# LED Programmable Constant Current Source

Jan Krieger <j.krieger@dkfz.de> / <jan@jkrieger.de>*

January 25, 2015

## Contents

*German Cancer Research Center (DKFZ), Department: Biophysics of Macromolecules (Prof. J. Langowski), Im Neuenheimer Feld 580, D-69120 Heidelberg, Germany, tel: +49-6221/42-3395

# 1 Introduction

This control box allows to operate high power LEDs with a constant (setable) current. The LED is driven using an analog constant current source, so there is no blinking from a PWM regulator, when the LED is on. In addition a dongle port allows to connect an I2C EEPROM (24C01) inside a dingle that identifies each LED, so the LED parameters do not have to be entered by hand.

# 2 USB Interface & Command reference

## 2.1 USB Interface

As mentioned before the control box contains a USB to serial converter. Thus the communication with a PC may be done by simply sending text commands over a terminal (e.g. `http://sites.google.com/site/braypp/terminal` for windows). Several commands are defined that allow to control the laser and the remote control box. Each command starts with a single character identifying it and the additional characters that give parameters. Commands with additional parameters require a closing line feed character (`0x0A`, or `\n` in C/C++). Any characters which are not recognized as commands will be ignored, so the interface usually does not stall. If a command features a return value a line feed character (`0x0A`, or `\n` in C/C++) ends the return value (except the info command which is ended by two consecutive line-feeds as it features multi-line return values).

The interface has a configurable baud rate (default is 115200), uses 8 data bits, no parity and one stop bit. The interface is implemented, using a serial to USB converter chip from FTDI. These chips register to the operating system as a virtual serial port, if an appropriate driver is installed. There are drivers for many Windows variants, Linus and MacOS 8,9 and X. Usually a Linux kernel will recognize the interface out of the box. The drivers may be downloaded and installed from `http://www.ftdichip.com/Drivers/VCP.htm`.

## 2.2 Command Reference

This section will give an overview over the available commands together with examples (identifiers in brackets `<...>` have to be replaced by the described data, a value in brackets like `<0xD4>` stands for the binary representation of the value, <LF> stands for a line feed character, <CR> for carriage return). Note that commands are generally case-sensitive!

| *command* | *description* |
|---|---|
| **Status Report & Configuration** | |
| `?` | identifies the remote control box and sends back copyright and version information (multi-line output ended by two consecutive `<LF>` characters. Example output:<br>`---=== PCCS  v1.0 ===---<LF>`<br>` Programmable Constant Current Source (for high power LEDs)<LF>`<br>`  (c) 07.2010 by Jan Krieger (DKFZ)<LF>`<br>`  j.krieger@dkfz.de --  jan@jkrieger.de<LF><LF>` |
| `V` | return version information. Example output:<br>`1.3.319.1735 (05.2010)<LF>` |

| *command* | *description* |
|---|---|
| `P<param_name>=<param_value><LF>` | set one of the internal parameters of the remote control box to a specified value. These parameter names are defined: • `FAN_START` (0..100) starting temperature of internal fan • `FAN_FACTOR` (0..255) fan speed increase per degree C of temperature increase • `FAN_OFFSET` (0..255) PWM value when the fan starts rotating • `BAUDRATE` (2400,...,9600,...,115200,... <=1000000) sets baud rate of the USB to serial converter (default is 115200) • `LCD_BACKLIGHT` (0..255) intensity of the displays backlight illumination • `CURRENT_MAX` (0..2000) maximum current (in mA) of the attatched LED • `CURRENT_100` (0..2000) 100% current (specified operation current) of the attatched LED in mA • `IADC_MEASUREMENTS` (0..100) number of measurements over which to average when measuring the LED current • `EADC_MEASUREMENTS` (0..100) number of measurements over which to average when measuring the external analog input • `IADC_WAIT` (0..100) number microseconds to wait between two consecutive digitalizations of the LED current • `EADC_WAIT` (0..100) number microseconds to wait between two consecutive digitalizations of the external analog input • `LED_BLINK` (0..20000) blinking frequency of the LED |

