Resistor Heater (PID loop)

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

March 11, 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

2 Circuit for the Main Controller

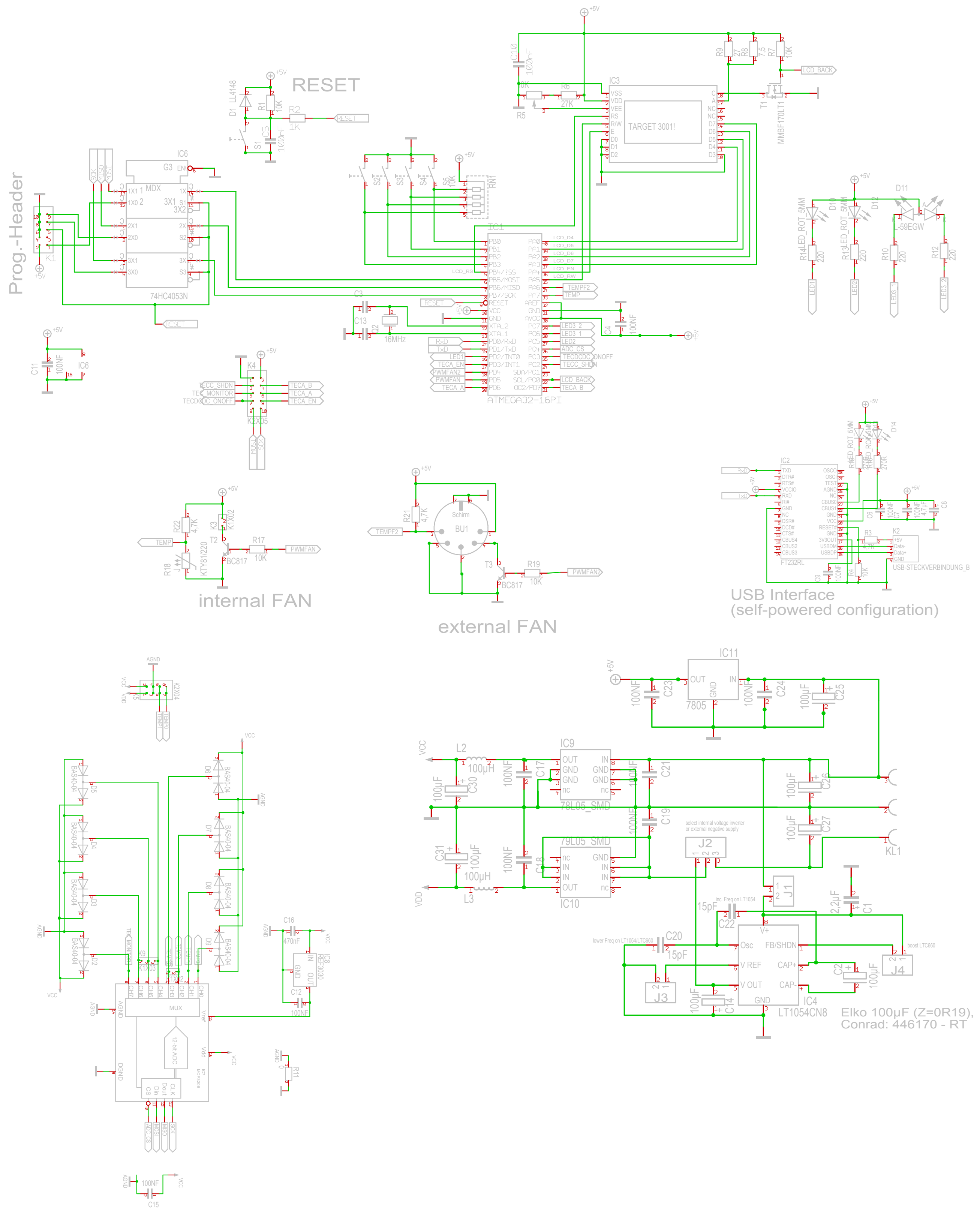


