00,0x4B,
                             0x00,0x00,0x03,0x00,0x02,0x00,0x00,0x00,
                             0x00,0x00,0x43,0x31};

/** turn on VTG */
static char EEMEM vtg_on[] = {"$PUBX,40,VTG,0,1,0,0*5F\r\n"};
/** turn on RMC */
static char EEMEM rmc_on[] = {"$PUBX,40,RMC,0,1,0,0*46\r\n"};
/** turn on GGA */
static char EEMEM gga_on[] = {"$PUBX,40,GGA,0,1,0,0*5B\r\n"};
/** turn off GSA */
static char EEMEM gsa_off[] = {"$PUBX,40,GSA,0,0,0,0*4E\r\n"};
/** turn off GRS */
static char EEMEM grs_off[] = {"$PUBX,40,GRS,0,0,0,0*5D\r\n"};
/** turn off GSV */
static char EEMEM gsv_off[] = {"$PUBX,40,GSV,0,0,0,0*59\r\n"};
/** turn off ZDA */
static char EEMEM zda_off[] = {"$PUBX,40,ZDA,0,0,0,0*44\r\n"};
/** turn off GST */
static char EEMEM gst_off[] = {"$PUBX,40,GST,0,0,0,0*5B\r\n"};
/** turn off GLL */
static char EEMEM gll_off[] = {"$PUBX,40,GLL,0,0,0,0*5C\r\n"};
```

4.4.3. Interrupt

In this module the main interrupt routine is implemented, that is triggered as soon as a new byte has been received via UART from the GPS receiver. A detailed visualization of this functionality is shown in figure 11.

4.4.4. SPI

