# Temperature Dependent Study of the Thermopower of Atomic and Molecular Scale Contacts

Dissertation Submitted for the Degree of
Doctor of Natural Sciences (Dr.rer.nat.)

presented by

Thomas Möller

at the


Faculty of Mathematics and Natural Sciences
Department of Physics

Date of the oral examination: July 1st, 2020

1st reviewer: Prof. Dr. Elke Scheer
2nd reviewer: Prof. Dr. Johannes Boneberg

Konstanz, 2020

# Contents

# Deutschsprachige Zusammenfassung

In dieser Arbeit wird die Thermokraft von atomaren Goldkontakten gemessen, die in einem lithographisch hergestellten und mechanisch kontrollierten Bruchkontakt erzeugt wurden. Auch an Einzelmolekülkontakten wurden entsprechende Messungen versucht. Die übliche Theorie für Ladungstransport ist in derart kleinen Strukturen nicht mehr anwendbar, da hier quantenmechanische Effekte berücksichtigt werden müssen. Moleküle können durch diese Effekte auch nützliche elektronische Eigenschaften haben, die nicht denen bekannter elektronischer Bauteile entsprechen. Um zu verstehen, bei welchen Energien das höchste besetzte Molekülorbital und das niedrigste unbesetzte Molekülorbital liegen, sowie um die Auswirkungen von quantenmechanischer Interferenz in entsprechend konzipierten Molekülen zu messen, liefern Thermokraftmessungen wertvolle Aussagen. Diese Messergebnisse können genutzt werden, um theoretische Berechnungen zu verifizieren, die wiederum die Grundlage für die Konzeption von Molekülen für spezifische Anwendungen sind. Eine experimentelle Herausforderung ist die zum Schutz der Probenstruktur gering zu haltende Temperaturdifferenz, denn dadurch beträgt die erzeugte Thermospannung von Goldkontakten nur einige Mikrovolt. Während hier der Widerstand im Bereich unter 20 kΩ liegt, beträgt er in Einzelmolekülkontakten üblicherweise zwischen 10 MΩ und 1 GΩ. Daher ist der von der Thermospannung erzeugte Strom in Einzelmolekülkontakten mindestens zwei Größenordnungen kleiner als in Einzelatomkontakten, obwohl die Thermokraft von Molekülen etwa zehn mal größer ist.

Das erste Messkonzept benutzt einen fokussierten Laserpuls, um einen thermischen Gradienten innerhalb der Probe zu erzeugen. Aus diesem Ansatz ergaben sich zwei Probleme: Zum einen wurde der Kontakt durch die thermische Ausdehnung instabil. Zum anderen war es nicht möglich, Simulationen zu erstellen, die die Wärmeausbreitung in der Probe akkurat beschreiben, sodass man daraus die Temperaturdifferenz am Kontakt entnehmen konnte.

Aufgrund dieser Schwierigkeiten wurde das gepulste Konzept aufgegeben und ein neues statisches Konzept implementiert. Dabei wurde ein Platinwiderstand als Thermometer benutzt, der in der Nähe der Engstelle auf den Proben aufgebracht wurde. Eine Seite der Engstelle wurde mit einem fokussierten Laser geheizt, um die Temperaturdifferenz zu erzeugen. Mit Hilfe von Simulationen, die nur wenig von den genauen Materialparametern abhängen, konnte daraus die Temperaturdifferenz mit einer Unsicherheit von nur etwa 20 % ermittelt werden.

Um Einatom-Kontakte zu erzeugen, wurde die Goldstruktur der Probe bis zum Abreißen gestreckt. Wenn der Kontakt somit geöffnet war, konnte er wieder geschlossen werden, indem man die beiden Seiten wieder annäherte. Vor dem kompletten Abreißen und zum Beginn des Schließens ergibt sich ein atomarer Kontakt. Während des Öffnens und Schließens wurden der Leitwert und zeitgleich auch die Thermokraft gemessen. Außerdem wurden Kontrollmessungen ohne Heizen und mit Heizen auf der anderen Seite durchgeführt, um das Driften der Verstärker auszugleichen.

Auf diese Art wurde die Thermokraft von atomaren Goldkontakten zwischen 20 und 300 K gemessen. In guter Übereinstimmung mit anderen Messungen und theoretischen Berechnungen ist die Thermokraft bei tiefen Temperaturen klein und ein Wert von $-1.1\,\mu V\,K^{-1}$ ergab sich für Raumtemperatur. Zwischen diesen beiden Punkten durchläuft die Temperaturabhängigkeit der Thermokraft aber unerwarteterweise ein deutliches Minimum von $-2\,\mu V\,K^{-1}$ bei etwa 150 bis 250 K. Ein ähnliches nichtlineares Verhalten zeigt auch die Standardabweichung der Thermokraft.
Leider waren die Ergebnisse der Thermokraftmessungen an Terphenyldithiol nicht auswertbar, da die Auswirkungen der thermisch induzierten Verschiebung der Eingangsspannung des Strom-Spannungs-Wandlers alle von der Probe stammenden Signale verdeckten.
Auch wenn die Messungen an Molekülkontakten keine verwertbaren Daten ergeben haben, zeigt diese Arbeit dennoch einen funktionierenden Ansatz für Thermokraftmessungen. Das entwickelte statische Messkonzept umgeht dabei die Probleme der thermischen Ausdehnung und ermöglicht so, die inhärente Stabilität der lithographischen Bruchkontakte zu erhalten. Des Weiteren werden die Anforderungen an die quantitative Zuverlässigkeit der Simulationen reduziert, indem die zeitliche Dynamik nicht mehr benötigt und das Probendesign mit dem integrierten Thermometer gezielt gestaltet wird.
Die zuverlässigen und reproduzierbaren Messungen der Thermokraft von Goldkontakten demonstrieren die Umsetzbarkeit, die Genauigkeit und die Vorzüge dieser entwickelten Messstrategie.

# Abstract

This thesis shows thermopower measurements on atomic contacts of gold and tries on single molecule contacts in lithographic mechanically controlled break junctions. In structures that small the conventional transport theory is no longer applicable, instead quantum mechanical effects have to be considered. In molecules, these effects may be able to provide useful electronic properties beyond the known behavior of conventional electronic devices. For the understanding of the energetic placement of the highest occupied molecular orbital and the lowest unoccupied molecular orbital and of quantum interference in the electronic structure of the molecule, the thermopower provides valuable insight. These results can verify theoretical calculations of the molecules, which can be used to create tailored molecules for specific applications. Experimentally, this is challenging, since the generated temperature differences have to be small to avoid damaging the sample, therefore, the thermovoltage of gold contacts is in the range of only a few microvolt. In these single atom contacts, the resistances are below 20 kΩ. Single molecule contacts, however, have resistances between 10 MΩ and 1 GΩ. Hence, the current generated by the thermovoltage is at least two orders of magnitude smaller than in single atom contacts, even though the thermopower of molecules is ten times bigger.
A pulsed measurement concept with optical heating to generate a temperature gradient in the sample was used. The resulting problems are the mechanical deformation of the contact due to the thermal expansion and the unsuccessful tries of getting reliable information about the temperature difference from finite element simulations.
Due to these problems the pulsed concept was discarded and a new static heating concept was implemented. This concept uses a platinum resistance close to the constriction as a thermometer. This thermometer combined with simulations fairly insensitive to the material constants allow to derive the temperature difference across the junction to within 20 %. The temperature difference between the two sides of the junction is generated by heating one side with a focused laser.
To create single-atom contacts, the gold structure is stretched and at some point broken. Once a contact is opened, the two ends are brought together and thereby close the contact. Single atom contacts are achieved before the contact is completely open and at the beginning of the closing. During the opening and closing, the thermovoltage and the conductance are measured simultaneously, and control measurements without the heating and with the heating on the other side are integrated to compensate the offset drift of the amplifier and to exclude further offsets generated by the heating.
The resulting thermopower was measured at different average temperatures from 20 to 300 K. In agreement with existing measurements and theory, the thermopower was found to be small at low temperatures, and a value of $-1.1\,\mu V\,K^{-1}$ was found at room temperature. Between these two ends, the thermopower shows an unexpected pronounced minimum of $-2\,\mu V\,K^{-1}$ between 150 and 250 K. Similar nonlinear behavior could also be shown for the standard deviation of the thermopower.

Unfortunately, the thermopower measurements on the molecule terphenyldithiol were inconclusive since the thermal drifts of the electronics during the measurement masked any visible signal.
Even though no useful data for the thermopower of molecules could be measured, this thesis presents a working approach for a thermopower measurement. The static approach eliminates the problem of thermal expansion and preserves the mechanical stability inherent in the lithographic break junction concept. The requirements for accurate quantitative simulations of the heat distribution is also relaxed due to the different measurement principle, the strategic sample design, and the integrated thermometer.
The reproducible results for the thermopower of gold contacts demonstrate the feasibility, the accuracy and the benefits of using this measurement strategy.

# 1. Introduction

Observing the development of the microelectronics in the beginning of the 1960s Moore found that the number of transistors in a circuit doubles every two years [Moo65]. His prediction for the next ten years was for this trend to continue. This proved to be a self-fulfilling prophecy which would be sustained for another 45 years. However, the size of the circuits is fairly constant. Following this empirical law, the linear structure size in integrated circuits has to halve every four years. This miniaturization of electronics leads to ever smaller structures fabricated in a top down approach. The current state is 6 nm [TSM19] wide gates in so called FinFETs produced in research labs, and 14 nm wide structures being available in normal commercially available CPUs [Int19].
This trend cannot continue unhindered because the substrate and the structures are made of atoms, and the current structures are already less than one hundred atoms wide. In structures that small or even smaller, down to a single atom, the conventional macroscopic transport theory for electrical current is no longer applicable. Instead, the quantum mechanical properties have to be accounted for.
Taking the miniaturization idea to the extreme, the current through a single metal atom has been measured [Gim87] and the resulting resistive behavior is well established. For the use in active circuitry, however, this resistor-like characteristic is not very useful since it behaves like a passive component. Thus the idea of using single molecules instead of only the metallic atomic contacts has been proposed [Ree97]. This approach would combine the precise control of the atomic configurations of the chemical synthesis with the most extreme form of miniaturization possible. A properly designed molecule might be able to replace conventional elements like diodes [Elb05] or transistors [Oso08] but might also have completely new properties.
To understand, develop, and advance "rules" for the design of such molecules it is important to have a good theoretical understanding of the transport not only through few metal atoms, but especially through a single molecule. The theory which is based on DFT (density-functional theory) and NEGF (non-equilibrium Green's functions) is able to describe the molecular orbitals and make some predictions about the transport properties. To verify the latter, IV-characteristics can be measured. By having deviating electron occupation functions due to a temperature difference, the transmission can be probed in another way. The deviation in the electron occupation generates a voltage and, hence, the measurement of the ratio of the voltage and temperature difference, the so called thermopower, provides complementary information.
To approach this theoretically useful information, the voltage across the junction has to be measured, but also the temperature difference has to be known. The further is rather straight forward, whilst the latter requires a lot of effort. This work will show different ways to access the temperature difference and provide measurements of the thermovoltage of atomic contacts and also of molecules.

# 2. Theory and Background

Since this work is centered on the measurement of the voltage generated at a molecular or single atom junction by having a thermal gradient across that junction, I will start with the background for electrical transport in such structures followed by the expansion of the theory to incorporate different temperatures. Afterwards there will be a short summary of classical heat transport which is necessary for the understanding of the experimental work of getting a temperature estimate at the junction.

## 2.1. Electrical Transport Through Small Geometries

An in-depth description of the electronic transport in confined geometries and further references can be found in chapter 4 and 11.7 of [Cue10]. The following is a short summary that introduces the concepts relevant to my experiments.
In all classical systems, electric transport is described by Ohm's law:

$$\vec{j} = \sigma \vec{E} \tag{2.1}$$

with $\vec{j}$ as the current density, $\sigma$ as electrical conductivity and $\vec{E}$ being the electrical field. Assuming a constant electrical field, it can be seen that the electrical current depends on the area of the cross section of the electrical conductor. In resistance terms this means that the thinner the conductor, the higher the resistance.
This classical approach is correct as long as the dimensions of the system are bigger than the mean free path and the phase coherence length of the charge carriers. These conditions for the macroscopic regime are not met in the case of the structures analyzed in this work, since the aim is to have structures that have only a single or a few atoms in cross section. To describe these small junctions, one can assume the leads to be macroscopic "baths" where the charge carriers are moving diffusely with defined temperature and chemical potential, whereas the charge carriers move ballistically inside of small structures.
In the normal quantum mechanical picture, one would argue that the reduced dimensions result in a quantization of the $k$-vector leading to a density of states (DOS) different from the one in bulk material. While this is a correct view of the problem, I find it more descriptive to think of the charge carriers, which are electrons in most cases, being bound to the orbitals of the atomic structure. When these orbitals "overlap" with those of different atoms, the charge carriers can pass from one atom to the next. However, if the orbitals are spatially separated the charge carriers need to tunnel. This tunneling process is only possible with a certain probability, hence some of the charge carriers moving towards the junction are reflected and only some are transmitted.
In this picture it is intuitive to see that every possible path, a so called channel, of getting the charge carriers from one side to the other can be assigned a probability of transmission $T$ that can assume values between zero and one. Since the tunneling probability depends

on the DOS on both sides, it is also intuitive that the probability $\tau$ is a function of the energy $E$ of the charge carrier. The current across the junction can thus be written as

$$I = \frac{2e}{h} \sum_n \int_{-\infty}^{\infty} \tau_n(E) \left[ f_L(E) - f_R(E) \right] \, \mathrm{d}E \tag{2.2}$$

with $f(E)$ as the Fermi function on the left and right side, and $\tau_n$ as the transmission of the channel $n$. The constant $\frac{2e}{h}$ is valid for spin degenerate electrons or holes, and can be derived as shown in [Cue10, Chapter 4].
In the case of a metallic junction $\tau_n(E)$ is usually rather flat. If we also assume that the difference between the Fermi functions on both sides is only a shift of $eV$ generated by an externally applied voltage $V$, the current can be simplified to

$$I = \frac{2e^2}{h} V \sum_n \tau_n = G_0 V \sum_n \tau_n \tag{2.3}$$

with $G_0 = \frac{2e^2}{h} \approx 1/12\,906\,\Omega$ being the conductance quantum.
In molecules however, where the transmission $\tau_n(E)$ is not flat, this approximation is not adequate. If we assume that the temperature is low enough and the voltage is symmetric on the contacts (around the Fermi energy $E_F$), the expression can be simplified to

$$I = \frac{2e}{h} \sum_n \int_{E_F - \frac{eV}{2}}^{E_F + \frac{eV}{2}} \tau_n(E) \tag{2.4}$$

or even further to

$$I = \frac{2e}{h} \int_{E_F - \frac{eV}{2}}^{E_F + \frac{eV}{2}} \tau(E), \tag{2.5}$$

if we use $\tau$ as the sum of all channels, which is usually not necessary since in molecules normally only one channel is responsible for the transport. The "non-flatness" of $\tau(E)$ is the reason why molecular junctions can show significant nonlinearities in the IV-curves, whereas metallic junctions show a mostly linear behavior.
Up until now, we assumed that the diffusive baths in the two leads absorb the charge carriers. If one takes into account that not only the junction is able to reflect the charge carriers, but that they might also be reflected by elastic scattering at defects in the leads, this leads to unpredictable interference between the different possible paths for the electron (shown in figure 2.1). Since the speed and thereby the wavelength is dependent on the bias voltage, one can observe different transmissions depending on the bias voltage. This conductance fluctuation will be minimal when the transmission or the reflection of the junction is zero, since this eliminates some interference possibilities.