Figure 1: Schematic

2.1 PCB

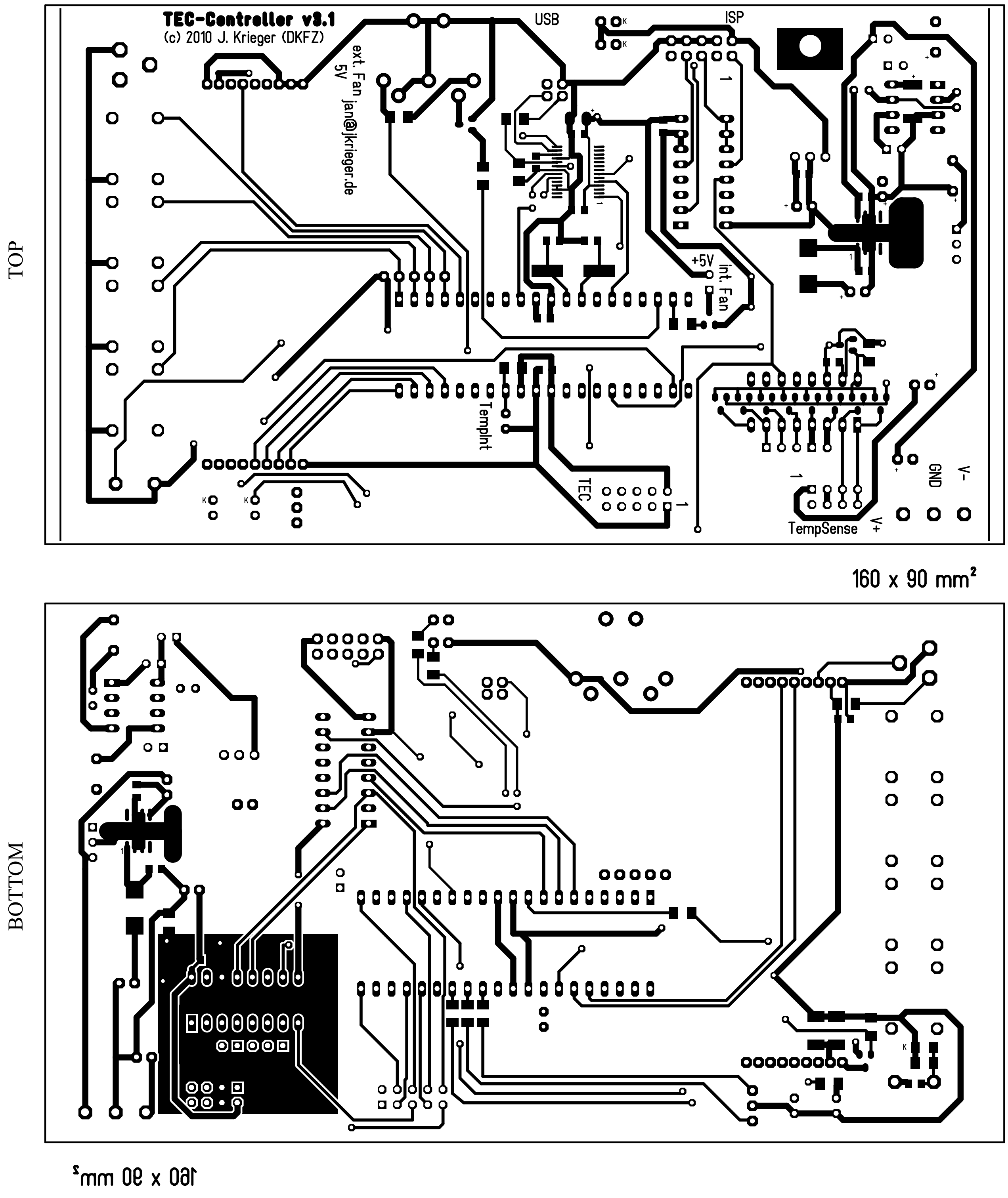
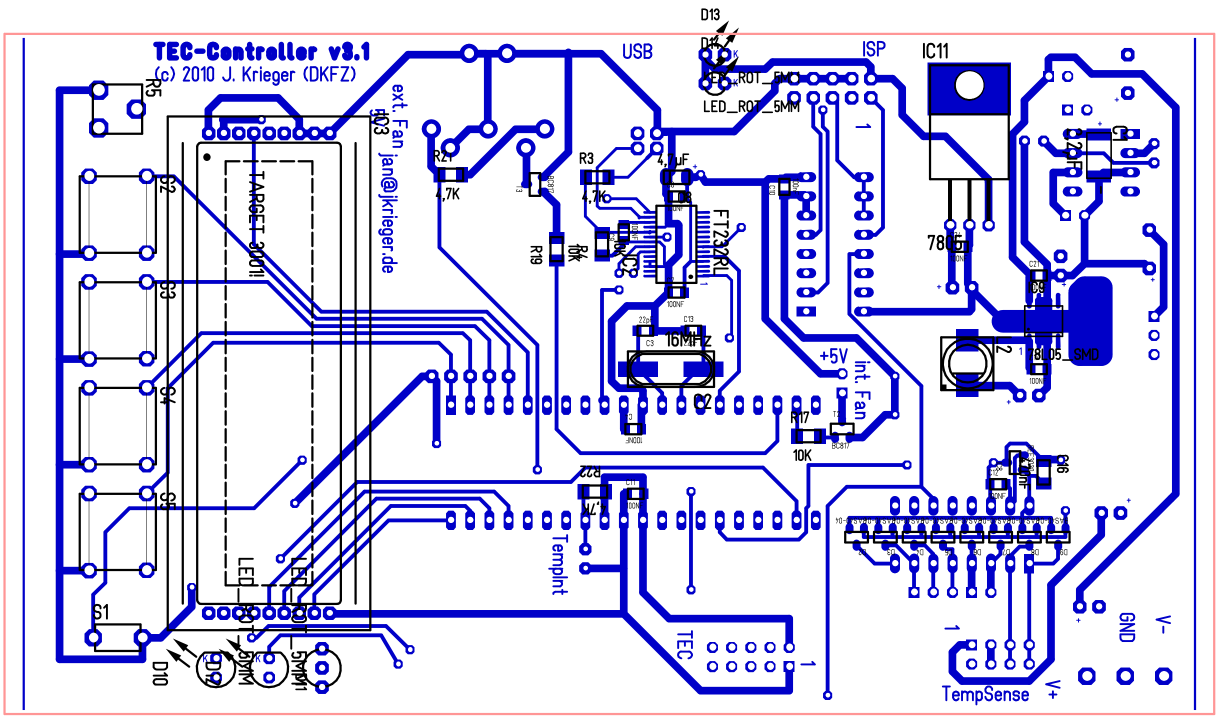