In this module the SPI driver is implemented. This includes the initialization (`spi_init()`), the sending of single bytes (`spi_send()`) and the reading of single bytes from the SPI (`spi_read()`).

4.4.5. UART

This module realizes the serial data communication between the GPS receiver unit and the microcontroller. This includes initialization of UART and receiving/sending of single bytes. One part of the initialization is the setting of the baudrate to 38400 (`uart_init_high()`). If the GPS receiver has still its default settings the communication between the microcontroller and the GPS receiver is only possible with a baudrate of 38400. After initializing the GPS receiver and setting its baudrate to 9600, the baudrate of the UART has to be set to 9600 too in order to keep up the communication. (`uart_init_low()`).

The high baudrate of 38400 is actually not necessary for the general communication between GPS receiver and microcontroller, because the processing time (data processing, display output, SD output) is so short, that in this case a lower baudrate is preferred. Another reason to have a low baudrate is that it's also applicable for communication between microcontroller and a PC (high baudrate causes short transmission range).

4.5. SDC/Fat16

The SDC/Fat16 modules are external libraries. Only in SDC some IO functionality has been adapted.

4.6. The application

The actual application is placed within the s.c. "Windows" module. Here the already prepared GPS data is converted into the different standardized representations. In function `window1()` the actual display output is controlled. Several helper functions provides easy access to the corresponding representations of the GPS data. The `window1()` function is triggered by every incoming `SIG_USART_RECV` (see also `interrupt.c`) interrupt, as soon as a data block has been completly processed.

5. User Guide

5.1. Turn on power supply.

The "Power ON/OFF" switch can be found on the right side of the product. Figure 12 shows that switch and the SD card slot.

Figure 12: Power ON/OFF Switch

5.2. Display layout

The general layout of the display is shown in figure 13.

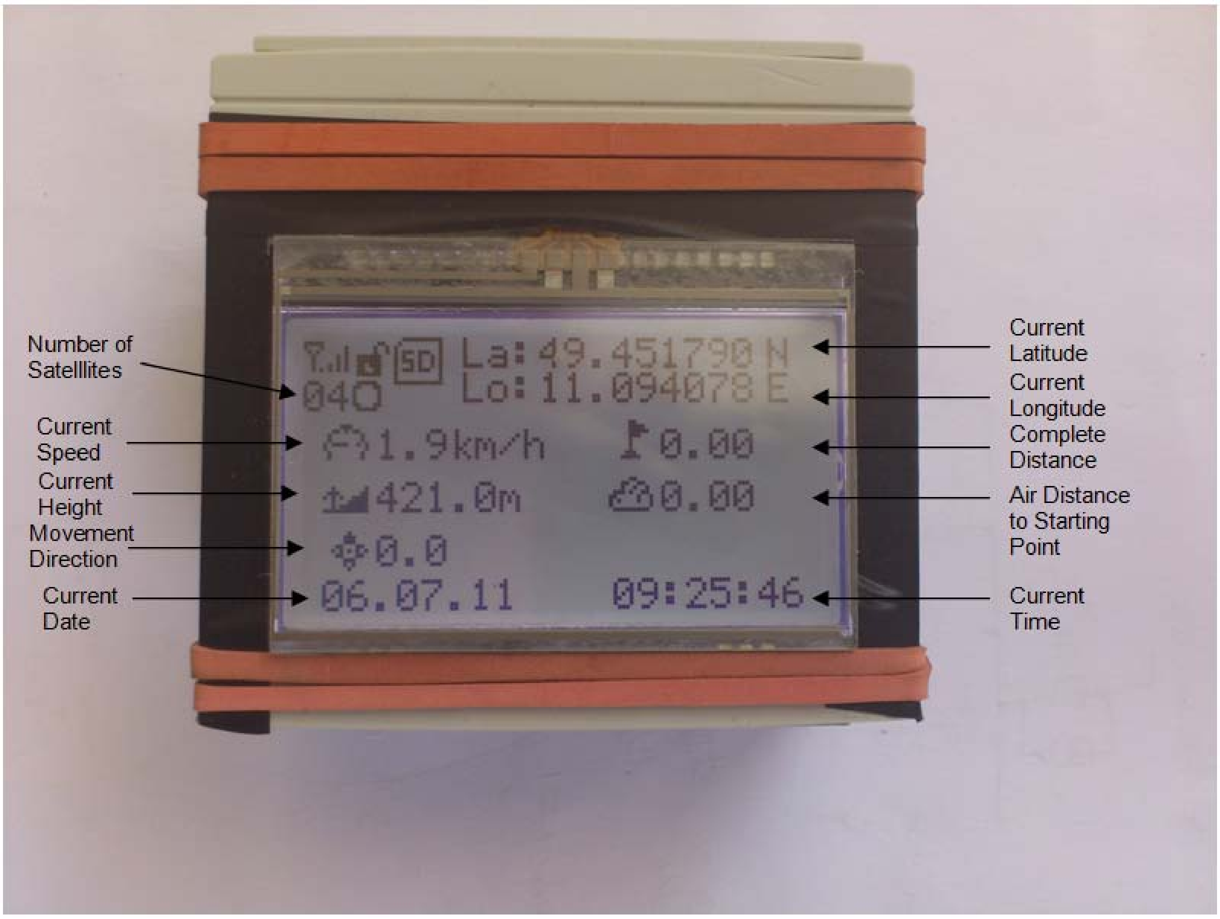


Figure 13: General display layout

5.3. Recording data to SD card

Precondition: A SD card has to be provided to the SD card slot.