## 2.2. Thermovoltage

In the previous section we assumed in the simplification that the thermal energy $k_B T$ is small compared to the potential difference $eV$ created by the applied voltage. This

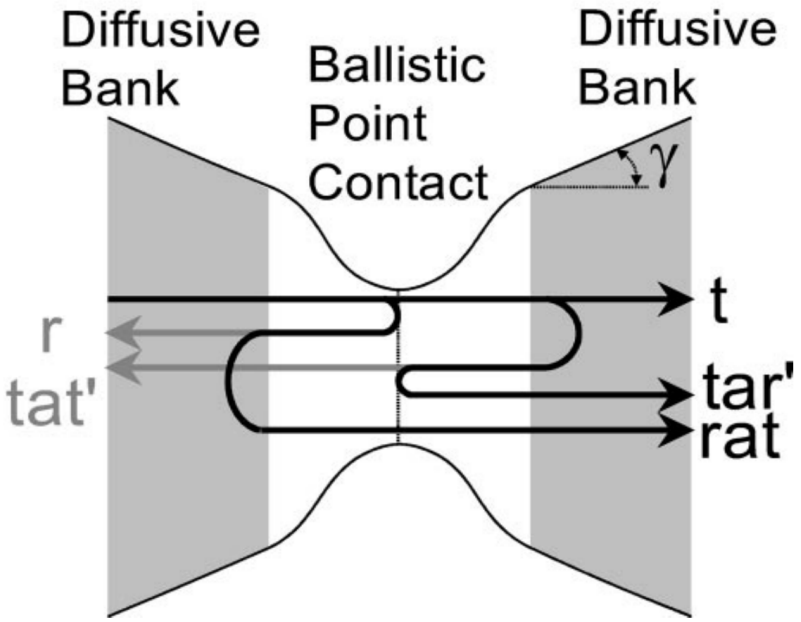



**Figure 2.1.:** An electron passing through a ballistic constriction may be scattered off the constriction or the diffusive banks. This leads to different transmission paths that interfere and thereby create a transmission probability that is dependent on the defects in the banks. These defects are uncontrollable and the result is a random fluctuation of the transmission. [Lud00, fig. 11.15]

assumption is not applicable for the following, since the thermovoltage can be measured without any external bias voltage.
Taking equation (2.2) and rewriting it for different temperatures instead of different external potentials, one gets

$$I = \frac{2e}{h} \int_{-\infty}^{\infty} \tau(E) \left[ f_L(T) - f_R(T) \right] \, \mathrm{d}E, \tag{2.6}$$

using $\tau(E)$ as the total transmission of the junction. The effect of the different temperatures as illustrated in figure 2.2 is that the edge of the Fermi distribution on the hotter side is more smeared than on the cold side. This gives rise to transport in two different regions: below the Fermi energy the cold side has more populated states, therefore a current towards the hotter side will appear; above the Fermi energy, however, the hotter side is more populated and a current towards the cold side will appear. The measurable net current is the sum of these two currents and is mainly generated by the difference in transmission between the energies slightly below and above the Fermi energy. Hence the current generated by a temperature difference depends on the derivative of the transmission at the Fermi energy.
Experimentally it is more common to measure the voltage instead of the current generated by the temperature difference. This voltage is the result of the previously described current being compensated by an opposing current generated by an electrical field, since there cannot be any current flow across the junction in the steady state. The easiest way to calculate the voltage is by dividing the current by the conductance of the junction.

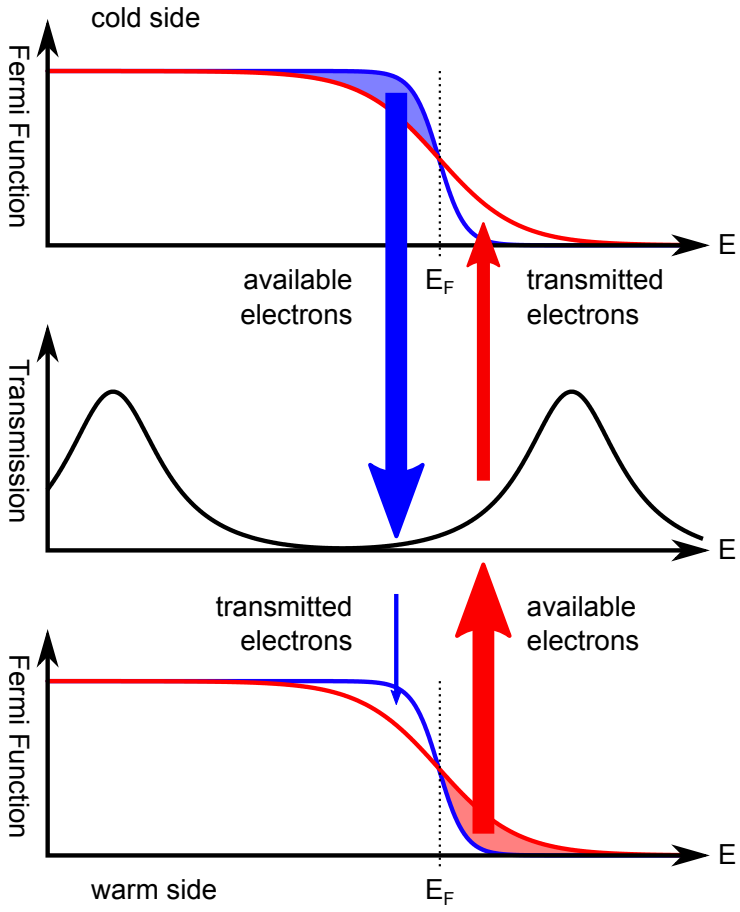



**Figure 2.2.:** The different width of the Fermi distribution due to the temperature difference results in the transmission of electrons where the distributions differ. These electrons are transmitted with the probabilities given by the transmission function at their energies. This results in a net current corresponding to the difference of the transmission on both sides of the Fermi energy $E_{\mathrm{F}}$. The difference is the same as the derivative of the transmission if the temperature differences are sufficiently small.

This simplifies the equation because the conductance is proportional to the transmission $\tau(E)$. To calculate the voltage this way is an approximation that only works for small voltage differences.
Since the voltage is increased by a bigger temperature difference, it is useful to specify the thermopower

$$S = -\frac{V}{\Delta T} \tag{2.7}$$

instead of the voltage. The full expression for the thermopower can be derived as

$$S = -\frac{\pi^2 k_B^2 T}{3e} \frac{(\frac{\partial \tau(E)}{\partial E})_{E=E_F}}{\tau(E_F)} \tag{2.8}$$

with $T$ being the average temperature [Cue10, chapter 4 and 19.3]. As explained above, this thermopower depends mainly on the ratio of the derivative of the transmission and the transmission itself. Since the transmission can be directly taken from the IV-characteristic, the slope of the transmission can be calculated from measurements. The expression for the thermopower is mathematically identical to the logarithmic derivative of the transmission function with respect to energy.

$$S = -\frac{\pi^2 k_B^2 T}{3e} \left( \frac{\partial \ln(\tau(E))}{\partial E} \right)_{E=E_F} \tag{2.9}$$

In molecules the electrical transport runs through either the highest occupied molecular orbital (HOMO) or the lowest unoccupied molecular orbital (LUMO). Thus the transmission is maximal if the Fermi energy is aligned with either orbital. As the Fermi energy in the molecular junctions is located between the energies corresponding to these two orbitals, the transmission here is lower than if the Fermi energy matched the energies of the orbitals. Thus the thermopower of a molecular junction can be used to determine which orbital is responsible for the electronic transport, as illustrated by figure 2.3. Using a similar mechanism as explained for the conductance fluctuations, the thermopower also shows fluctuations when the channels are only partially open. For measurements on molecules however, the transmission is usually very small and this effect will therefore be negligible.

## 2.3. Bulk Thermovoltage in Metals

Thermovoltage is also a well known phenomenon in bulk metals and not limited to ballistic contacts. In bulk, there is no transmission function $\tau(E)$ to break the symmetry. But still, the hot electrons have a wider energy distribution than the cold electrons. This means that electrons with energies above the Fermi energy diffuse into empty states on the cold side, while electrons slightly below the Fermi energy are more prevalent on the cold side and will diffuse to the hot side. Any imbalance in the two effects like e.g. energy dependent scattering processes can therefore lead to a net current.
Another effect that can be seen in bulk is called phonon-drag. A temperature difference along the conductor generates a net flow of phonons from the warm to the cold part.

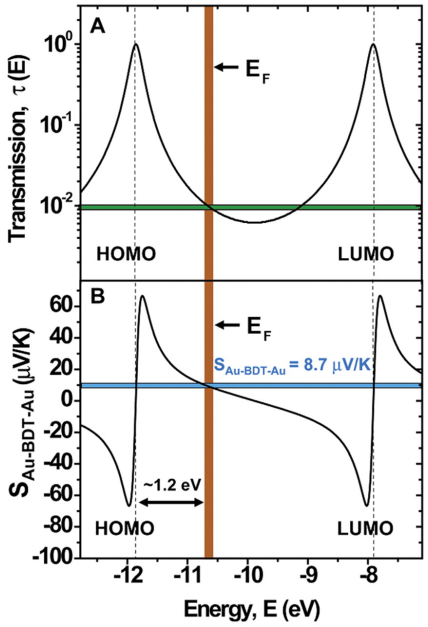



**Figure 2.3.:** Since the transmission is maximal if the Fermi energy aligns with the HOMO or the LUMO, the sign of the thermopower which is the logarithmic derivative of the transmission can be used to determine, whether the Fermi energy is closer to the HOMO or the LUMO. This indicates which orbital is dominant for the electric transport. [Red07, fig. 3]

These phonons can transfer some of their momentum to the electrons by electron-phonon-coupling. This contributes to the electronic current.
The total net current will create charged regions at the hot and cold end. These charges build up until their electric field is strong enough to stop the net current. The potential difference between the two ends is the absolute thermovoltage of the material.
It is worth noting that these voltages cannot be measured directly since the temperature difference will also generate thermovoltages in the measuring wires. Hence, a voltage can only be measured if materials with a different thermopower are used so that the thermovoltages of the bulk is not completely compensated by the thermovoltages in the measuring lines.

## 2.4. Classical Thermal Conductivity

In order to generate a temperature difference to measure any thermovoltage, it is necessary to understand the temperature distribution inside the sample structure. The temperature distribution can be described by classical theory if the sample structures are much larger than the mean free path of the phonons. In my sample this requirement is violated in two aspects: First the constriction itself has the size of a few atoms and therefore has to be described by non-classical theory [Pau11]. Here the thermal conduction is very small compared to thermal conductivity through the substrate, therefore it can be neglected. Second the films are too thin to be described by classical theory in all directions. However,

the thermal transport in the thickness of the film is much faster than the timescales of my experiment and does not need to be simulated. In the lateral directions, the effect of the reduced thickness and thus increased scattering of the phonons can be accounted for by using a reduced thermal conductivity compared to bulk values found in the literature. Considering this reduced thermal conductivity, the complete sample can be described with classical theory.
The temperature distribution $T$ can be described by the diffusion equation

$$\dot{T} = \frac{1}{c_p \varrho} \left( k \nabla^2 T + q \right), \tag{2.10}$$

where $k$ is the thermal conductivity of the material, $c_p$ is the specific heat at constant pressure and $\varrho$ is the density, $\nabla$ is the nabla operator and $q$ describes an external heat source.
The equation can be solved analytically for some simple systems, but in my case I use a finite element simulation to calculate a grid based numerical approximation. Hereby, I can include the correct geometry, the different materials of the sample, and the additional thermal resistance when heat transfers from one material to the other. This resistance is known as Kapitza resistance, which was found to be very prominent in a liquid helium to metal interface [Kap41]. Even though my sample has no interface to liquid helium, the effect can still be found, since it is based on the phononic mismatch of two materials at an interface. Although it is theoretically possible to calculate the portion of phonons that are reflected at such an interface, the experimental value will depend strongly on the microscopic structures of the interface. The Kapitza resistance is therefore very hard to control and thus is a free parameter in my simulations.