**User I/O**

| | |
|---|---|
| `A` | return the voltage (in mV) at the external AD input |
| `O0`, `O1` | switch the external TTL output off or on |

**LED Control & Status**

| | |
|---|---|
| `T` | return the internal temperature in degree C |
| `L0`/`L1` | switch the LED off or on |
| `M<pulselength_in_ms>` | switch the LED on for the given duration in milliseconds only |
| `U<pulselength_in_us>` | switch the LED on for the given duration in microseconds only |
| `N` | return whether the LED is switched on/off |
| `I<current_in_milliamps>` | set the current LED current in mA |
| `G` | return the current set current in mA |
| `C` | return the current measured current in mA |
| `Y` | return the name of the current LED |
| `X` | return the maximum current of the LED in mA |
| `%` | return the 100% current of the LED in mA |
| `q` | return the manufacturer of the current LED |
| `Q<manufactirer>` | set the manufacturer of the current LED |

| *command* | *description* |
|---|---|
| `j` | return the wavelength of the current LED |
| `J<wavelength>` | set the wavelength of the current LED |
| `f` | return the max. power of the current LED |
| `F<wavelength>` | set the max. power of the current LED |
| `D` | dump the contents of the current dongle EEPROM |
| `W<LED_NAME>,<max_current_mA>, <100%_current_mA>` | write data into the dongle EEPROM. The LED name may not be longer than 10 characters and the last two parameters not longer than 4 digits. |