For recording NMEA data to a SD card, simply push on the graphical display for at least three seconds. As soon as recording starts a status picture will inform you about the beginning of data recording as shown in figure 14. Figure 15 shows the display during a record. The recording is signalized by the small circle that is alternating filled and unfilled. To stop recording, simply touch on the display for three seconds again.

Note: Before recording the SD card has to be formatted to the `FAT16` format. For doing this with a Linux distribution use the command "`mkfs.vfat -F 16 -S512 /dev/sdXY`" (`sdXY` is the identifier for your SD card) on a terminal. On a PC running Windows simply format the SD card with FAT16 format and a block size of 512Byte (Note: this has not been tested). **The formating is mandatory for recording data.**

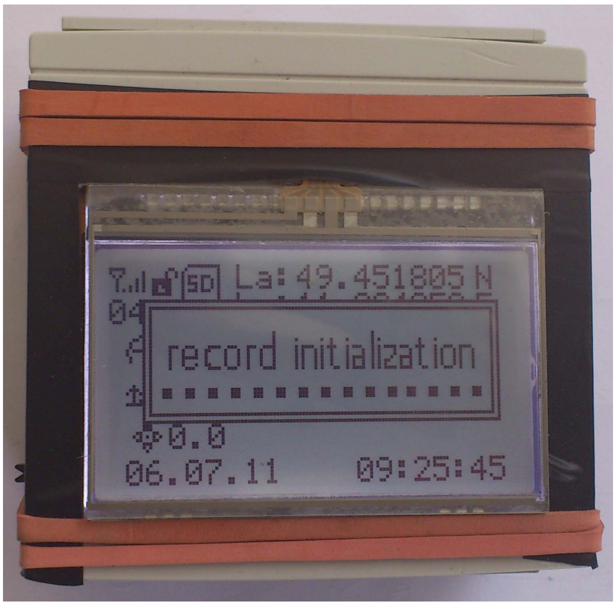


Figure 14: Record Initialization status message

Figure 15: Recording GPS data

5.4. Visualization with `Google Earth`