# 3. State of Research

## 3.1. Metallic Contacts

The first measurements of the thermopower across a small atomic junction were done in the late nineties by Ludoph and van Ruitenbeek [Lud99]. Their setup was a cryogenic notched wire break junction with heaters and thermometers on both sides. The heaters and thermometers were used to generate and measure the temperature gradient across the junction. Some of the heat will flow through the substrate and therefore the thermometers will not be able to measure the temperature directly at the junction. Instead they used the large-contact thermopower to determine the ratio between the measured temperatures and the temperature difference at the junction.
They could observe that the thermopower changes when a conductance change and an atomic reordering happens, but the values and the sign of the thermopower are random. This is shown in figure 3.1. Using several opening and closing traces, they created a density plot, showing the thermopower as function of the conductance. This plot is reproduced in figure 3.2 and shows that at an average temperature of about 12 K no average thermopower can be measured.
Similar results were obtained by Matsushita and coworkers in [Mat15]. They also used a notched wire break junction setup but they performed the measurements at room temperature and slightly elevated temperatures.
Their results are shown in figure 3.3. This figure shows that the average thermovoltage and thermopower are close to zero.
However, there are other measurements showing an average thermopower different from zero.
Tsutsui and later also Morikawa used a lithographic mechanically controllable break junction (MCBJ) with an integrated platinum heater to generate the temperature difference. They were able to measure distinct non zero thermovoltage (see figure 3.4) which they converted to a thermopower of between $-2$ and $-10\,\mu V\,K^{-1}$ [Tsu13].
They derived the temperature difference across the junction from the lifetime of the $1\,G_0$ plateau. This is an elegant solution as it allows for a local temperature estimation at the junction. However, the lifetime can also be influenced by other parameters and, furthermore, the connection between the lifetime and the temperature difference is based on unproven assumptions. Thereby this analysis is a source for significant uncertainties.
The authors claim that the non-zero thermopower is due to the fact that the contacts were not deeply reformed during the closing and thus the structural defects in the vicinity of the junction remained undisturbed. Later [Mor14] attributed the non-zero thermopower to the bulk thermopower inside of the heated lead.
Evangeli and coworkers used a scanning tunneling microscope with a heated substrate to generate a temperature difference between the substrate on the one side and a thermally anchored tip on the other [Eva15]. Using the piezo-controlled motion, the tip can be de-

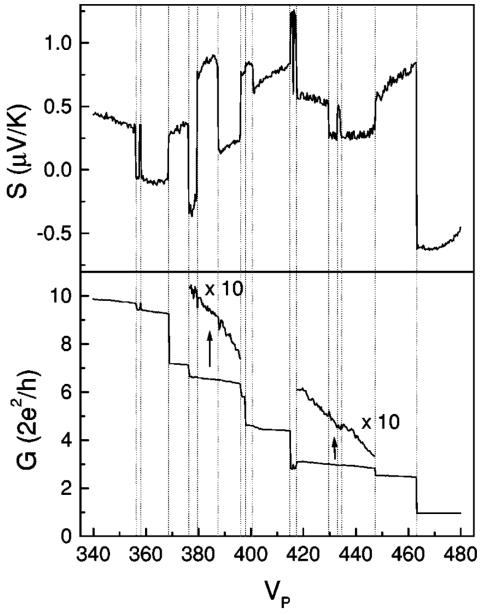



**Figure 3.1.:** Simultaneous measurements of conductance and thermopower of gold at 12 K as the contact is opened by a piezo. The piezo voltage $V_P$ corresponds to the stretching of the contact. It can be seen that changes in the contact which result in only small differences of conductance can still create large and random changes in the thermopower. [Lud99, fig. 2]

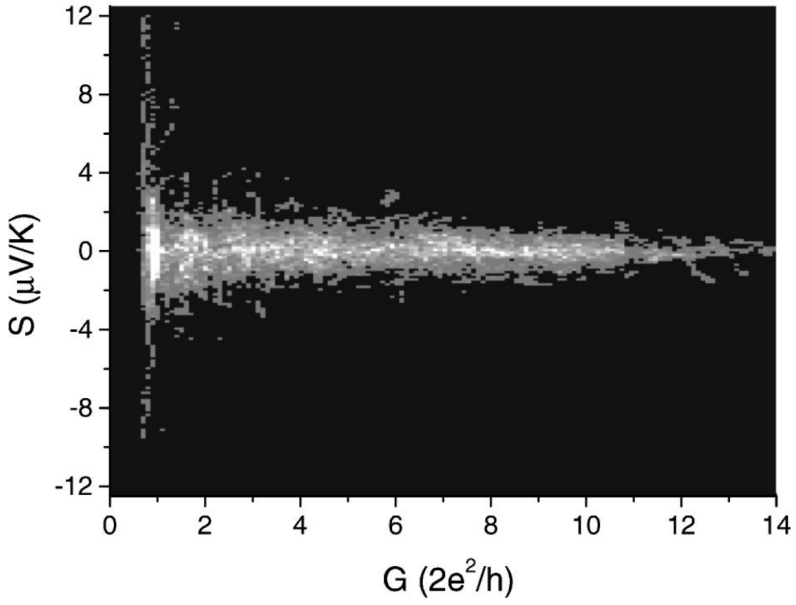



**Figure 3.2.:** This density plot shows the thermopower as a function of conductance indicating that there is no average thermopower. However, the fluctuations increase with lower conductances and are minimal at integer conductances. [Lud99, fig. 3]

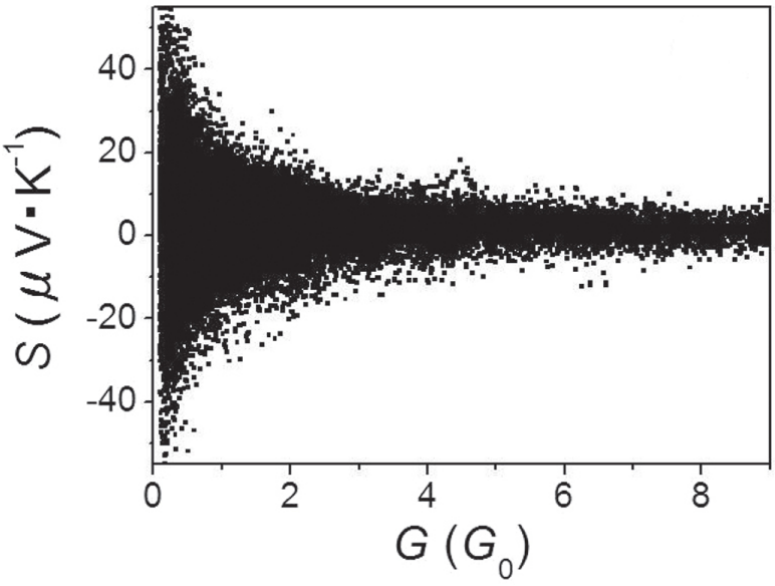



**Figure 3.3.:** This density plot is generated by many opening traces of a gold contact at room temperature. It shows the clear increase of the fluctuations with decreasing conductance, but it shows no average thermopower; the reduction of the fluctuations at integer conductance quanta is also not visible. [Mat15, fig. 3a]

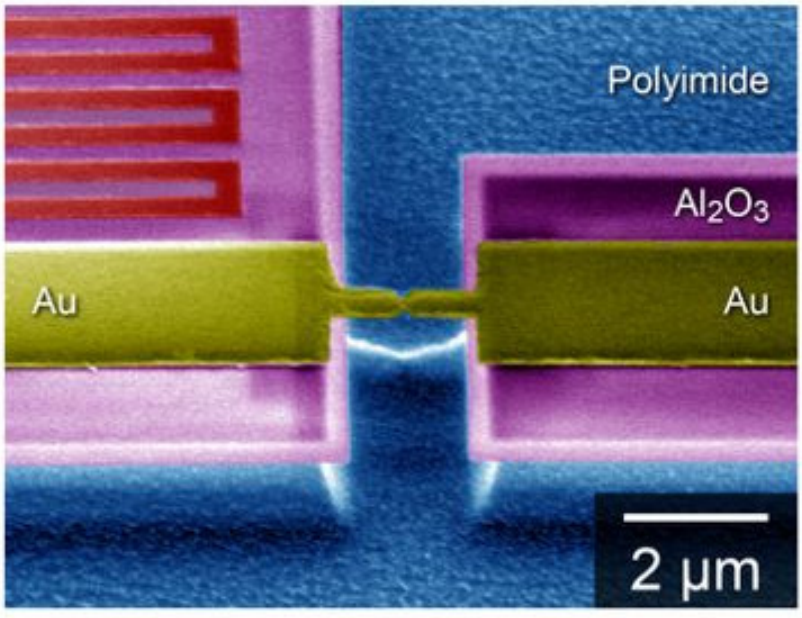



**Figure 3.4.:** This micrograph shows the structure of the lithographic break junction with the integrated platinum heater. [Tsu13, fig. 1b]

liberately crashed into the substrate and then be pulled out to generate atomic contacts. Their results are shown in figure 3.5 demonstrating clearly the non zero thermopower of $-0.75\,\mu V\,K^{-1}$. Here the argument that the defects were preserved cannot be used since the data is generated from 134 breaking contacts starting with conductances in the range of $1 \cdot 10^4$ to $1 \cdot 10^5\,G_0$. Therefore these measurements do not necessarily contradict the results of Tsutsui [Tsu13].
The research shows that the thermopower of an atomic gold contact is negligible at low temperatures and in the range of $-1\,\mu V\,K^{-1}$ at room temperature.

## 3.2. Molecular Contacts

Morikawa and coworkers also measured the thermopower of the molecule benzenedithiol (BDT)[Mor14]. This is one of the common test molecules since it shows conductance plateaus in the range up to $1 \cdot 10^{-2}\,G_0$. They found a predominantly negative thermovoltage as shown in figure 3.6 that results in a positive thermopower.
However, they also found that the thermovoltage switches the sign depending on the heating of the contact. They attributed this to the change in geometry due to the thermal expansion of the gold contacts.
Reddy and coworkers used a modified ambient condition STM setup with a heated substrate to measure not only BDT but also the dibenzenedithiol (DBDT) and tribenzenedithiol (TBDT) with a molecular structure similar to BDT except for the additional benzene rings [Red07]. They performed rigorous measurements at various temperature differences with and without the molecules, resulting in thermopowers of $8.7(21)\,\mu V\,K^{-1}$ for BDT, $12.9(22)\,\mu V\,K^{-1}$ for DBDT and $14.2(32)\,\mu V\,K^{-1}$ for TBDT.
Baheti and coworkers continued the work and investigated the influence of substituents on the BDT molecule. They found that compared to BDT with $7.2(2)\,\mu V\,K^{-1}$, 2,5-dimethyl-BDT shows $8.3(3)\,\mu V\,K^{-1}$, tetrafluoro-BDT $5.4(4)\,\mu V\,K^{-1}$, and tetracloro-BDT $4.0(6)\,\mu V\,K^{-1}$ [Bah08]. They thereby demonstrated that side groups and substituents shift the HOMO and LUMO levels, leading to a different thermopower according to their electron affinity. Changing the linking groups from thiols to cyanides has a strong effect on the thermopower, too: benzenedicyanide has a thermopower of $-1.3(5)\,\mu V\,K^{-1}$. This implies that the conduction is no longer dominated by the HOMO but the LUMO.
Miao and coworkers further measured the effects of interference inside a molecule by using the setup to compare meta- and para-oligo(phenylene ethynylene) (OPE). Meta-OPE has a thermopower of $20.9(154)\,\mu V\,K^{-1}$, whereas para-OPE only has $10.8(95)\,\mu V\,K^{-1}$ [Mia18]. These STM measurements were confirmed by measurements on ensembles which are more precise.
Guo and coworkers used a similar setup and measured the thermopower of 1,4'-butanedithiol as $2.11(11)\,\mu V\,K^{-1}$, 1.6'-hexanedithiol as $5.55(13)\,\mu V\,K^{-1}$, 4,4'-biphenyldithiol as $7.92(14)\,\mu V\,K^{-1}$, 4-4'-dimercaptostilbene as $8.35(23)\,\mu V\,K^{-1}$ and 4,7-dithiophenyl-2,1,3-benzothiadiazole-3',3"-dithiol as $15.46(15)\,\mu V\,K^{-1}$ [Guo13].
Using a similar setup as Evangeli, to be specific: an STM with a heated tip, Yzambart and coworkers [Yza18] measured the thermopower of several 2,7-dipyridylfluorene molecules with different substituents at the central carbon atom. They found that the thermopower can be changed from $-5.5\,\mu V\,K^{-1}$ to $9.0\,\mu V\,K^{-1}$ while maintaining the same conductance.