## 3 Technical Realization

### 3.1 Hardware Overview (Electronics)

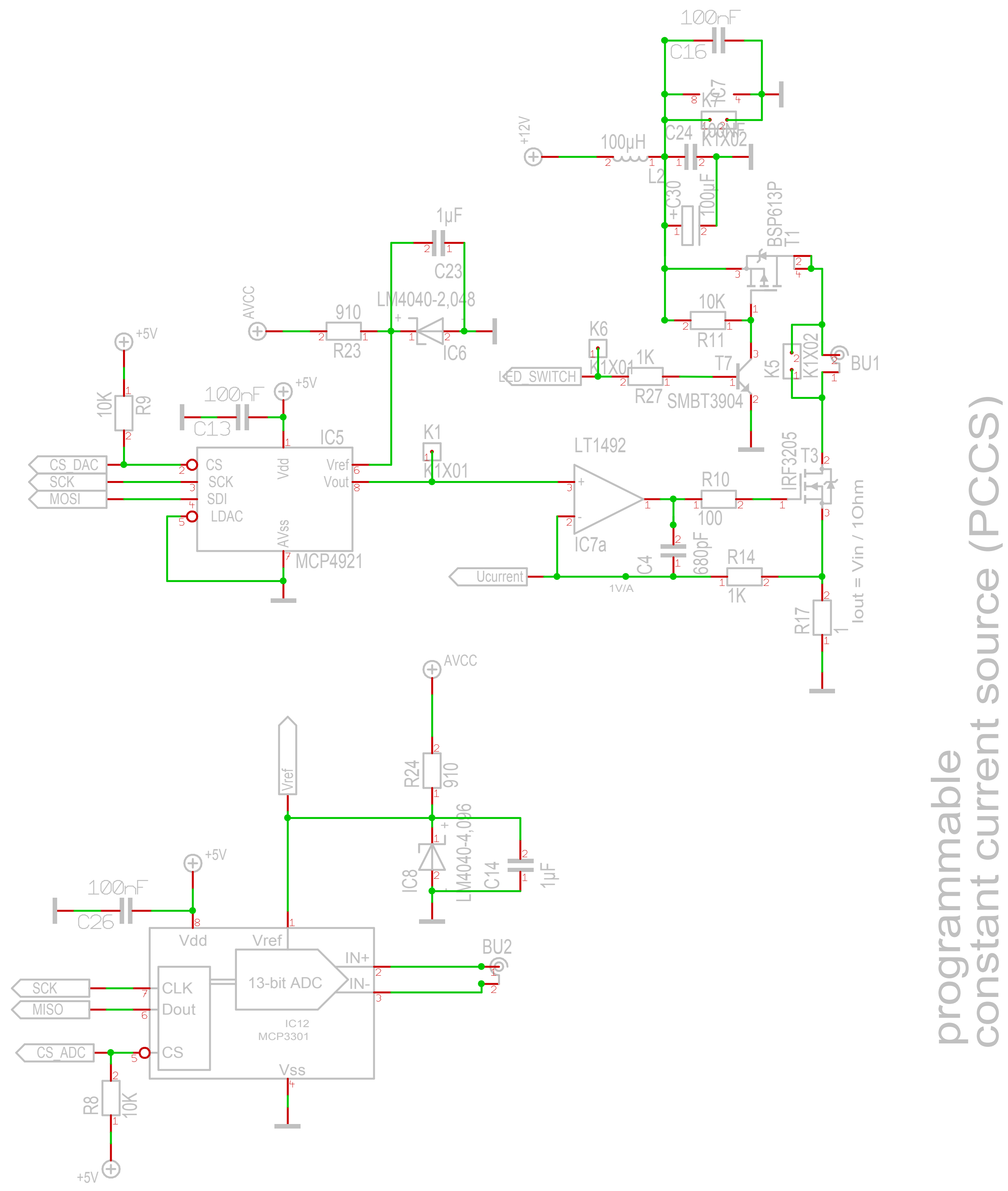


Figure 1: Schematic: Analog Part

The PCCP circuit can be separated into three parts:

1. **the analog part**, which controls the current through the LED, see Fig. 1.
   The analog part consists of a DA converter (IC5), which is connected to an analog constant current sink (IC7a, T5). With T1, the LED can be switched on and off completely. The resolution of the DAC is approximately 1mA. The supply current is filtered with L2,C24,C30. Since the current sink is not set up as a switched-mode regulator, the excess power is converted to heat by T5. Therefor cooling of T5 should be added (with a heatsink and a fan, see Fig. 2).
   In addition, the analog part contains an optional (!!!) AD converter (IC12), which can be used to measure any voltage in your setup. This circuit is NOT necessary for the operation of the basic power sink.

2. **the digital part**, which contains the microcontroller, display, the USB connection and power-supply, see Fig. 2.
   The digital part contains an ATmega microcontroller, which controls the analog part. The internal AD converter is used to read the current through the LED. In addition, the controller is connected to an LC display and four buttons, that allow to set the current through the LED. Also a serial-to-USB converter (IC2) is part of the circuit and allows for easy communication with a PC. The temperature sensor R13 and T2 are sued to build a temperature controlled fan controller to cool T5. The transistors T4/T5 constitute an optional digital output.

3. a small circuit inside a **dongle**, which is specific to each LED. This dongle is a circuit, connected to a D-SUB 9-pole connector and basically contains an IIC EEPROM, which stores the LED parameters. This way, the circuit can read the LED parameter (max. current, name, ...) from this dingle and you can easily excahnge LEDs during operation: Just plug in the new LED and exchange the dongle. See Fig. 2 for teh schamtic.