The recorded NMEA data sets can be used for a 3D-Visualization by using Google Earth. First of all the NMEA data has to be converted to the "Keyhole Markup Language" (KML). This is the supported format for Google Earth geographic data. For this you can use "Wugsis's GPS2KML Converter" that can be found within the `tools` directory of the project. After you have copied the recorded data sets from the SD card to your PC, start `Wugsi` and push on `Choose files` within the `GPS2KML Konverter` tab. Note that the recorded files have a file suffix `.NMA`, so you have to adapt the file filter within the `Open` dialog to `All files (*.*)`. After loading your file in `Wugsi` you can choose some features for the converted `KML` file, like hight, speed, etc. To actually convert the data simply press the `Start` button. Within the logging console on the bottom of the application the file name is printed in which the converted data is stored (`Output`) as shown in figure 16.

Google Earth itself can be downloaded at "http://www.google.de/intl/de/earth/". After an successful installation simply start the application and choose **File→Open...** and select the converted `KML` file. For further detailed information about the features and the services of Google Earth please refer to the official manual. An example of a 3D visualization is shown in figure 17.

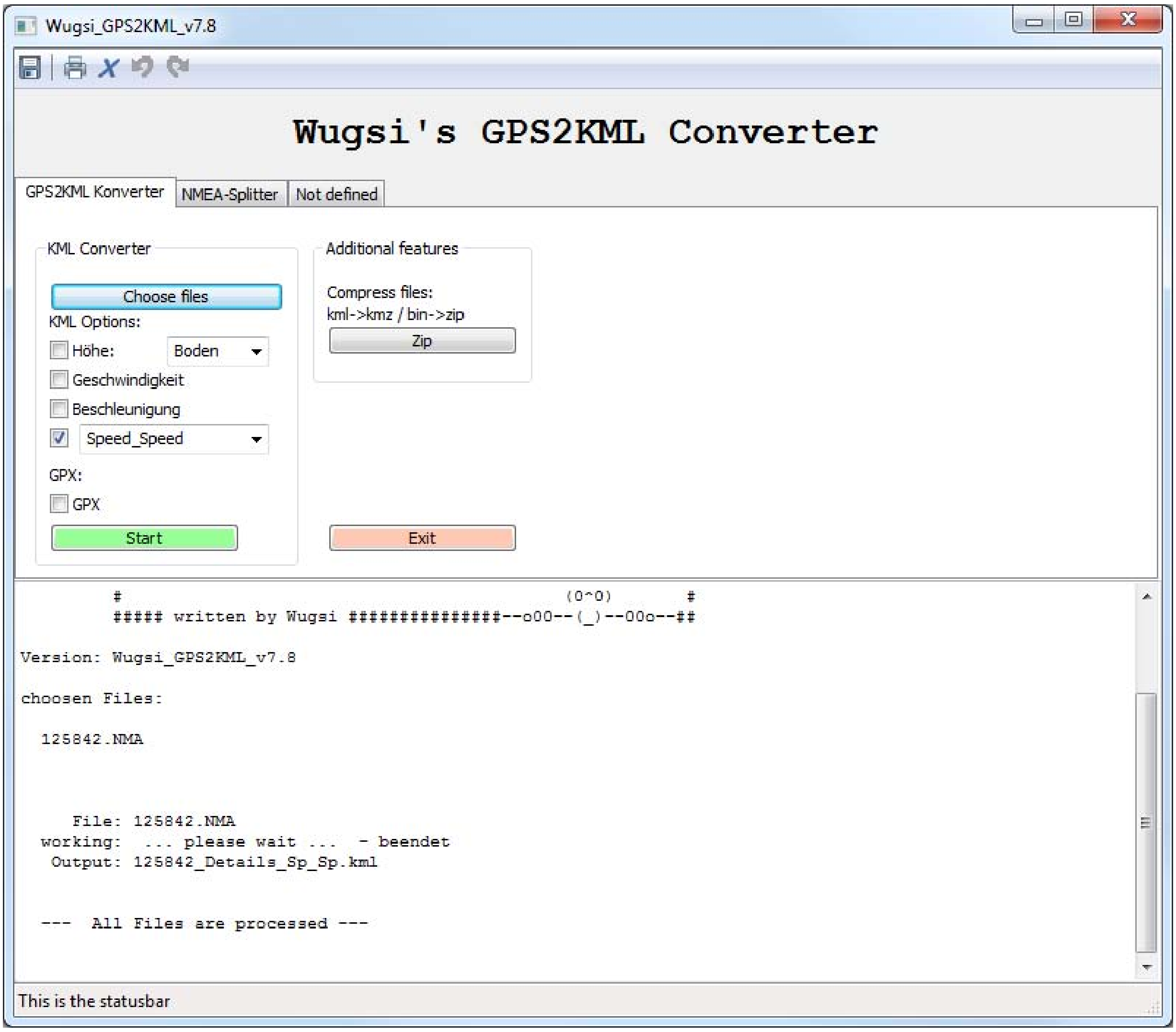


Figure 16: Wugsi's GPS2KML Converter

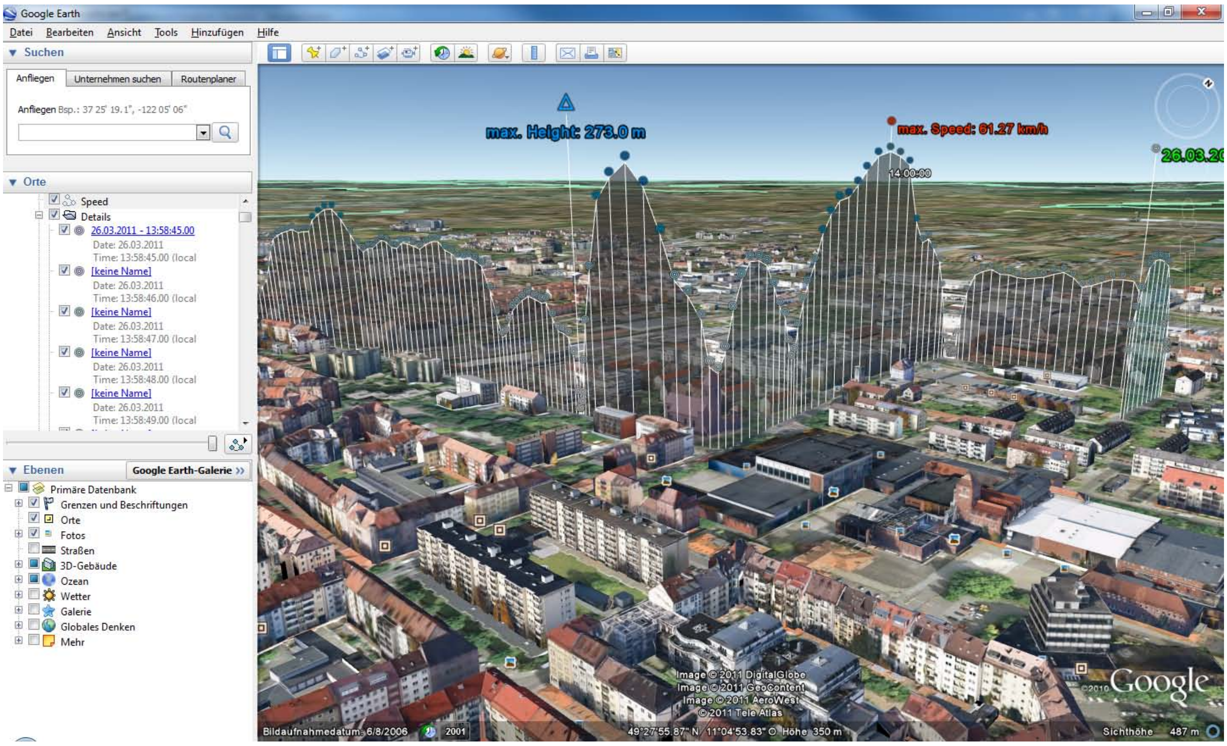


Figure 17: 3D Visualization within Google Earth

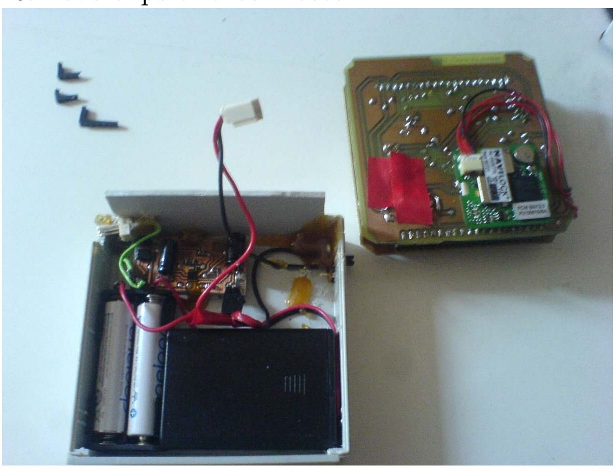
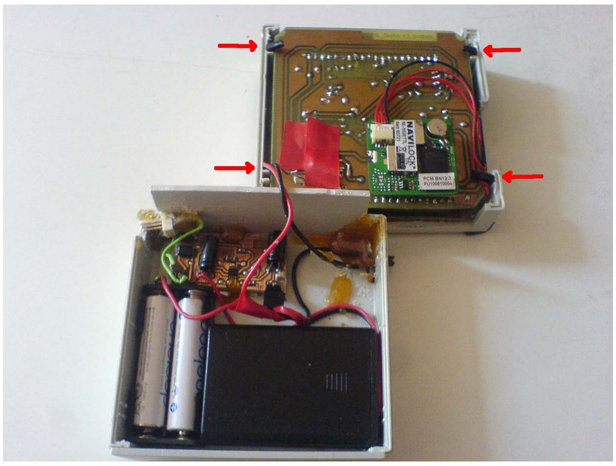
6. Developers Guide

6.1. Setting up the Hardware

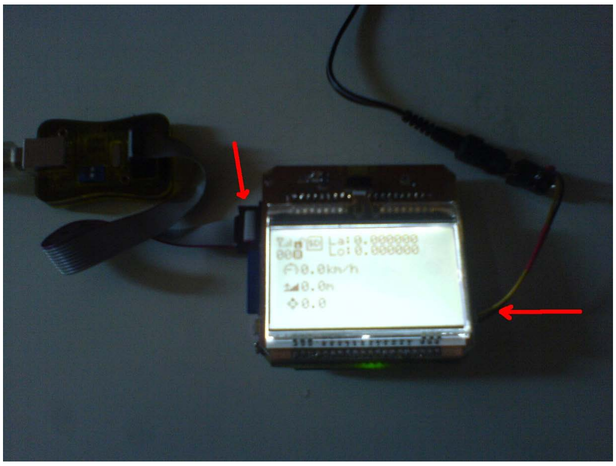
1. Remove rubber bands

2. Open the device

3. Remove clips and connector

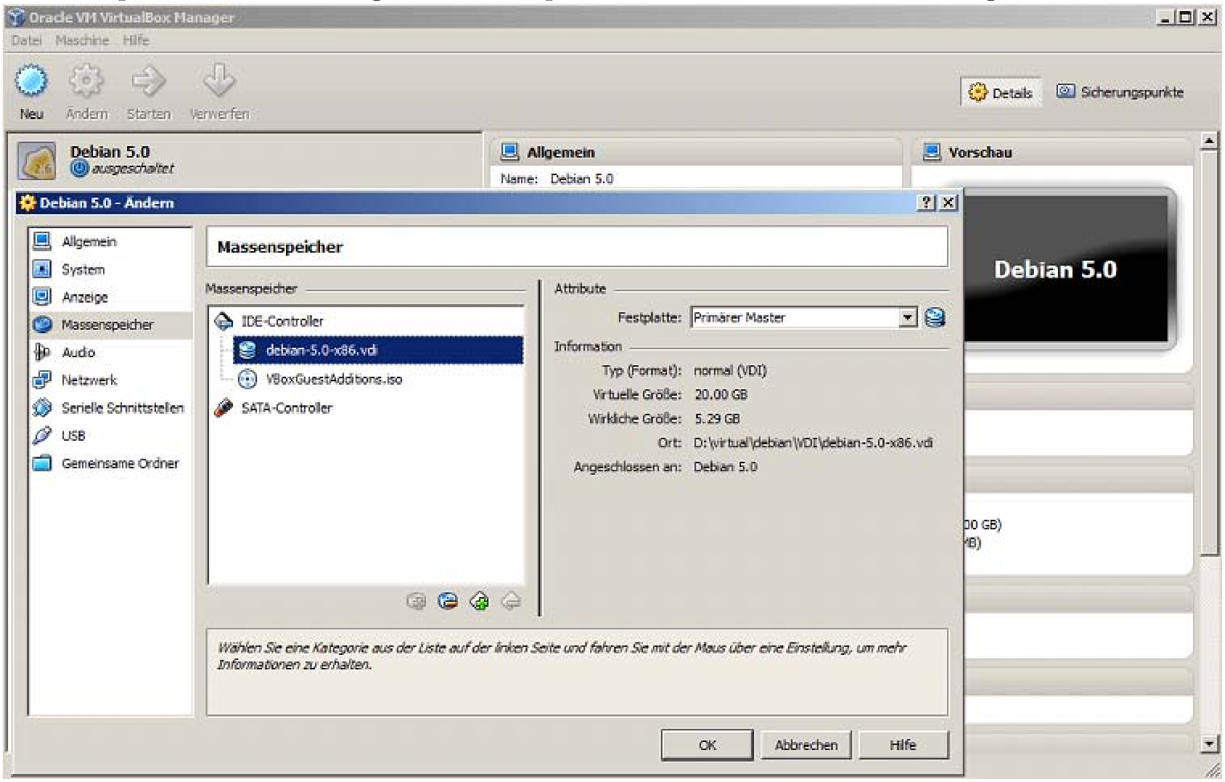
 

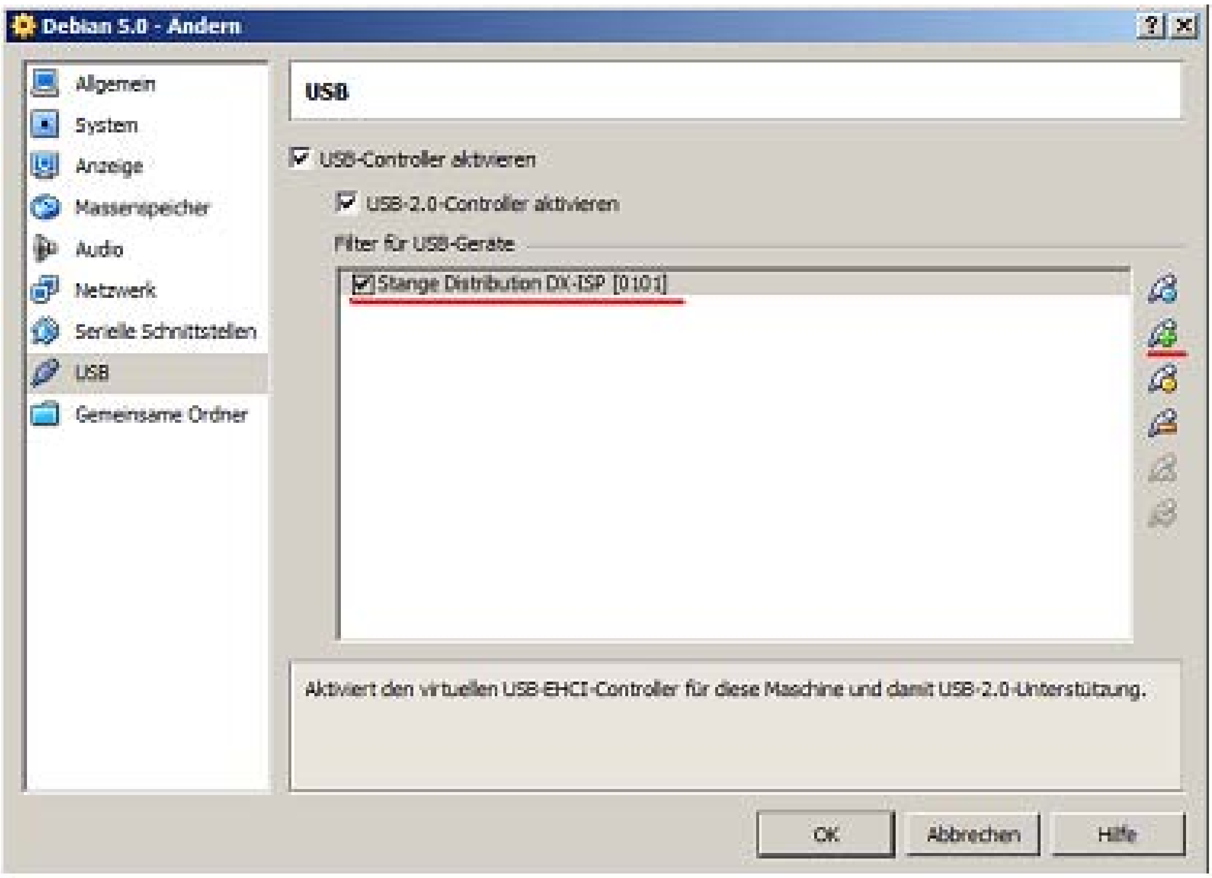
4. plug in external supply voltage and programming device. Be absolutely sure that the black cable is and the yellow one is +! Supply voltages 5.5 V 9 V. Next to the SD Slot is the ISP connector. Next to the voltage jack is the JTAG connector. For JTAG a JTAG controller is needed who is compliant with the ATMega644.



6.2. Software Setup

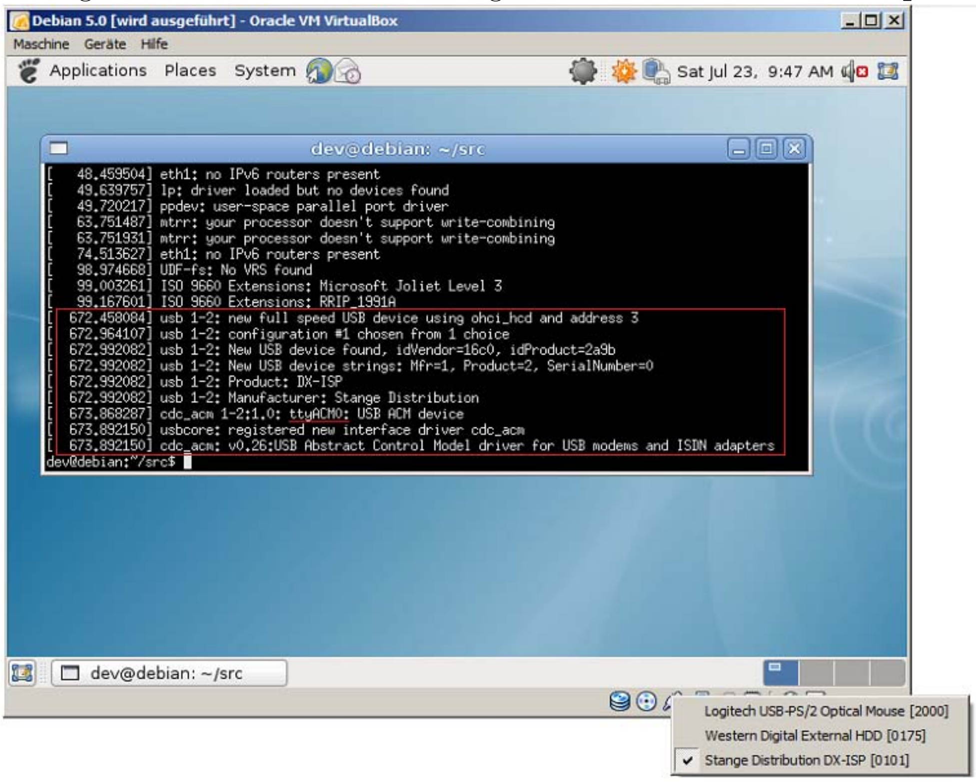
- Extract the Linux Image debian.rar provided on the DVD
- Download VirtualBox: http://www.virutalbox.org/wiki/Downloads, and set up Virtualbox with the provided Linux Image. Also set up a filter for the ISP or JTAG Programmer.





- Boot the Linux Image and login with user dev:
  - **user:** dev
  - **password:** dev
  - **root:** linux
- The directory structure of the home directory of user `dev`:
  - `src` contains the source code
  - `hw` contains hardware circuit and layout files
  - `doc` contains the documentation of the project
  - `doxygen` contains the API documentation
  - `tools` contains some tools used
  - `stubs` contains some code stubs
  - `mfile` is a program that creates initial Makefiles

- Plug-in the ISP Programmer; VirtualBox should connect it automatically due to the filter. With the Strange Distribution Diamex ISP Programmer the device is `/dev/ttyACM0`



- If you are not using the Diamex DX-10 ISP Programmer you need to make changes in `src/Makefile` with the AVRDUDE_PORT and the AVRDUDE_PROGRAMMER.
  Note: See avrdude man-pages which protocol you need for your programmer.

- type `cd  /src` and `make`: The whole source tree is built and if successfully compiled the binary the device will be programmed.

- After make has sucessfully built the source and programmed the device it gives the following messages:

```
avrdude -p atmega644p -P /dev/ttyACM0 -c stk500v2    -U flash:w:main.hex -b 1152
00 -U eeprom:w:main.eep

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.05s

avrdude: Device signature = 0x1e960a
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed
         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "main.hex"
avrdude: input file main.hex auto detected as Intel Hex
avrdude: writing flash (25122 bytes):

Writing | ################################################## | 100% 5.60s

avrdude: 25122 bytes of flash written
avrdude: verifying flash memory against main.hex:
avrdude: load data flash data from input file main.hex:
avrdude: input file main.hex auto detected as Intel Hex
avrdude: input file main.hex contains 25122 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 6.09s

avrdude: verifying ...
avrdude: 25122 bytes of flash verified
avrdude: reading input file "main.eep"
avrdude: input file main.eep auto detected as Intel Hex
avrdude: writing eeprom (262 bytes):

Writing | ################################################## | 100% 0.71s

avrdude: 262 bytes of eeprom written
avrdude: verifying eeprom memory against main.eep:
avrdude: load data eeprom data from input file main.eep:
avrdude: input file main.eep auto detected as Intel Hex
avrdude: input file main.eep contains 262 bytes
avrdude: reading on-chip eeprom data:

Reading | ################################################## | 100% 0.10s

avrdude: verifying ...
avrdude: 262 bytes of eeprom verified

avrdude: safemode: Fuses OK

avrdude done.  Thank you.

dev@debian:~/src$
```

- If you get an error like this check the permissions of the programmer for your user

```
avrdude: ser_open(): can't open device "/dev/ttyACM0": Permission denied
```

- If you get an error like this, be sure if the target does not have any short circuits

```
avrdude: AVR device initialized and ready to accept instructions
avrdude: Device signature = 0x000000
avrdude: Yikes!  Invalid device signature.
```

- If you get errors like this be sure if the target is connected to the supply voltage

```
avrdude: verifying ...
avrdude: verification error, first mismatch at byte 0x0000
         0x0c != 0x00
avrdude: verification error; content mismatch

avrdude: safemode: lfuse changed! Was dc, and is now 0
Would you like this fuse to be changed back? [y/n]
```

- If you get errors like this, the ISP Programmer needs to be reseted. In order to do that, simply unplug and reconnect it to the PC.

```
avrdude -p atmega644p -P /dev/ttyACM0 -c stk500v2    -U flash:w:main.hex -b 115200 -U eeprom:w:main.eep

avrdude: stk500_2_ReceiveMessage(): timeout
avrdude: stk500_2_ReceiveMessage(): timeout
avrdude: stk500_2_ReceiveMessage(): timeout
avrdude: stk500_2_ReceiveMessage(): timeout
^Cmake: *** [program] Interrupt
```

- If you get errors like this make sure the programmer is connected to the computer and the AVRDUDE_PORT is correct in Makefile

```
avrdude -p atmega644p -P /dev/ttyACM0 -c stk500v2    -U flash:w:main.hex -b 115200 -U eeprom:w:main.eep
avrdude: ser_open(): can't open device "/dev/ttyACM0": Permission denied
make: *** [program] Error 1
```

- If you get errors like this, make sure the AVRDUDE_PROGRAMMER is correct in the Makefile

```
avrdude -p atmega644p -P /dev/ttyACM0 -c stk200    -U flash:w:main.hex -b 115200 -U eeprom:w:main.eep
avrdude: can't claim device "/dev/ttyACM0": Invalid argument

make: *** [program] Error 1
```

- Without errors the device is now successfully programmed.
  Note: We only support the Diamex DX-10 ISP Programmer. Other ISP or JTAG programmers are untested but should work just fine.

A. Optional features for future development

There are a lot of possibilities to improve/extend this project. First of all the implementation of an OS would be a major improvement. Using this feature complex user interaction via touchscreen would be possible.

B. Known Bugs

- The graphical notification that indicates whether a SD card is inserted and writable or not, does not work properly.
- In every SD card data record a $\$G\backslash n\backslash r$ is received. But this has no effect on the actual functionality of `Wugsi`
- The distance calculation does not work properly.

References

[EADOGL08] Electronic Assembly, “DOGL GRAFIK SERIE”, 2008, http://www.lcd-module.de/pdf/grafik/dogl128-6.pdf

[NAVLOK09] u-blox 5, “NAVILOCK Datenblatt”, 2009

[NMEA08] u-blox 5, “NMEA, PUBX Protocol Specification”, 2008

[AVRTUT09] www.Microcontroller.net, “AVR-GCC-Tutorial”, 2009, http://www.mikrocontroller.net/articles/AVR-GCC-Tutorial