Figure 2: PCB

TOP



160 x 90 mm²

BOTTOM

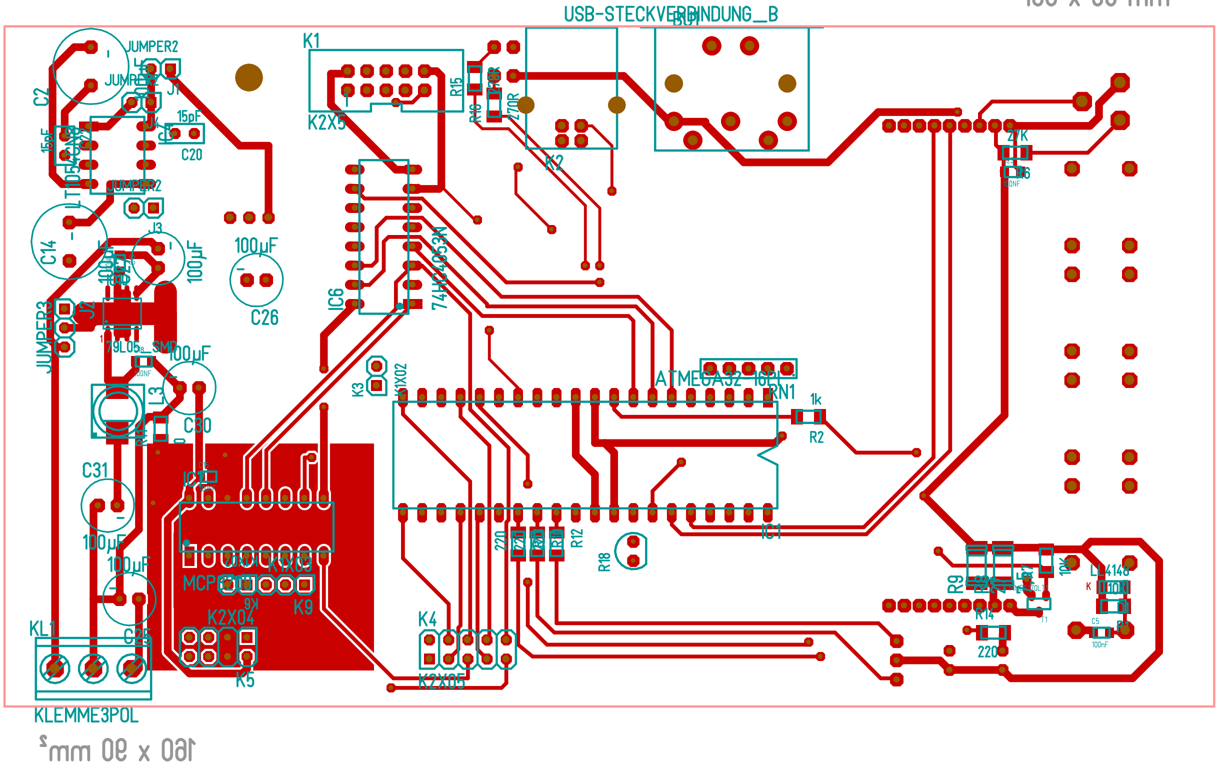


Figure 3: PCB with parts

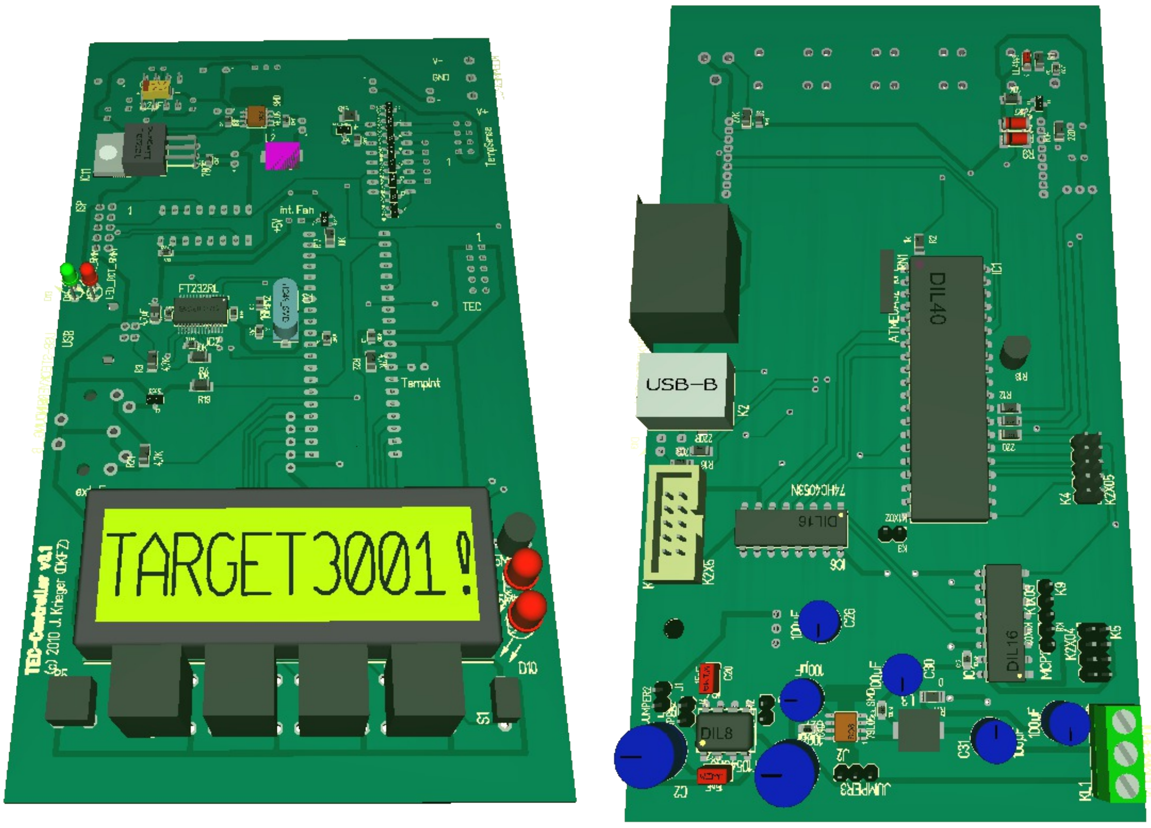


Figure 4: 3D view of the PCB

3 Circuit for dual PT100 Amplifier

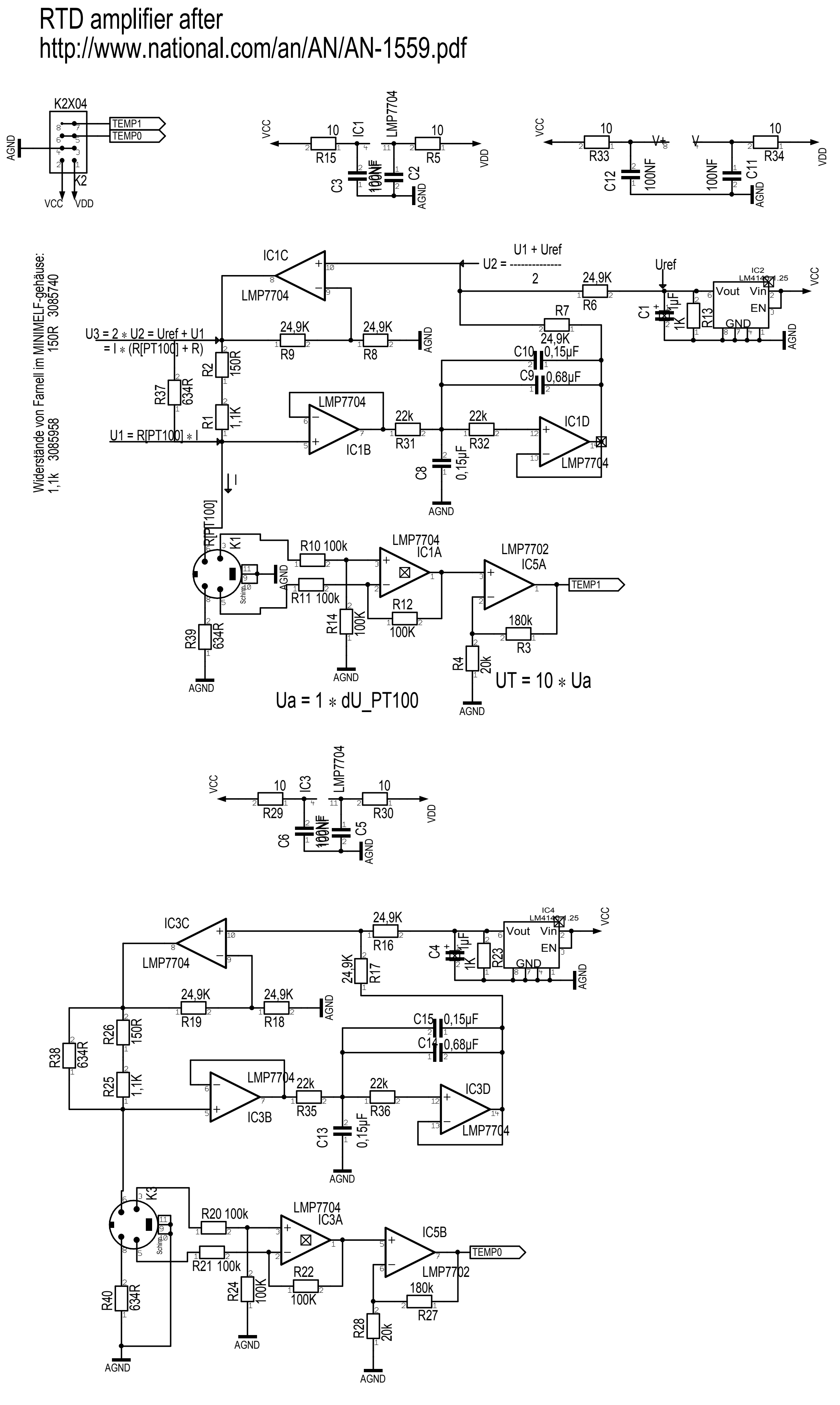