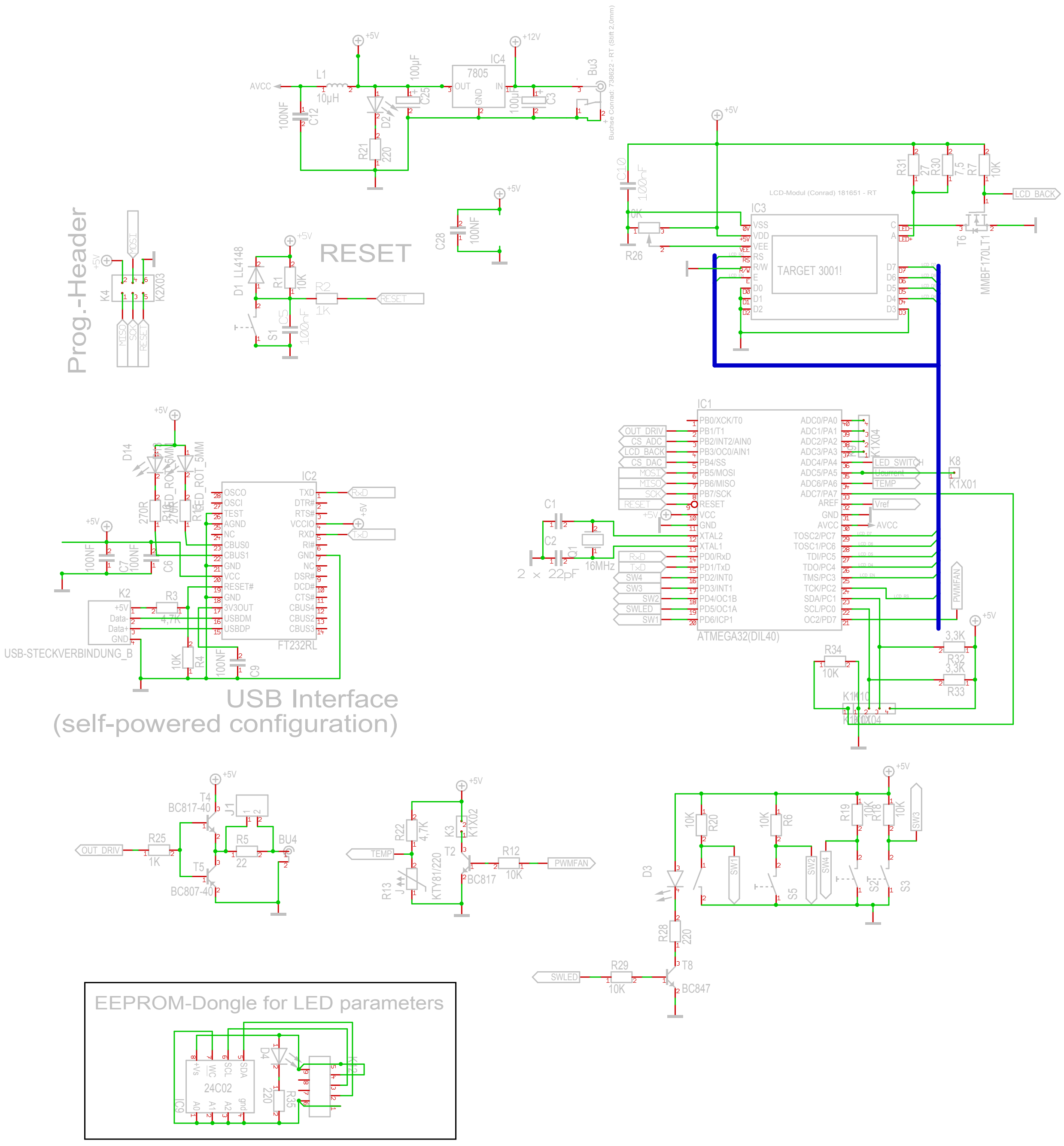