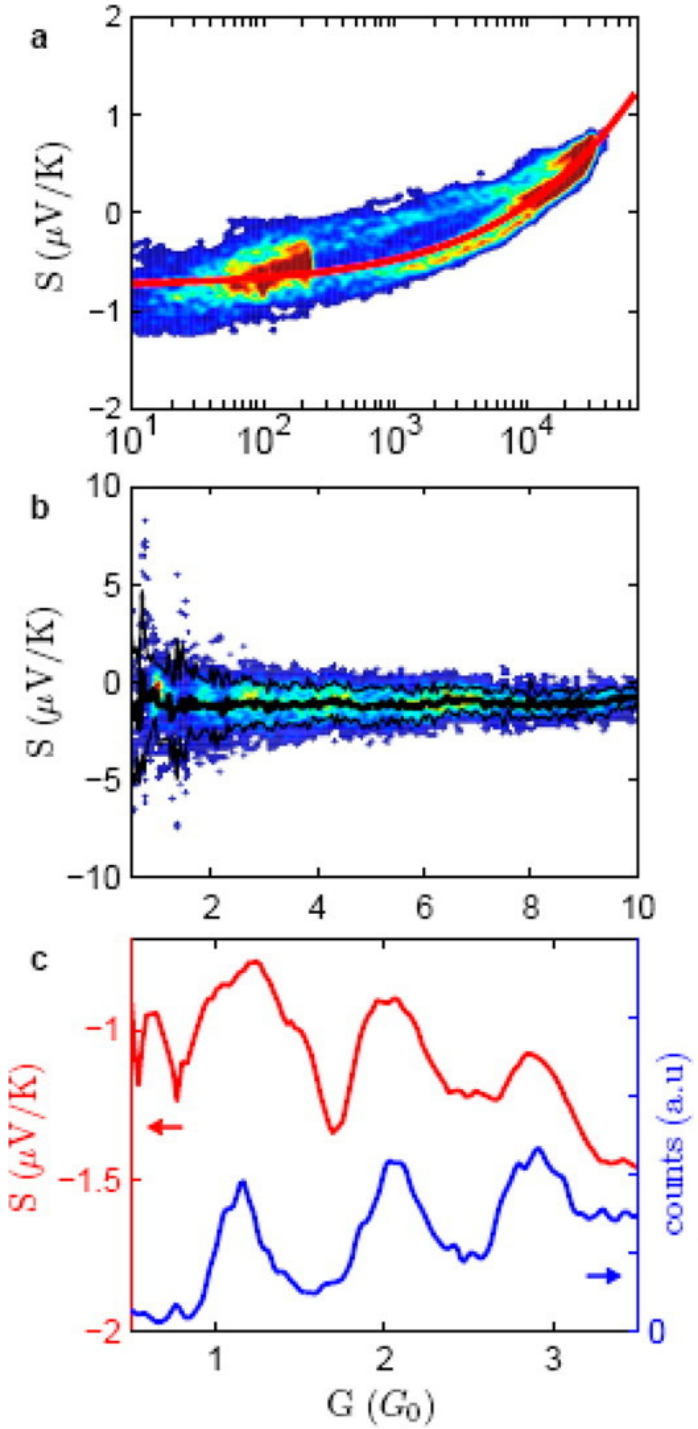



**Figure 3.5.:** The density plots (a and b) show the thermopower as function of conductance for several opening traces of a gold contact at ambient conditions. The graph in panel c shows the average thermopower clearly indicating a negative average thermopower. [Eva15, fig. 2a-c]

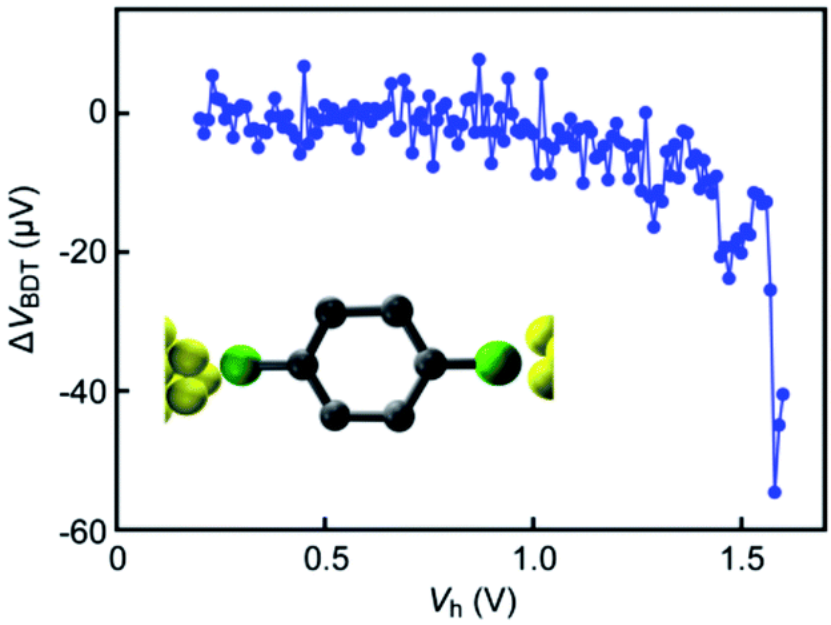



**Figure 3.6.:** This shows the thermovoltage of a gold-BDT-gold junction as function of the heating voltage applied to the platinum heater. Ideally one would expect a linear correlation between the heating and the generated thermovoltage with the slope being the thermopower. However, the measurements show that the relation is nonlinear, the thermovoltage fluctuates and even changes the sign. Morikawa attibutes the fluctuations to the effects of thermal expansion which changes the contact distance and geometry as the heating power is increased. [Mor14, fig. 7b]

Widawsky and coworkers measured different molecules in an STM with a heated substrate around room temperature. They could show that the thermopower can be tuned from $-12.3\,\mu V\,K^{-1}$ to $13.0\,\mu V\,K^{-1}$ with the proper choice of molecules [Wid11]. Further research of them showed the thermopower of alkanes and oligophenyls where the carbon chain is covalently bonded to the gold electrodes [Wid13].
For molecules, the research shows that the thermopower at room temperature can be tuned in a wide range. Depending on the molecule, the linking group, and the side chains, common thermopowers are in the range of $-15\,\mu V\,K^{-1}$ to $15\,\mu V\,K^{-1}$.

# 4. Measurement concept

To measure any property of an atomic or molecular contact we need such a contact. In my experiment, this is realized in a lithographically fabricated mechanically controllable break junction. Using electron beam lithography, I can create a thin gold film with a narrow constriction on top of a substrate with an isolating layer. An etching step cuts into this layer and removes the support from underneath the narrow gold part. This complete sample is bent in the setup in a controlled way. Since the gold film is on top of the substrate it gets stretched and at some point it breaks. This break occurs at the constriction in the film because this part is the weakest and it is not supported by the isolation layer.
The geometry of the sample structure and the depth of the etching achieves a large ratio between the movement of the sample bending mechanism and the stretching of the constriction. This ratio is sufficiently large that the bending is performed by conventional mechanical parts but the stretch can be controlled to atomic distances. This makes the constriction rather stable against vibrations.
When measurements of single atom contacts are performed, the bending is stopped if the electrical conductance of the junction is in the correct range. For molecular contacts, the sample is controlled the same way, however, the molecules have two linking groups that are attached to the gold. With a bit of luck one molecule can have the linking groups attached to the different sides of the junction which makes the molecule bridge the gap between the contacts when the gold chain rips. In this state, measurements on a single molecule can be performed.
In my case the measured quantity is the thermopower of an atomic or molecular junction which is a two part problem: The voltage and also the temperature difference at the junction have to be measured. Also the thermopower is dependent on the electrical conductance of the junction, so it needs to be measured as well.
The measurement of the voltage is experimentally challenging, because the voltages generated by the thermopower are small compared to the offset voltages of the amplifiers and their drift. This problem is solved by including reference measurements into the procedures.
The measurement of the temperature difference across the constriction is even harder, since the length scales of the constriction are too small to use conventional temperature sensors. To solve this problem, the sample structure is designed such that indirect temperature estimations, supported by heat flow simulations which are calibrated to describe the sample behavior.

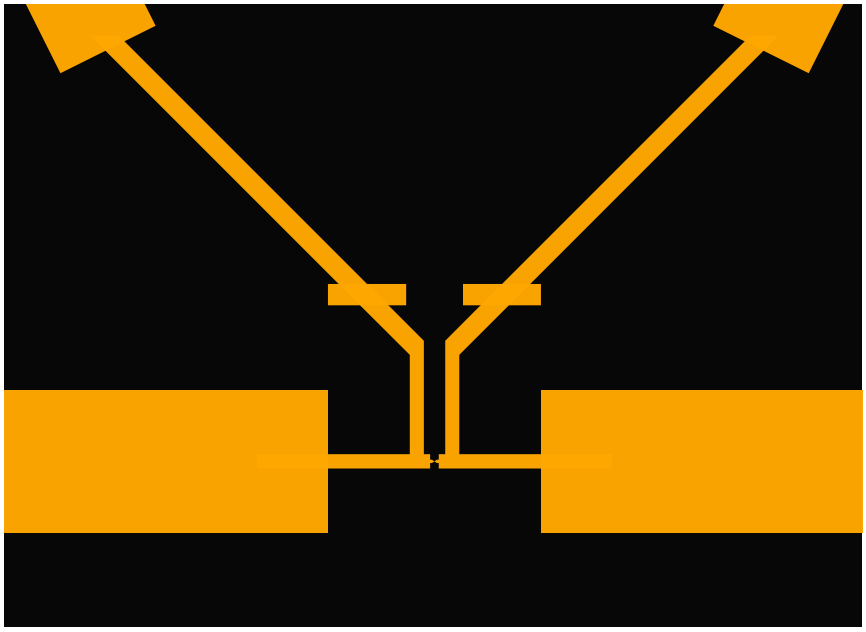


**Figure 4.1.:** Schematic of the micrometer scale structure of the break junction sample used for the pulsed measurements with a gold structure (orange) on the substrate (black)

## 4.1. Pulsed Measurements

In the pulsed measurement approach the sample is a lithographically fabricated gold break junction on a black plastic substrate. To heat the sample asymmetrically, a modulated laser diode generates short pulses with a duration of 4 ms every 34 ms, which are focused on the sample substrate on one side of the constriction. These pulses are short enough that the heat does not travel more than 100 µm which serves a dual purpose: It creates a strong temperature gradient across the constriction, which would not be possible on larger length scales and it simplifies simulations since only the central part of the sample needs to be calculated. Only a single laser pulse needs to be considered in the simulations, because after 34 ms the heat has diffused into the substrate far enough, that it can be approximated by a homogeneous background.
The lithographic structure (see figure 4.1) has four contacts to the constriction. They are used to perform four-point measurements for a precise evaluation of the resistance of the constriction, and, additionally, the heat induced resistance change in each lead is measured. This resistance change can also be extracted from the heat flow simulations and allows to check if the sample is accurately modeled.
Using the pulsed heating means that the electronics has to be fast enough to resolve the time evolution of the signals. But it also solves the problem of separating the amplifier offset voltage from the thermovoltage since the time before the pulse can be used as a reference signal containing only the offset voltage.
Unfortunately the pulsed concept also leads to thermal expansion during each pulse which prevents the formation of stable contacts. This problem can be avoided when performing static measurements.

## 4.2. Static Measurements

In the static measurements a lithographic gold break junction on bronze substrate is used. The focused laser heats the gold lead on one side of the constriction and creates a temperature difference between the two sides, since the bronze substrate works as a heat sink that prevents significant heat flow to the other side of the constriction. This means that a temperature gradient can be sustained throughout a complete opening and closing cycle. Successive cycles are performed, where every second cycle has no heating and serves as reference measurement. This is possible since the amplifier's offset voltage drifts significantly only on timescales which are much longer than the opening and closing cycles. After every second cycle, the laser is moved to the other side of the constriction. This inverts the temperature distribution. Thus the measured thermovoltage will be inverted and the resistance thermometer can be used to measure the temperature of the cold side of the junction. This temperature can differ from the sample chamber temperature since the sample is only coupled to the chamber by the bending mechanism and the remaining gas in the chamber.
A lock-in-amplifier generates the voltage to drive the junction via a high impedance voltage source. Measuring the voltage at the junction and the current through the junction, the lock-in signal gives the conductance. The offsets of the modulated signals are extracted by averaging over time. They contain the generated thermovoltage.
On top of one lead to the junction is an electrically isolated resistance that provides temperature information. Placing the resistance on top of the structure means that the resistance is only coupled to the gold structure underneath via the isolation layer and not to anything else. In a static temperature distribution, the thermometer will thus be at the same temperature as the gold, even though the electrical isolation layer prevents good thermal coupling, because there can be no static heat flow through this layer, and thus no temperature difference. If the resistance were between the gold structure and the substrate, that means underneath the gold, the static heat flow from the heated gold to the cooled substrate would pass through the isolation layer. This would generate a temperature difference through the isolation layer and the resistance would be at a lower temperature than the heated gold structure.
The resistance thermometer on top of the gold structures leads to a more complex sample design shown in figure 4.2.

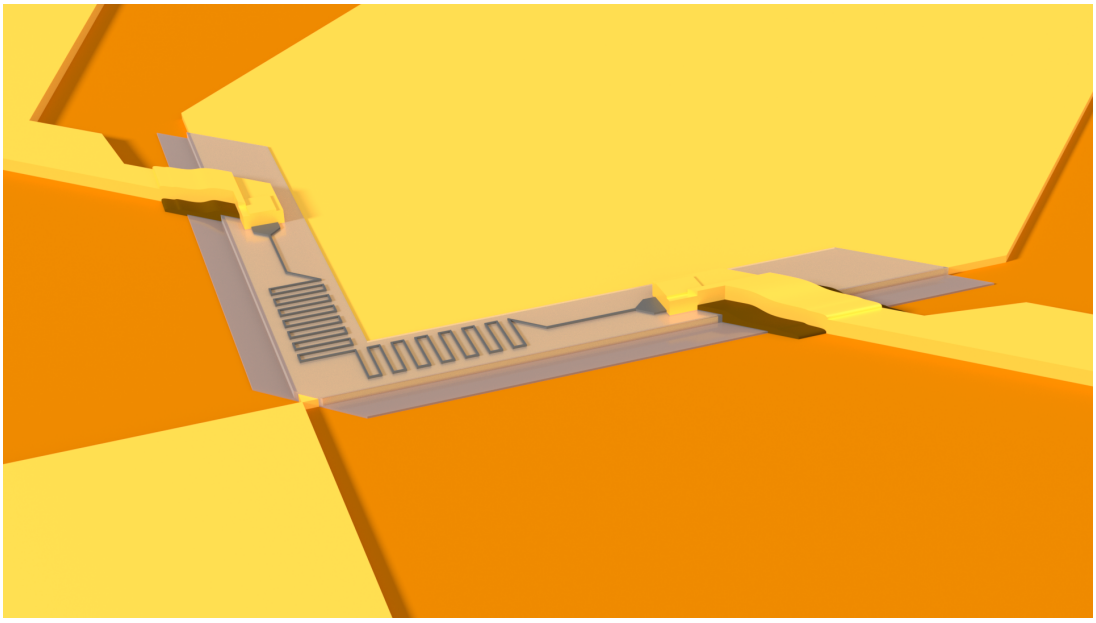


**Figure 4.2.:** Artistic rendering of the micrometer scale structure of the static break junction samples (substrate: orange, gold: yellow, isolation: brown and transparent gray, resistance: dark gray)

# 5. Setup

In order to measure the thermovoltage of a MCBJ when heated with a laser at cryogenic temperatures, you need a cryogenic setup, with mechanics for breaking the junction, optical access and additional optics for focussing the laser on the sample, and wiring and electronics to perform the voltage and conductance measurements.