Figure 5: schematic of dual PT100 amplifier

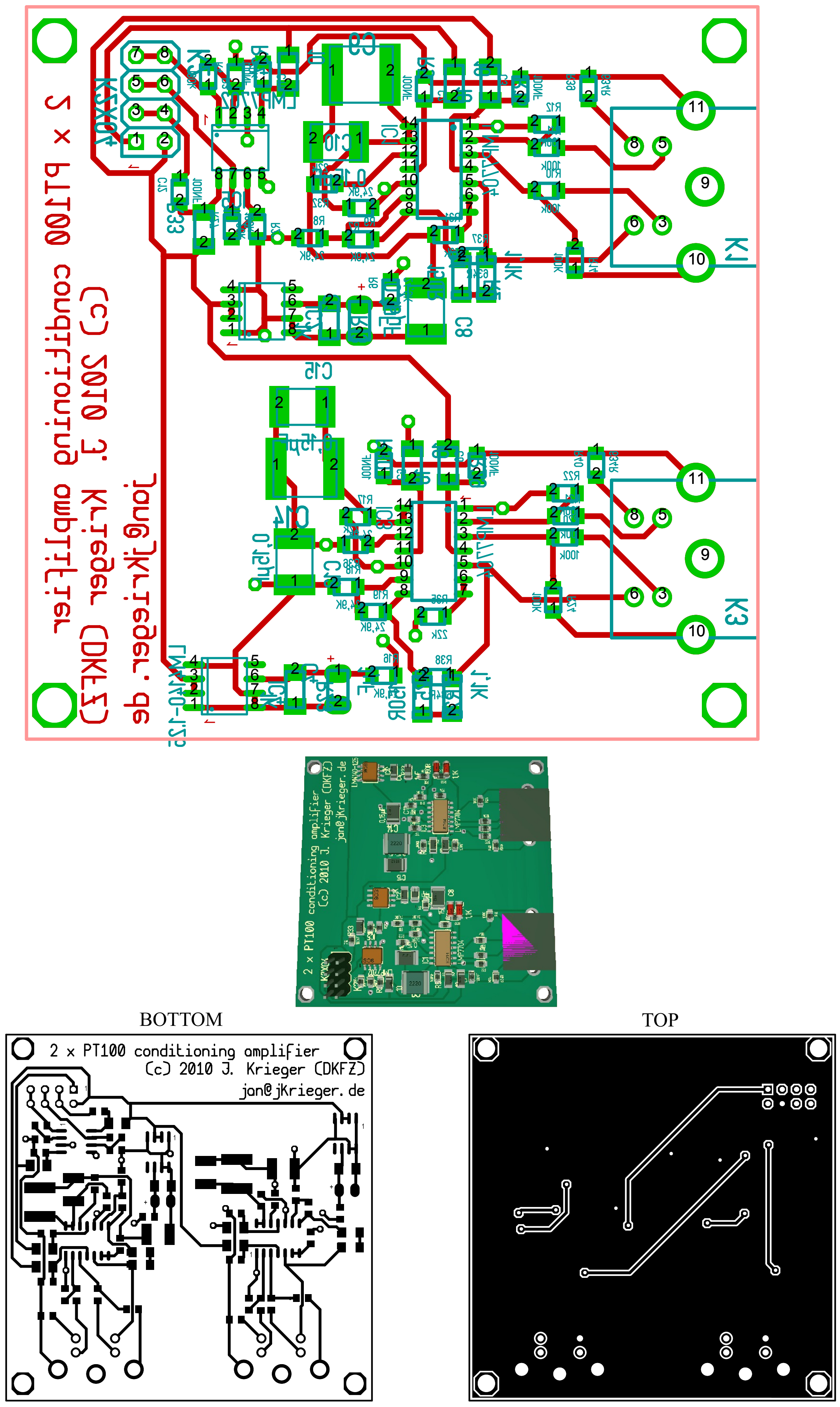


Figure 6: PCB of dual PT100 amplifier

4 USB Interface & Command reference

4.1 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 two line feed characters (`0x0A`, or `\n` in C/C++) end the return value.

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`.

4.2 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:
	`---=== Resitance Heater Temperature Control  v1.0 ===---<LF>`
	`  (c) 07.2010 by Jan Krieger (DKFZ)<LF>`
	`  j.krieger@dkfz.de --  jan@jkrieger.de<LF><LF>`