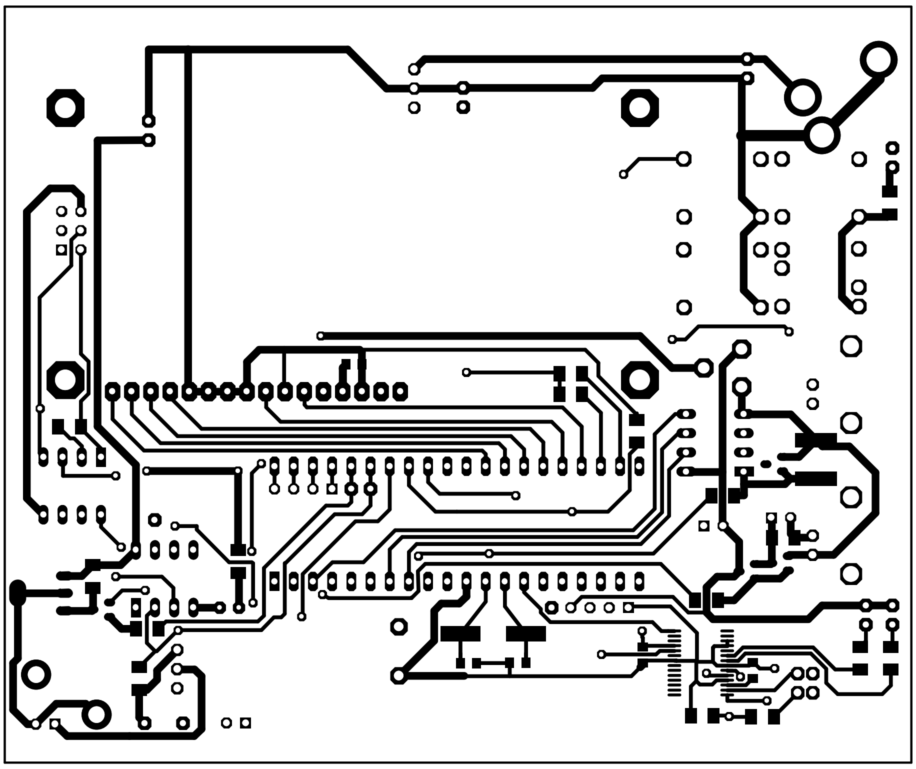