## 5.1. Cryogenics

My setup is built around a "Janis Model SVT Research Cryostat" system from Janis Research Company equipped with a cryogenic variable temperature insert as shown in figure 5.1. This means that my sample chamber is not immersed in the liquid coolant, but is cooled by a gas flow from the vaporized liquid helium. This gas flow can be heated to any temperature between 2 and 300 K. The sample chamber is made of copper to provide a homogeneous temperature on the inside where the sample is kept in vacuum with a small amount of exchange gas. A thermometer (Lakeshore Cernox CX-1070-CU-HT) calibrated down to 4 K and a home built heater with 200 Ω resistance which provides up to 2 W of heating power inside of the sample chamber can be used to control and stabilize the temperature.
The dewar is equipped with optical windows providing optical access to the sample chamber from four directions. The sample chamber is also equipped with windows seated in indium to ensure a gas-tight seal while maintaining enough flexibility to prevent the glass from cracking due to the different amounts of shrinkage in the metal and the glass while cooling the chamber.
However, these windows are still susceptible to thermal shocks and therefore the complete system, not only the cryogenic insert, must be cooled down and warmed up.
The system is equipped to work with liquid helium and can be cooled down to about 2 K by pumping. For the temperature range needed in most of my experiments, however, it was enough to fill the liquid helium bath with liquid nitrogen and cool the sample chamber by the coupling to this bath without any gas flow.

## 5.2. Mechanics

The core of a MCBJ setup is the bending mechanism. In my setup this bending is controlled by a dc motor mounted on top of the cryogenic insert in ambient conditions. Attached to this motor is a long axle with a long cross shaped cross-section at the end. This cross section engages a fine pitched screw that is mounted to the top of the sample chamber. The screw pushes on a ball which isolates the rotation while still pushing on a spring loaded rod. At the end of the rod, there is a wedge that moves the sled. This wedge pushes against a roller mounted in the sled. This moves the sled but also keeps the

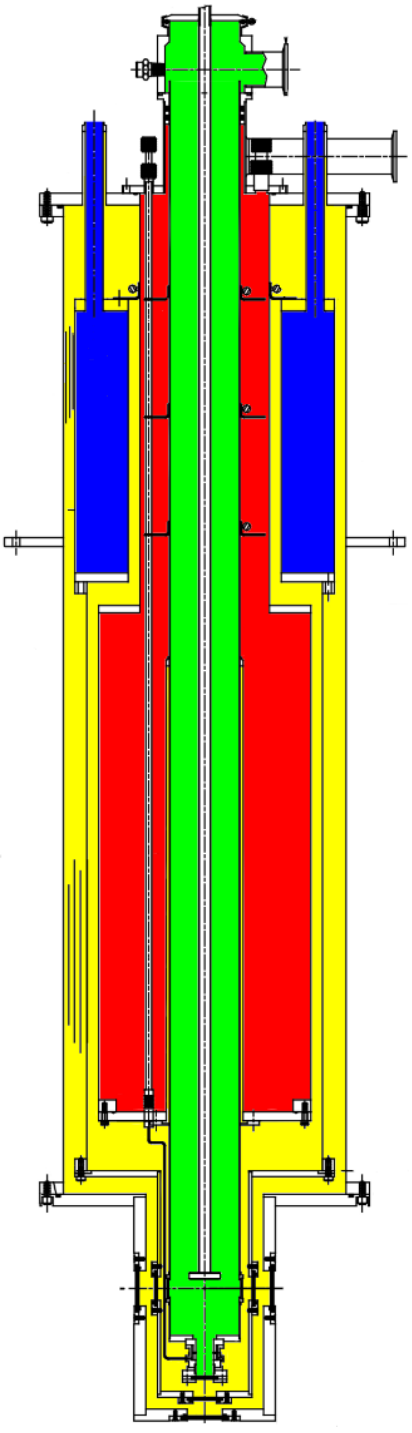


**Figure 5.1.:** Schematic drawing of the optical cryostat system with (yellow) isolation vacuum, (blue) nitrogen bath, (red) helium bath and (green) heated gas around the sample chamber[Jan]

wedge in position. The rod at the top is designed to be small enough that the bearing surfaces do not restrict the rotation, they only fix the position at the top. The bottom is fixed between the roller and a linear ball bearing, which is also designed to constrain any sideways motion. The wedge is wide enough that the rotation of the whole piece is constrained by the contact with the roller.
The sled is mounted on balls acting as bearing surfaces and translates the vertical movement of the wedge into horizontal movement of the sled. The roller that the wedge pushes against, makes sure that the vertical motion is not imparted on the sled. A cutaway drawing of the mechanics is shown in figure 5.2.
The sample is mounted between a rigidly mounted wedge and supports on the outside of the sample. The supports are mounted on the sled so that the sample is bent when the sled is moved. The center of the sample stays in position, which is required for the optics described in section 5.4.
The axle driving the mechanics is also connected to a big threaded rod outside the cryostat. An indicator rides on this rod to provide direct visual control of the position of the mechanics. This indicator has several centimeters of travel so that it interacts with two adjustable switches directly tied into the motor controller. The switches act as limit switches and prevent the motor from moving into positions where the mechanics would be damaged.

### 5.2.1. Design of the Bearing Surfaces of the Sled

To prevent the mechanics from getting stuck, the bearing surfaces of the sled are designed to limit only the necessary degrees of freedom (DOF). An over constrained system can only work reliably when the manufacturing precision is better than the bearing tolerances, especially since cryogenic mechanics experience significant thermal shrinking and cannot be lubricated by normal means.
Looking at the sled from behind (see figure 5.3) the left bottom groove in the support has a corresponding groove in the sled. This means that the two balls riding in the groove are limited in translation to only the groove direction. The sled has a similar groove which leads to four contact points with the balls. The four contact points limit four degrees of freedom, constraining the movement of the sled to only the motion and rotation along the groove axis.
The right bottom groove is similar in the support and will also limit the balls to move only along the groove, but the sled part of the bearing surface is flat. This means that the balls have only one point of contact constraining the remaining rotational DOF. As a result the sled will only move along the axis of the bottom left groove.
Since the balls only work as constrains if the sled is actually touching them, there is a spring loaded plate above the sled pushing against balls riding in grooves on top of the sled. This plate is flat and can move in all directions because it is only held by the springs. Therefore it provides only a downward force and does not limit any DOFs.
A remaining problem of this mechanical concept with the motor on the top is that, when the motor turns the screw, the necessary torque will be carried by the cryogenic insert which will wind slightly. This twists the sample chamber which moves the laser focus relative to the sample. Two concepts were applied to reduce this effect. The cross shaped cross-section of the screw in the top of the mechanics is designed to be much wider than

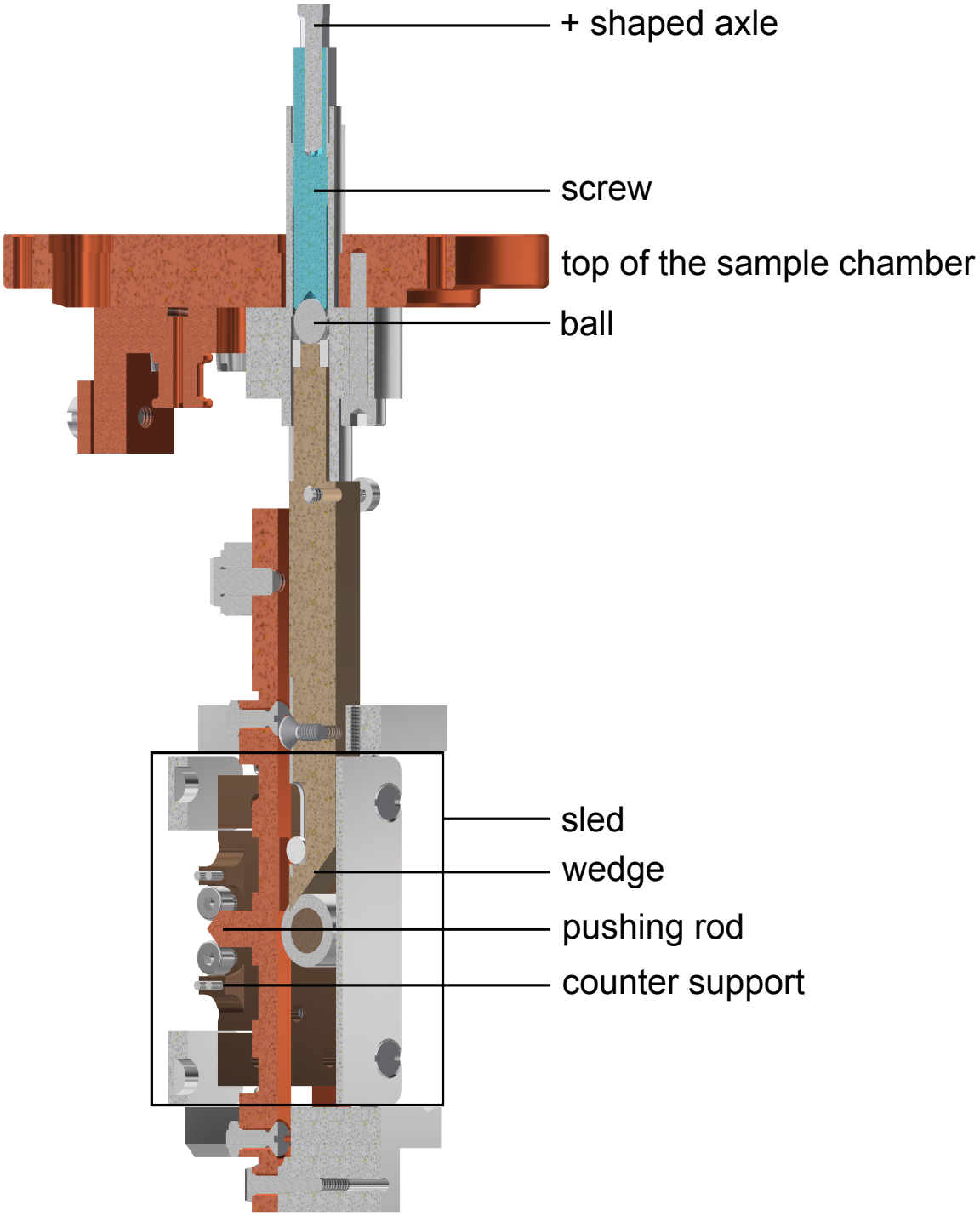



**Figure 5.2.:** Cutaway of the bending mechanics: The rotation of the cross shaped axle turns the screw. The screw pushes against the ball which pushes against the wedge. This moves the sled to the right side.

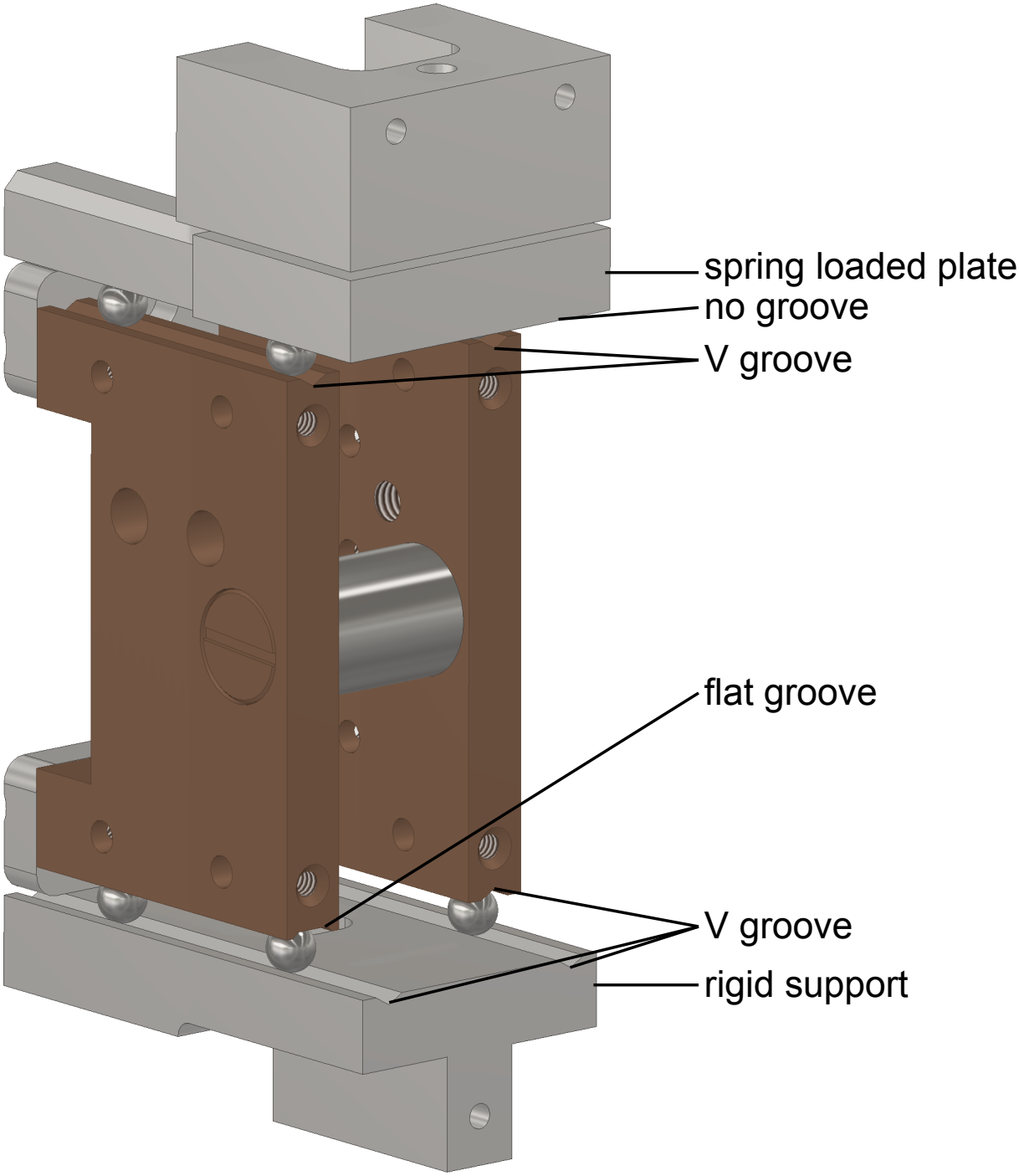



**Figure 5.3.:** Support of the sled: The sled is supported by balls in grooves that are designed to limit five of six degrees of freedom. Every degree of freedom is only limited once. The balls on top of the sled limit no degrees of freedom, the two balls in the right bottom track have four contact points limiting four degrees of freedom, and the flat profile in the left track limits the remaining rotational degree of freedom.