`V`	return version information. Example output:
	`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: • `T_INPUT` (0,1,2) sets the input to use for the PID loop (0: sensor 1 T_1, 1: sensor 2 T_2, 2: average of sensors 1 and 2 $(T_1 + T_2)/2$) • `PT100_OFFSET0` (-1000..1000) sets the offset for the first PT100 in units of 0.1 °C • `PT100_OFFSET1` (-1000..1000) sets the offset for the second PT100 in units of 0.1 °C • `SHUTDOWN` (0..100) sets the shutdown temperature • `BAUDRATE` (2400,...,9600,...,115200,... <=1000000) sets baud rate of the USB to serial converter (default is 115200) • `FAN_MIN` (0..100) start temperature in °C for the internal fan • `FAN_INCREASE` (0..100) increase of fan speed in PWM setps per °C • `PID_I` (0..1000) integral term prefactor for PID loop • `PID_IMAX` (0..1000000) maximum for error integration • `PID_D` (0..1000) differential term prefactor in PID loop • `PID_P` (0..1000) proportional term prefactor in PID loop
`P<param_name><LF>`	get the current value of one of the internal parameters of the remote control box. The parameter names are the same as above.
Temperature Control & Status	
`T<set_temp>`	set the current set temperature in units of 1/10°C
`t`	return the current set temperature in units of 1/10°C
`O0/O1`	switch heater on or off
`o`	returns whether the heater is currently turned on
`s`	return all state variables' current values <TEMP0>, <TEMP1>, <SET_TEMP>, <SET_VALUE>, <HEATING_ONOFF_STATE>, <ERROR>, <INTERNAL_TEMPERATURE>
`1`	returns the temperature of the first PT100 in units of 1/10°C
`2`	returns the temperature of the second PT100 in units of 1/10°C
`i`	returns the internal temperature in units of °C
`v`	returns the current set value
`e`	returns the current error state

5 Technical Realization

5.1 Hardware Overview (Electronics)

5.2 Implementation Details and Hints

- The FTDI USB to serial converter has areprogrammed device name

  ***

  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-usbboxes.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}=="A800d0B1",
      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.

5.3 Programming the Microcontroller

The main microcontroller of the control box is an Atmel ATmega. 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 remove the "TargetSupply" jumper of the usbasp and connect the power supply of the control box. Then you can use a programmer software like avrdude to program the controller. The programmer and the controller are connected by a standard 2×3 ribbon cable connectors (see Fig. 2).

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 ATmega32 controller without bootloader are:

- The fuse bits for the ATmega32 controller with bootloader (2047 bytes) are:

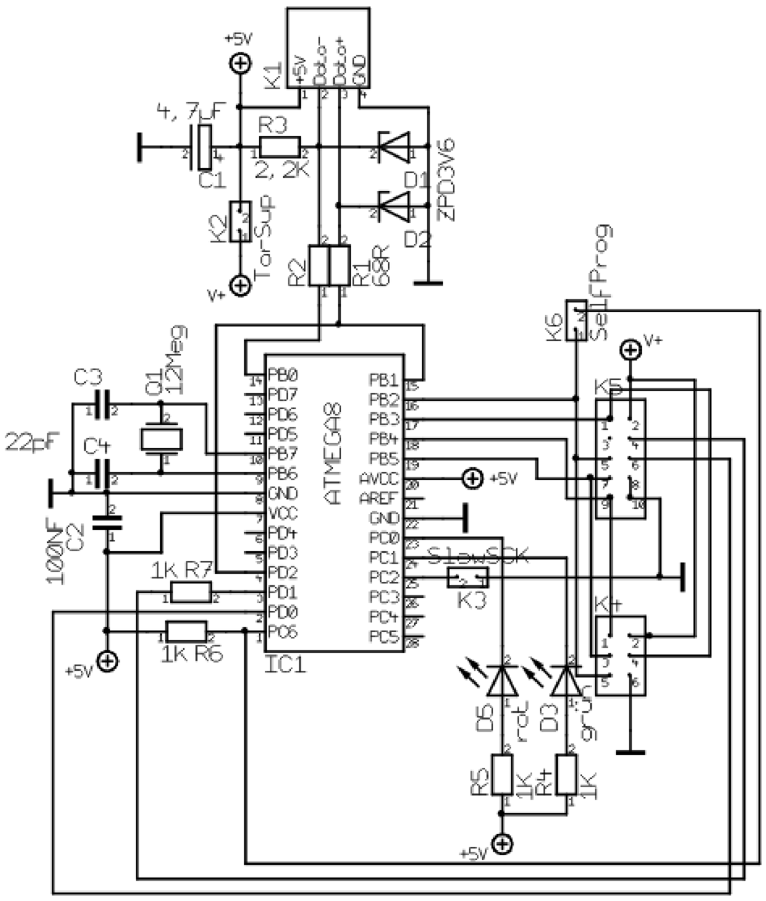


Figure 7: USBasp – AVR programmer for USB

6 Literature & Datasheets

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