Figure 2: Schematic: Digital Part

## 3.2 PCB

copper top



copper bottom

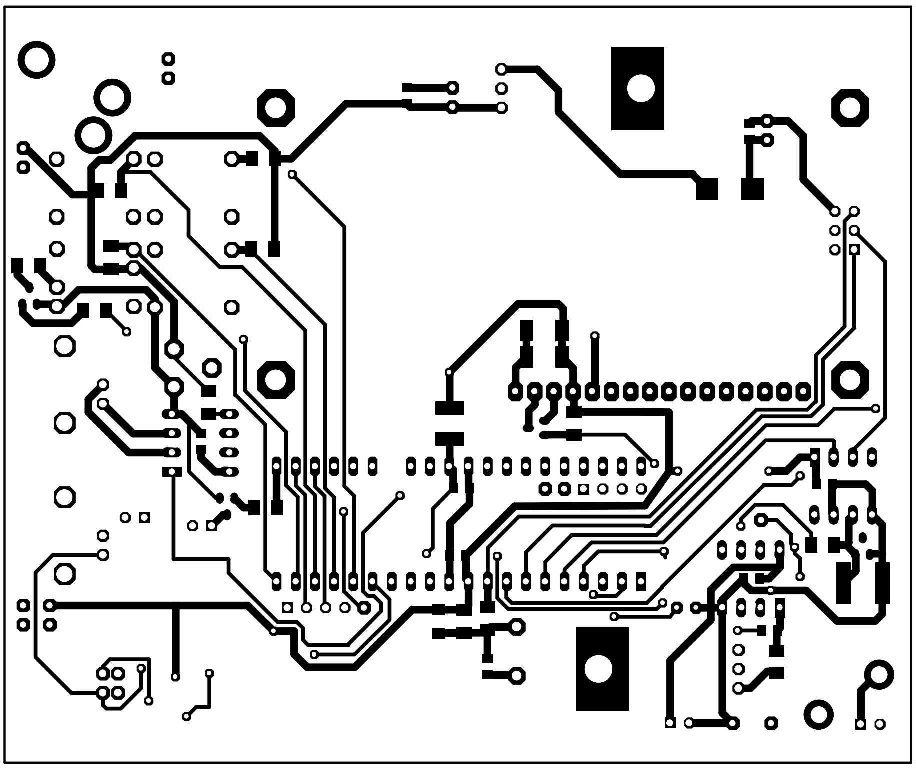
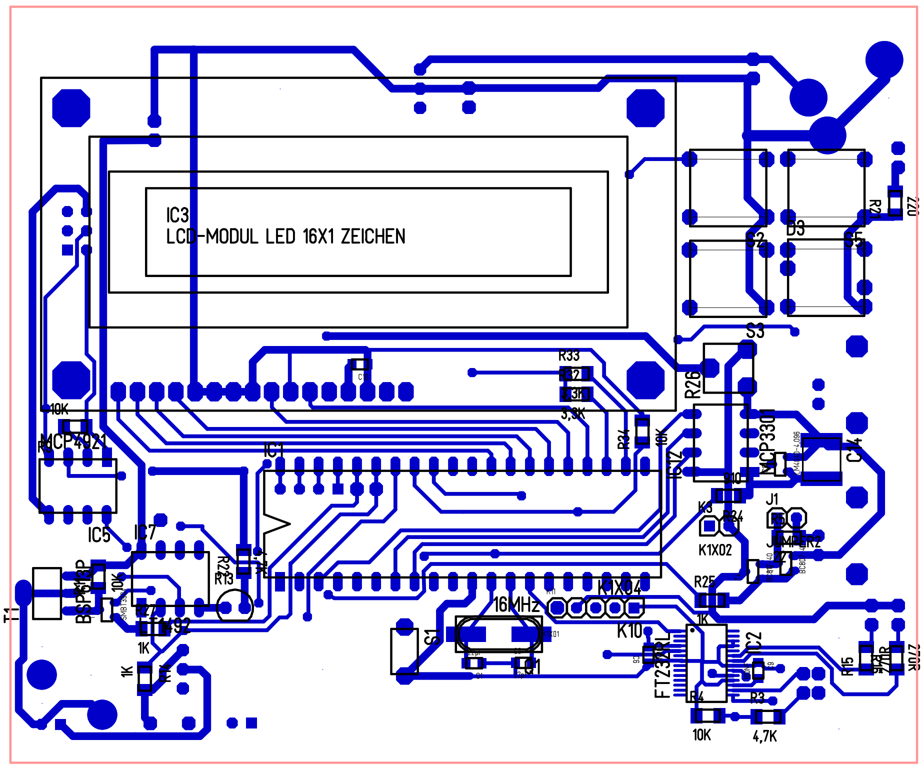