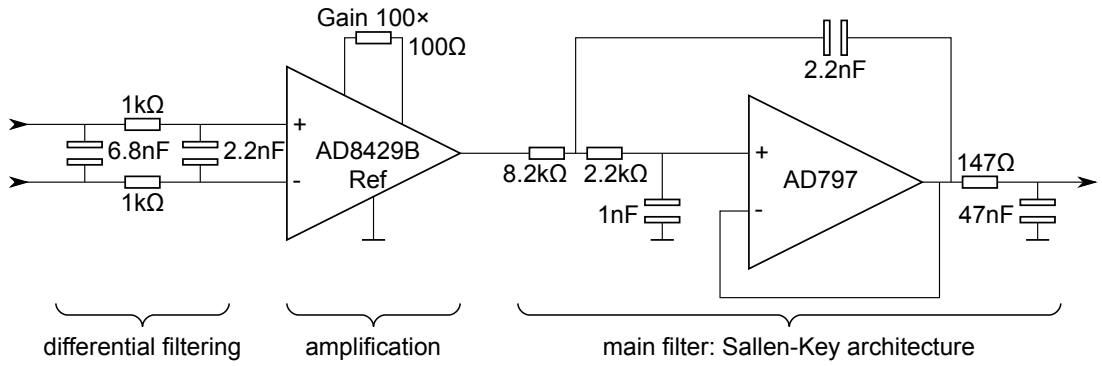



**Figure 5.4.:** Simplified schematics of the differential amplifier insert of my IceCube

the mating section on the turning axle. Hence the motor can be moved back into this backlash region where no torque is applied. For most of the measurements, however, this is impractical since the motor is constantly moving the sample. An additional approach to solve this problem is to mount the motor not to the top of the cryogenic insert, but on the sample chamber itself via a long tube. Then the motor will not push against the top of the cryogenic insert but against the sample chamber resulting in no net torque on the chamber. Although this is theoretically a perfect solution, the real implementation needs the tube to be vacuum tight in the top flange. This seal will also take some of the torque and thus hinders the perfect balancing of the torque at the chamber.

## 5.3. Electronics

The setup electronics is mainly built in the IceCube concept developed by Thomas Lorenz [Lor18]. It allows to have interchangeable electronic inserts in a shielded box, which is mounted directly on the cryogenic insert to keep the sensitive measurement wires short and inside of the shielded setup.
The measurement wiring consists of four pairs of shielded twisted wires which are individually sleeved inside the pair to reduce the parasitic capacitance to the other wire of the pair. These four pairs are each fed into differential amplifiers based of the ADA8421B with an additional Sallen-Key filter to reduce the bandwidth and noise. The amplifier has a gain of 100 (40 dB) and a cutoff frequency of about 20 kHz. The simplified schematic is shown in figure 5.4 and the full schematic and layout is reproduced in appendix C.2. This insert is called “sense”.
The IceCube has a common power supply insert that provides the other inserts with symmetric 15 V rails for the analog electronics and a single 5 V rail for the non-sensitive parts over additional connections in the backplane of the cube which are not part of the measuring wires. The insert is powered by the symmetric 24 V supply of the NIM-Crate which serves as power supply, grounding point and central access point to all setup electronics.

### 5.3.1. Resistance Map

As described in section 4.1 the resistance change of each side of the break junction can be used to support the heat flow simulations.
To adapt the eight measuring lines to the four wires of the sample, two small printed circuit boards (PCB) are used inside of the sample chamber: One PCB feeds the four pairs into a 11-pin 1.27mm pin pitch header, where each pair is separated from the others by a ground pin to avoid crosstalk. The second PCB plugs in the first and converts the four sample connections A to D (see appendix B for the labeling) to the four wiring pairs (1 to 4) in the following way: Pair 1 is connected to A and B, pair 2 is connected to B and C, pair 3 to A and D and pair 4 to D and C.
The two-PCB-solution in the chamber allows to easily change the connections between the eight measuring lines and the sample connector by switching the adapter PCB. Resoldering of the measurement wires is not necessary, which reduces possible wiring problems.
To measure these resistances, two equal voltage sources supply current over a resistor to each side. The resistor of 10 kΩ is connected to a differential amplifier to measure the current. The two equivalent voltage sources with their current read-out are combined into one insert for the IceCube, called "ΔR". The full schematic of the insert is shown in appendix C.3.
This insert sources the currents over the wires 2+ and 2− with ground on 3+ and 3−. The pairs 1 and 4 are used to sense the voltage at the sample connector between point A and B and between C and D with the "sense"-insert. A simplified schematic of the measurement is shown in figure 5.5.
The two simultaneous four wire measurements of the resistances are not ideal, because the measured value contains not only the resistance of the sample, but also the resistance of the wires from the plug to the sample. Since these wires have little resistance compared to the gold structures of the sample and the wires are not heated by the laser, the influence is small enough to be ignored.

### 5.3.2. Thermovoltage Measurements on Four-Wire MCBJs

Using the same cold PCB and a similar source insert with different wiring called "ΔV", it is possible to source a symmetric bias across the junction by inverting one of the voltage sources in the "ΔV"-insert on the wire pair 3+ and 3−.
When sourcing different currents across the junction, the "sense"-insert is used to measure the voltage across the junction on the paired wires 2+ and 2−. This four-wire measurement eliminates all effects of the cabling and is used to measure IV-curves. The thermovoltage can be extracted from these IV-curves as the voltage at which the current across the junction is zero, the conductance is the slope of the curve.
A simplified schematic for the measurement is shown in figure 5.6.

### 5.3.3. Thermovoltage Measurements with IV-Converter

To measure the current and voltage across the sample in the static configuration different electronics are used. Similar to the pulsed measurements, the voltage is sourced via a 10 kΩ resistor to limit the current, however, the current is not measured across this resistor but by using an IV-converter. This reduces the effect of the parasitic capacitance of the wiring

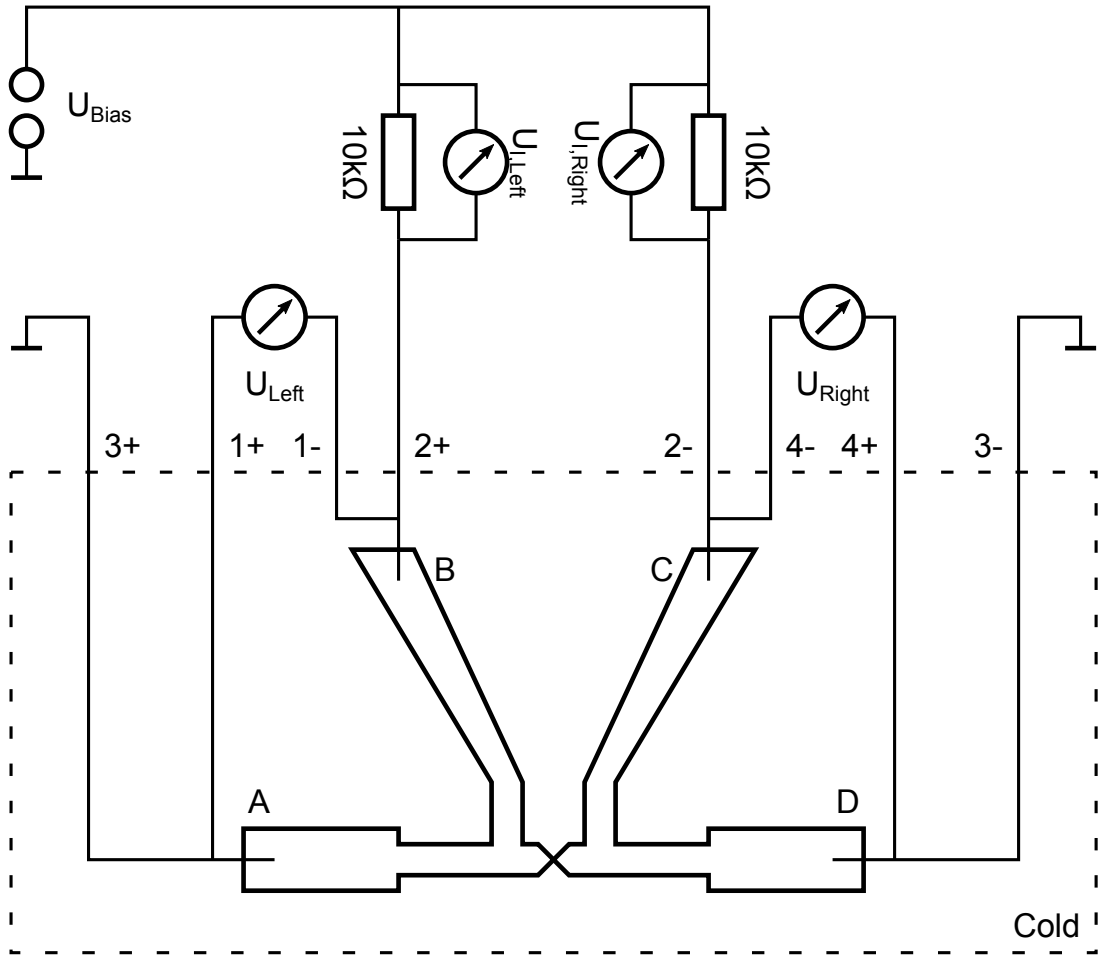



**Figure 5.5.:** Simplified schematic for the measurement of the resistance change of the leads: Similar currents are sourced over the two resistors, and the voltage drop due to the lead resistances of the sample is measured on the left and right side. For details see section 5.3.1.

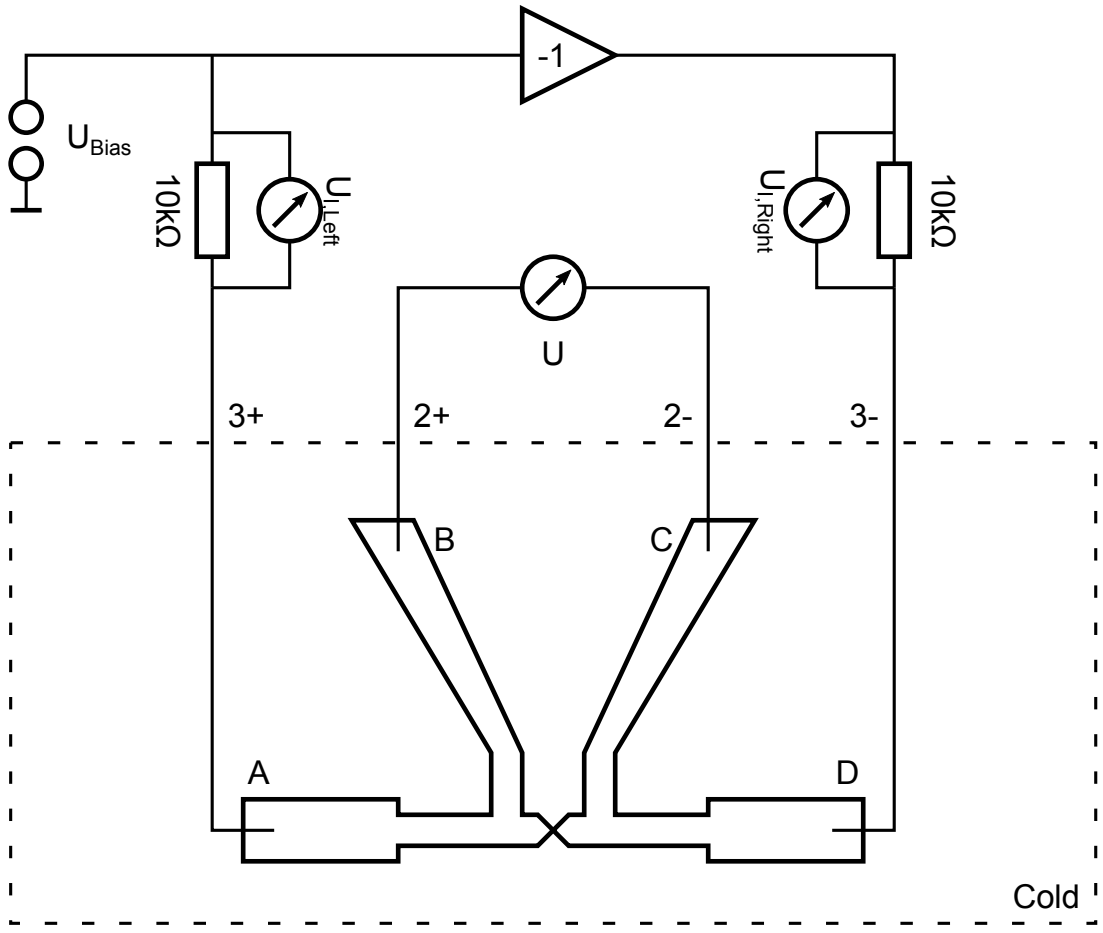



**Figure 5.6.:** Simplified schematic for the simultaneous measurement of the thermovoltage and the conductance of the MCBJ: Symmetric currents are sourced across the sample, and the voltage drop over the constriction is measured in a four wire setup. For details see section 5.3.2.