Figure 3: Board, top and bottom

copper top



copper bottom

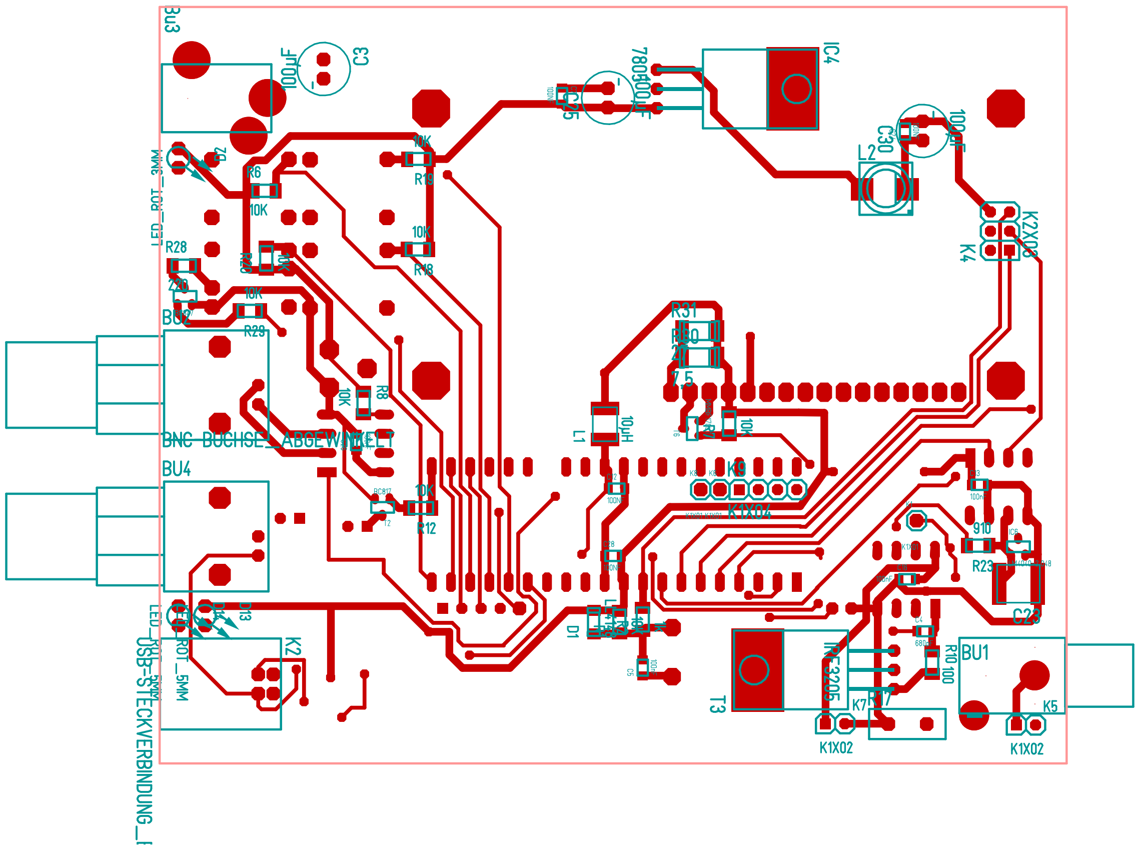


Figure 4: Board, top and bottom, including parts

## 3.3 Implementation Details and Hints

- The FTDI USB to serial converter has areprogrammed device name

*LASERBOX*

Still you can use the standard FTDI virtual COM port driver for this device:

`http://www.ftdichip.com/FTDrivers.htm`

- The FTDI USB to serial converter has to be programmed to be self-powered. Otherwise the serial connection does not work on Linux (on Windows system the problem does not seem to exist).
- You can bind a box to a spoecific serial port. On **Windows** this can be done in the driver settings. On **Linux** this may be achieved using the `udev` hardware enumeration system:
  1. create a udev rules file with a name like e.g.: `/etc/udev/rules.d/99-laserboxes.rules`
  2. add a line like this:

     ```
     #LASERBox
     KERNEL=="ttyUSB*", SUBSYSTEMS=="usb", ATTRS{product}=="B040SERVO", SYMLINK+="ttyUSB_B040SERVO"
     ```

     which will map the ttyUSB device with the product name B040SERVO to the device file `/dev/ttyUSB_B040SERVO` in addition to its default file `/dev/ttyUSB`*x*. Instead of the product name you can also use any other USB device property, as reported by

     ```
     lsusb -v -s <bus>:<devnum>
     ```

     Possible filters are e.g.:

     ```
      ATTRS{serial}=="A800d0Bl",
      ATTRS{idVendor}=="1a72",
      ATTRS{idProduct}=="1007",
      ATTR{vendor}=="0x149a",
      ATTR{device}=="0x0005"
     ```

  3. After saving the rules file you will have to tell udev that there are new/changed rules by executing

     ```
     /etc/init.d/udev restart
     ```

     The rules will be executed when the new device is connected to the computer. So if it's already connected: pull and reconnect the plug.