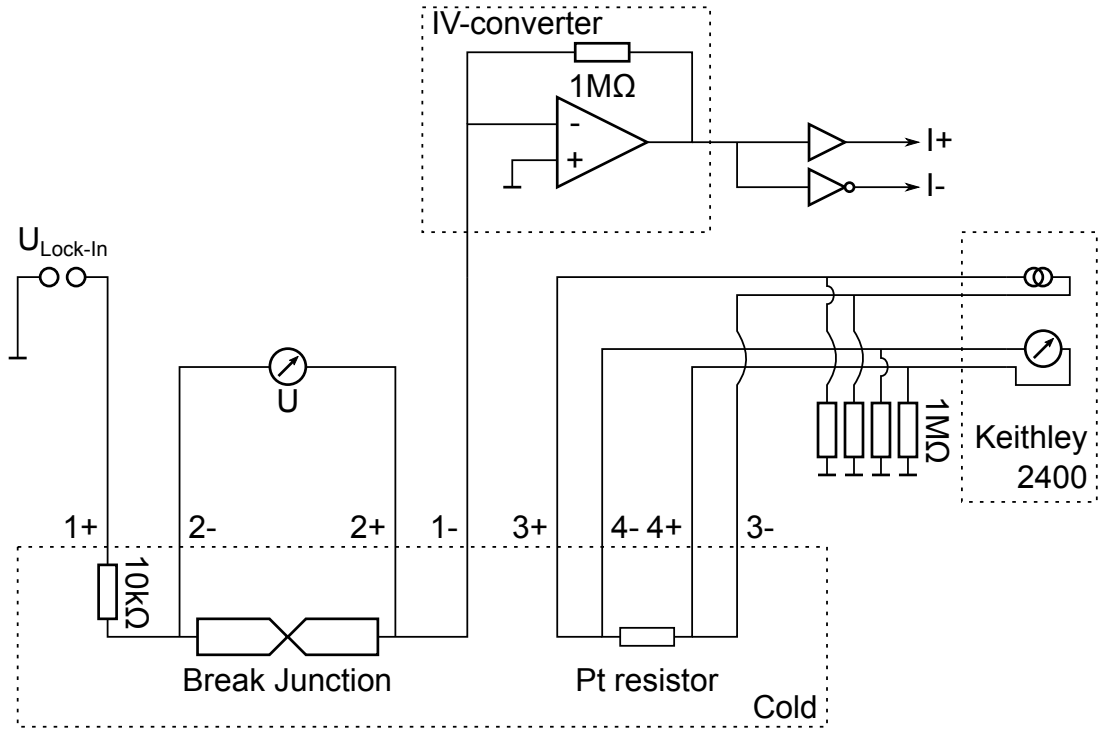



**Figure 5.7.:** Simplified schematic of the measurement electronics during the static measurements with the IV-converter and the resistance measurement to determine the sample temperature: The lock-in voltage is applied to the sample through the series resistance. The current is measured by the IV-converter and the voltage drop across the constriction is measured in a four wire setup. The resistance of the platinum resistance thermometer is also measured in the four wire configuration with the Keithley Source Meter.

allowing me to use a modulation frequency of 2 kHz for the lock-in amplifier. Additionally, the resistor is mounted inside the sample chamber to benefit from the possible noise reduction when the setup is cooled.
The resistance on the sample which is used to determine the temperature is measured in a four wire configuration which is separated from the wiring of the junction. The bias voltage across the junction is sourced over pair 1 and the voltage drop across the junction is measured across pair 2 by a differential amplifier using the "sense" insert. The IV-converter and the source are integrated into one insert, using the same plugs and compatible pinouts as the source inserts "$\Delta$R" and "$\Delta$V". The wiring for the temperature sensing resistance is also handled by this insert: The signals are filtered and pulled to ground to prevent the destruction of the sample by having a floating potential and are accessible via a plug that is compatible with the existing hardware for the "breakout" insert which will be described in section 5.3.5.
A simplified diagram of the complete wiring is depicted in figure 5.7. The complete description of the cold PCB and the insert can be found in appendix C.2.

### 5.3.4. Thermovoltage Measurements with Cold Front-End IV-Converter

To reduce the influence of the parasitic capacitances of the cables even further, I built an IV-converter with a matched JFET pair on a common silicon chip as a discrete front-end. The front-end and the feedback resistor are placed in the cooled sample chamber, while the rest is integrated into an insert for the IceCube.
This separation of the converter is designed in such a way that the long wires are only driven by low impedance sources. This prevents the parasitic capacitances of the wires from slowing down the electronics, and it also reduces the noise. The noise is further reduced since the feedback resistor is also the cooled sample chamber.
The schematic for this can be found in appendix C.6.
The drawback of this separation is that the four pairs of existing wiring are now used to supply the front-end and thus only normal two wire junctions can be measured. The additional contacts for the measurement of the temperature-dependent resistance cannot be used.

### 5.3.5. Auxiliary Electronics

For the use of the setup it is necessary to be able to change the electronic inserts and to plug or unplug them while a sample is connected. To ensure that no voltage spikes or unwanted currents flow through the sample, an additional insert is used to clamp all the measuring wires to ground. Since the plugging of this insert can itself generate an unwanted current flow, the clamping is equipped with switches to discharge the electronics over a $1\,\mathrm{M}\Omega$ resistor before shorting the wire to ground.
Another insert is the "breakout" insert which gives direct, albeit filtered, access to the eight wires in the cryostat. The wires are directly accessible via SMC connectors in the NIM-crate. It allows to quickly debug problems of the wiring or the sample.
This flexibility is enhanced by plugging these eight wires into a self-built NIM-crate module, that can be configured by software to connect any of the twelve inputs to the center conductor or the shield of eight output BNC-connectors. Three of the outputs are connected to a HP34401A multimeter and the source and sense terminals of a Keithley 2400 source-measure-unit, allowing automated testing of the complete wiring.
For increased comfort two white LEDs are mounted inside the sample chamber. They are used as light source for visual debugging and during the primary alignment of the optics.

## 5.4. Optics

The main purpose of the optics is to achieve a stable laser focus in the desired position on the sample for controlled heating. This focus is accomplished by an aspherical lens ($f = 8\,\mathrm{mm}$) inside the sample chamber. To change the focus of the laser, the lens is mounted on a screw with a fine pitch thread. So the lens can be moved towards the sample in a very controlled manner. The lens mount on the screw is realized by a spring pressing the lens against a Teflon spacer. The spacer's thickness is designed to compensate the thermal shrinkage of the rest of the holder. Thus the distance between lens an sample remains constant during cool down.

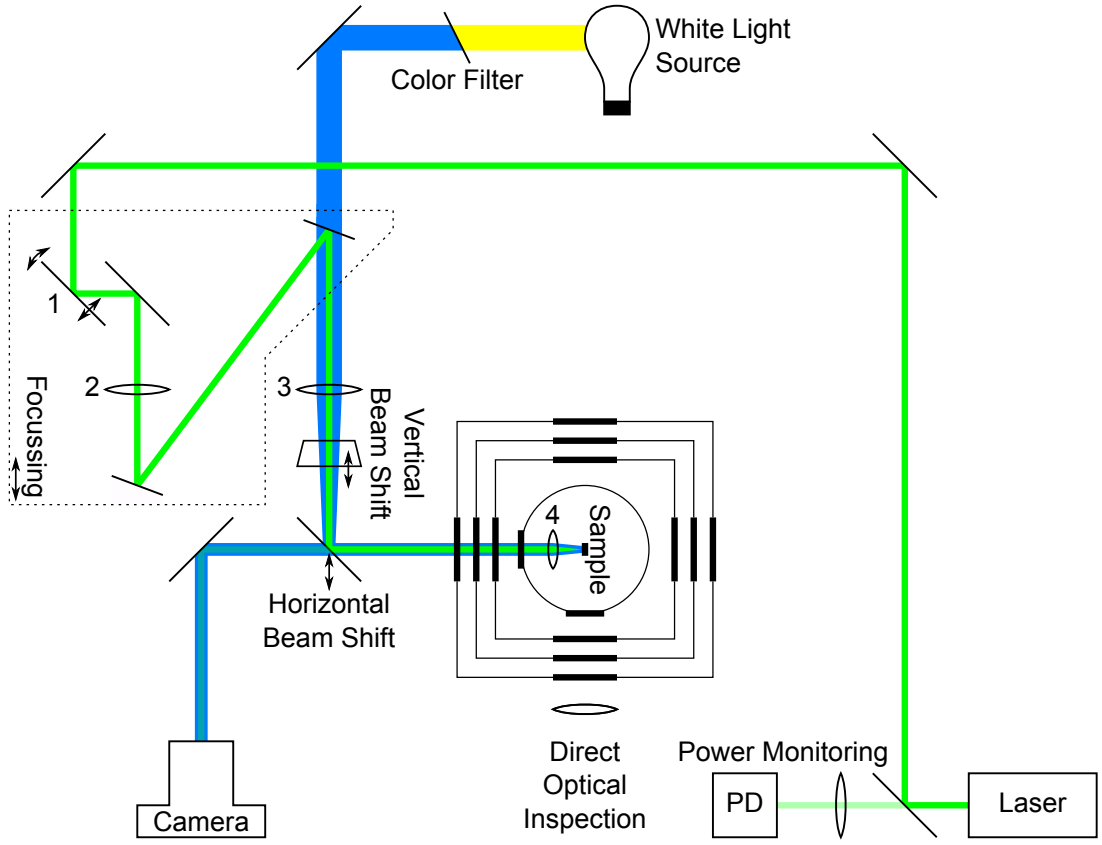



**Figure 5.8.:** The laser is guided through a folded 4-$f$-setup with an adjustable distance between the lenses. This allows to move the laser focus on the sample and to change the focus slightly without access to the interior of the cryostat. The 4-$f$-setup consists of the mirror 1 to move the beam, the lens 2 at a distance of $f$, the lens 3 at a distance of $2f$, and the target lens 4 at a distance of $f$. The white light is used to illuminate the sample when positioning the laser focus.

To move the focus across the sample, I use a 4-$f$-setup consisting of two lenses and one motorized mirror. When the distances between the lenses, the mirror, and the sample chamber are correct, the rotation of the mirror will change the incident angle on the sample chamber while maintaining the laser centered on the lens inside. The rotation of the beam while it stays in the same position enables me to move the focus across the sample without having to readjust any other part of the optics.
This setup is folded with two mirrors into a more compact geometry to keep some of the space of the optical table clear to stand on when working on the cryostat. One of the folding mirrors is a dielectric mirror designed for $\lambda = 532\,\mathrm{nm}$ with a polished back surface. Therefore the laser is not completely reflected, but it can also be used to insert blue light into the sample chamber. This light is used to illuminate the sample to see the structure as a reference when moving the laser to the correct position. Since the blue light passes through only one of the 4-$f$-lenses it is not focused but rather broad on the sample. Otherwise the illuminated spot would be rather small.
Additionally some parts of the optics can be moved as a group to change the distance between the two lenses. Even though this is not ideal for the 4-$f$-setup, it changes the divergence or convergence of the beam entering the sample chamber. This allows to slightly modify the focus on the sample even when the sample chamber is closed.
All the light is coupled into the sample chamber by another dielectric mirror with a polished backside. The mirror brings most of the laser power into the sample chamber while allowing enough transmission to take pictures of the sample and the laser position with a conventional camera. The mirror is mounted on a translation stage so that the laser can be shifted horizontally to center it on the lens. There is also a thick glass plate in front of this mirror, that can be pivoted on a horizontal axis. The rotation moves the light up and down allowing to center the laser vertically on the lens inside of the sample chamber. The camera objective is not mounted directly in the path, but gets the light by an additional beam splitting plate, so that some of the alignment can be checked directly by eye along a straight path.
The laser-head is mounted close to the controller. To reduce the risk of accidentally moving the laser-head by pushing on the rather ridged cables, they are clamped to the table. Since the laser-head is on the opposite side of the cryostat, three mirrors guide the light around the cryostat and feed it into the 4-$f$-optics in the direction of the focus adjustment motion, to keep the movement of the beam minimal when adjusting the focus.
The first mirror is dielectric and backside polished so that a very small portion of the light is transmitted. This light gets focused on a photo-diode to monitor the laser power and the pulse shape.
As an additional comfort, a lens is mounted in front of the side window to magnify the side view on the interior of the sample chamber. This allows one to directly observe the bending of the sample.

## 5.5. Sample Fabrication

All samples are fabricated by electron beam lithography. But the details are different depending on the type of sample used.