## 3.4 Programming the Laserbox Controller

The main microcontroller of the control box is an Atmel ATmega32. It's flash memory may be programmed either by a standard AVR programmer, like usbasp (see `http://www.fischl.de/usbasp/` and Fig. 5) or by a bootloader already installed on the controller. In the first case you should set the "TargetSupply" jumper of the usbasp and disconnect the power supply of the control box. This will supply 5V to the electronics of the control box which is OK for short periods (few minutes). Then you can use a programmer software like avrdude to program the controller. The programmer and the controller are connected by a standard $2 \times 3$ ribbon cable connectors (see Fig. **??**).

If the controller is already equipped with the bootloader `avrprog_boot v0.85` (by Martin Thomas, `http://www.siwawi.arubi.uni-kl.de/avr_projects#avrprog_boot`), you simply have to reset the controller (switch power off/on or use reset switch) while holding the ON/OFF button pressed down. Afterwards a new program can be loaded using a AVR109/AVR910 programmer software (e.g. avrdude, AVR Prog or AVR Studio) via the serial host port of the control box. Note that the bootloader is started only if the ON/OFF button is held down during a reset, otherwise the standard software starts and the control may be used to control a laser.

Here are some more hints on programming the controller:

- The fuse bits for the ATmega324p controller without bootloader are:

  `LFUSE = 0xFD; HFUSE = 0xDF; EFUSE = 0xFF.`

- The fuse bits for the ATmega324p controller with bootloader (2047 bytes) are:

  `LFUSE = 0xFD; HFUSE = 0xD8; EFUSE = 0xFF.`

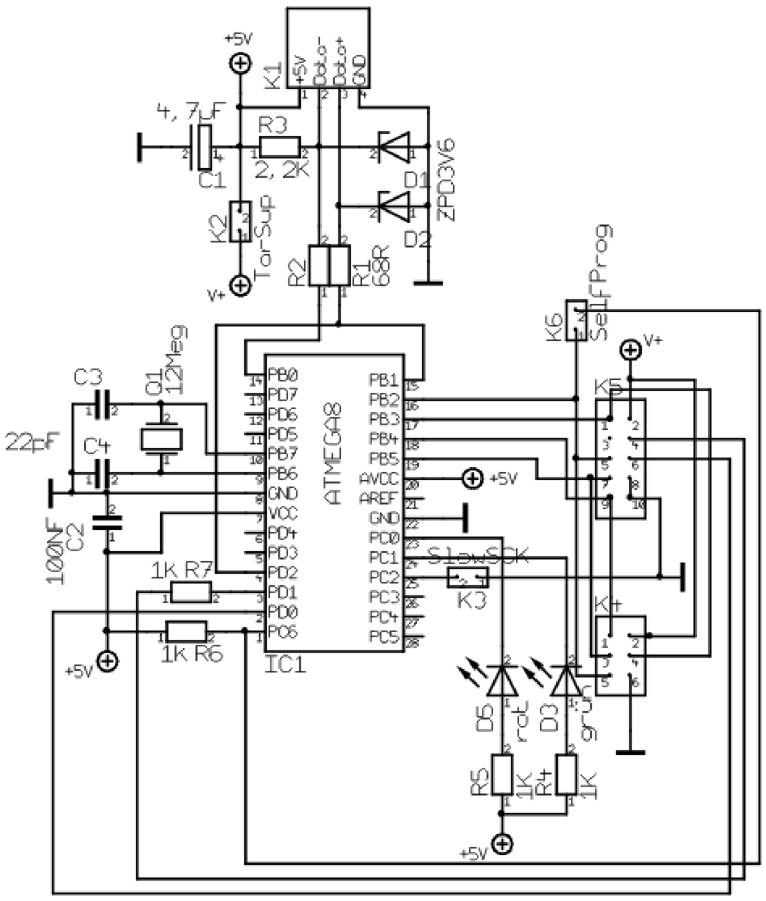


Figure 5: USBasp – AVR programmer for USB

## 4 Literature & Datasheets

- FT232RL: `http://www.ftdichip.com/Documents/DataSheets/DS_FT232R.pdf`