### 5.5.1. Samples for pulsed measurements

This kind of samples use a 500 µm thick Kapton$^{TM}$ substrate (Cirlex$^{TM}$ by Fralock). This substrate material is cut into wafer form and then processed in the following way:

- Spin-coating of Durimid 115A$^{TM}$ by Fujifilm at 4000 rpm to get a roughly 3 µm sacrificial layer for the etching process at the end and a smooth surface
- Prebaking at 130 °C for 5 min in a convection oven
- Baking in vacuum overnight
- Spin-coating of MMA MAA EL11 copolymer at 2500 rpm
- Prebaking at 100 °C for 5 min on a hot plate
- Spin-coating of 950PMMA A4 resist at 5000 rpm
- Baking at 170 °C for 30 min in a convection oven
- Cutting the wafer to sample size
- Spin-coating AR-PC 5090.02 (Electra 92)$^{TM}$ conductive resist at 4000 rpm
- Baking the resist at 120 °C for 2 min on a hot plate
- Electron beam lithography with the structure shown in figure 5.9 and a dose of 80 µC cm$^{-2}$ at 10 kV
- Removing the conductive resist with deionized water
- Developing the PMMA resist in 1:3 mixture of methyl isobutyl ketone and isopropanol for 30 s
- Developing the copolymer in isopropanol for 180 s to get an undercut
- E-beam evaporation of 80 nm of gold at a rate of 2 Å s$^{-1}$
- Lift off in warm acetone for 5 min
- Plasma etching in 50:1 mixture of $O_2$ and $SF_6$ at 1 mbar and 45 W plasma power for 22 min
- Contacting with silver conductive paint
- Gluing of the wires with SofortFest$^{TM}$ by UHU for strain relief

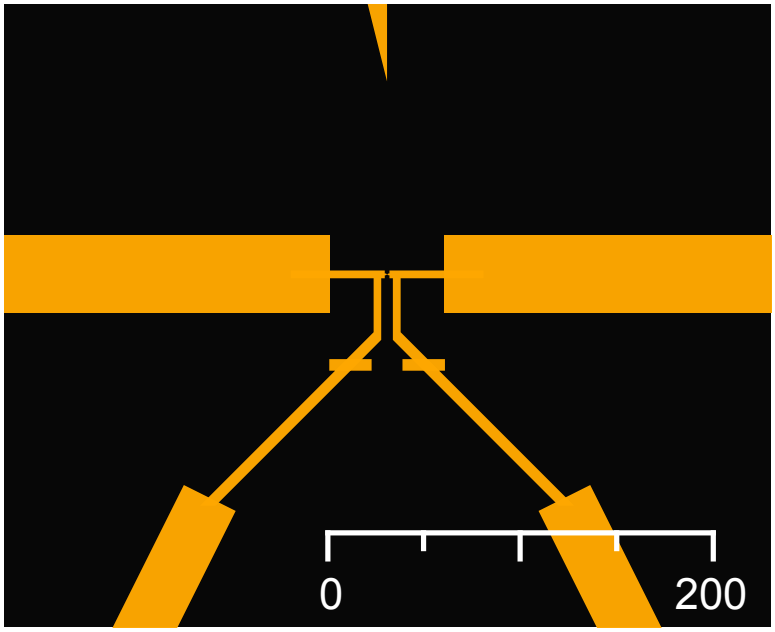



**Figure 5.9.:** The structure of the gold samples used for the pulsed measurements contains the constriction and four leads for contacting and sensing of the resistance change.

### 5.5.2. Samples for static measurements

For the static measurements the additional platinum resistor for the thermometry makes the sample fabrication more complicated. Thus the following fabrication procedure is structured following the fabrication steps.

- Wafer preparation
  - Polishing bronze wafer
  - Spin-coating of Durimid 115A$^{TM}$ by Fujifilm at 4000 rpm to get a roughly 3 µm sacrificial layer for the etching process at the end and a smooth surface
  - Prebaking at 130 °C for 5 min in a convection oven
  - Baking in vacuum overnight
- first step: gold break junction
  - Spin-coating of MMA MAA EL6 copolymer at 1700 rpm
  - Prebaking at 100 °C for 5 min on a hot plate
  - Spin-coating of 950PMMA A2 resist at 5000 rpm
  - Baking at 170 °C for 30 min in a convection oven
  - Cutting the wafer to sample size
  - Electron beam lithography with the structure shown in figure 5.10 and a dose of 140 µC cm$^{-2}$ at 10 kV

  - Developing the PMMA resist in 1:3 mixture of methyl isobutyl ketone and isopropanol for 30 s
  - Developing the copolymer in isopropanol for 180 s to get an undercut
  - E-beam evaporation of 80 nm of gold at a rate of 2 Å $s^{-1}$
  - Lift off in warm acetone for 30 min
- second step: $Al_2O_3$ isolation layer
  - Spin-coating of MMA MAA EL6 copolymer at 1700 rpm
  - Prebaking at 100 °C for 5 min on a hot plate
  - Spin-coating of 950PMMA A2 resist at 5000 rpm
  - Baking at 170 °C for 30 min in a convection oven
  - Electron beam lithography with the structure shown in figure 5.10 and a dose of 140 µC $cm^{-2}$ at 10 kV
  - Developing the PMMA resist in 1:3 mixture of methyl isobutyl ketone and isopropanol for 30 s
  - Developing the copolymer in isopropanol for 180 s to get an undercut
  - Sputtering of 20 nm of $Al_2O_3$ at a 200 W for 3333 s at an argon pressure of 2 mTorr
  - Lift off in warm acetone for 30 min
- third step: Pt resistance thermometer
  - Spin-coating of MMA MAA EL6 copolymer at 5000 rpm
  - Prebaking at 100 °C for 5 min on a hot plate
  - Spin-coating of 950PMMA A2 resist at 5000 rpm
  - Baking at 170 °C for 30 min in a convection oven
  - Electron beam lithography with the structure shown in figure 5.10 and a dose of 85 µC $cm^{-2}$ at 10 kV
  - Developing the PMMA resist in 1:3 mixture of methyl isobutyl ketone and isopropanol for 30 s
  - Developing the copolymer in isopropanol for 60 s to get an undercut
  - E-beam evaporation of 20 nm of Pt at a rate of 2 Å $s^{-1}$
  - Lift off in warm acetone for 30 min
- forth step: crosslinked PMMA edge softeners
  - Spin-coating of 950PMMA A2 resist at 5000 rpm
  - Baking at 170 °C for 30 min in a convection oven
  - Electron beam lithography with the structure shown in figure 5.10 and a dose of 7000 µC $cm^{-2}$ at 10 kV
  - "Developing" the PMMA resist in warm acetone for 30 min

- fifth step: gold contact for the resistor
  - Spin-coating of MMA MAA EL11 copolymer at 5000 rpm
  - Prebaking at 100 °C for 5 min on a hot plate
  - Spin-coating of 950PMMA A2 resist at 5000 rpm
  - Baking at 170 °C for 30 min in a convection oven
  - Electron beam lithography with the structure shown in figure 5.10 and a dose of 180 µC cm$^{-2}$ at 10 kV
  - Developing the PMMA resist in 1:3 mixture of methyl isobutyl ketone and isopropanol for 30 s
  - Developing the copolymer in isopropanol for 180 s to get an undercut
  - E-beam evaporation of 150 nm of gold at a rate of 2 Å s$^{-1}$
  - Lift off in warm acetone for 30 min
- finishing procedures:
  - Plasma etching in 50:1 mixture of $O_2$ and $SF_6$ at 1 mbar and 45 W plasma power until a depth of 500 nm is achieved
  - Contacting with silver conductive paint
  - Gluing of the wires with EndFest$^{TM}$ by UHU for strain relief

### 5.5.3. Samples for molecule measurements

The yield of break junctions with molecules is low, and the electronics occupy all measurement lines. Therefore, there is no thermometry structure on the sample which makes the sample fabrication much more efficient. These samples also use a bronze substrate, which, too, is prepared by polishing. Afterwards the following steps are performed.

- Spin-coating of Durimid 115A$^{TM}$ by Fujifilm at 4000 rpm to get a roughly 3 µm sacrificial layer for the etching process at the end and a smooth surface
- Prebaking at 130 °C for 5 min in a convection oven
- Baking in vacuum overnight
- Spin-coating of MMA MAA EL6 copolymer at 1700 rpm
- Prebaking at 100 °C for 5 min on a hot plate
- Spin-coating of 950PMMA A2 resist at 5000 rpm
- Baking at 170 °C for 30 min in a convection oven
- Cutting the wafer to sample size
- Electron beam lithography with the structure shown in figure 5.11 and a dose of 140 µC cm$^{-2}$ at 10 kV

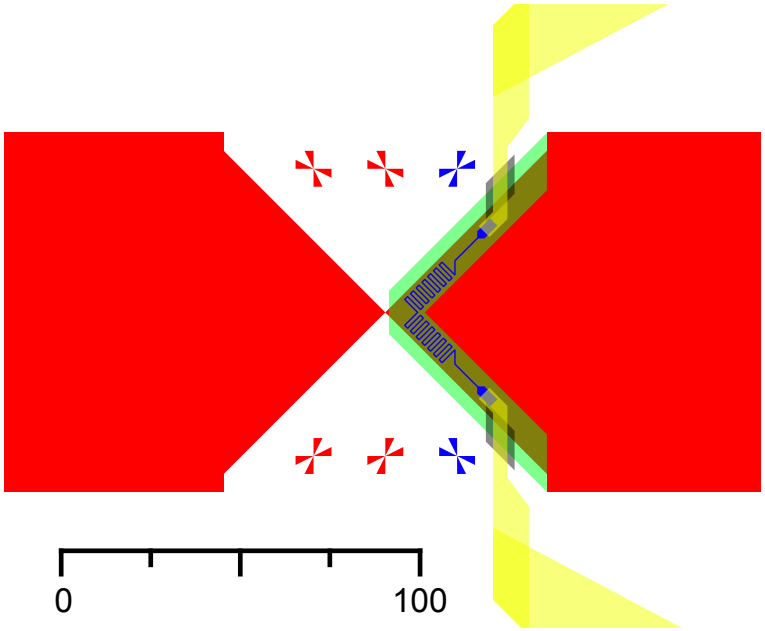



**Figure 5.10.:** The structure of the samples used for the static measurements contains the constriction and an electrically isolated Pt resistor for the thermometry. The different lithography steps are color coded: The gold break junction written in the first step is shown in red, the second step is the electrical isolation between the break junction and the Pt resistor. The isolation layer made of $Al_2O_3$ is shown in semi transparent green and the Pt resistor is colored blue. To round over the edges of the underlying gold structure and to prevent electrical contact at this sharp edge, a layer of cross-linked PMMA, shown in semi transparent black, is used. As last step, the Pt resistor is connected by a gold structure shown in semi transparent yellow. The scale bar shows the size in µm.

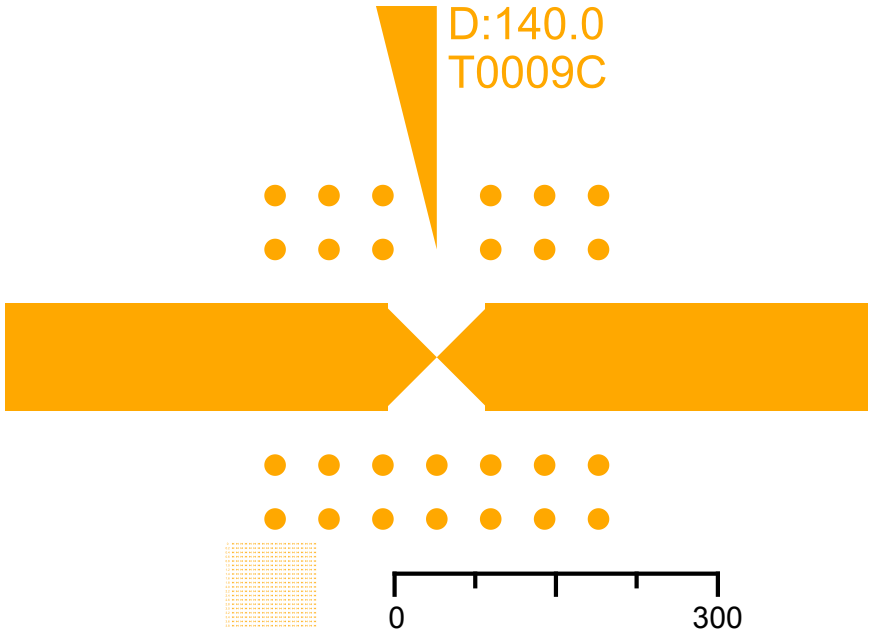



**Figure 5.11.:** The structure of the gold samples used for the molecule measurements contains the constriction and additional reference marks to facilitate the correct positioning of the laser focus. The scale bar shows the size in µm.

- Developing the PMMA resist in 1:3 mixture of methyl isobutyl ketone and isopropanol for 30 s
- Developing the copolymer in isopropanol for 180 s to get an undercut
- E-beam evaporation of 80 nm of gold at a rate of 2 Å $s^{-1}$
- Lift off in warm acetone for 30 min
- Plasma etching in $O_2$ at 1 mbar and 45 W plasma power until a depth of 500 nm is achieved
- Drop casting molecules on the sample
- Evaporation solvent in high vacuum
- Contacting with silver conductive paint
- Gluing of the wires with SofortFest™ by UHU for strain relief

## 5.6. Setup Surveillance

Since the setup is not just a simple bath cryostat, several parameters are recorded and can be checked even remotely to verify the proper working conditions of the cryogenic system.

- The level of the liquid nitrogen bath is detected by a home built capacitive sensor. The sensor consists of two tubes arranged as a cylindrical capacitor with the outer tube grounded to prevent the radiation of electrical fields. The liquid between the metal tubes will change the capacitance which in turn changes the resonant frequency of an LC circuit. This frequency is measured and the nitrogen level is calculated in hardware.

- If the setup is used with liquid helium, the flow through the needle valve depends on the helium level in the bath. This bath level is monitored with a commercial helium level sensor and the Model 135 Liquid Helium Monitor$^{TM}$ from AMI. Additionally the total helium evaporation in the system is monitored by a sensor on the gas volume counter in the recovery system. The evaporation rate can be used to deduce changes or complete blocking of the needle valve and will also show larger problems with the cryogenic system.

- The supply voltages of the NIM crate and the external grid power are also monitored, to allow an easy diagnosis of the results of power cuts and overloads of the system power supply.

- The isolation vacuum is pumped with a turbo molecular pump. The rotation speed and consumed electrical power are monitored, as well as the pressure in the isolation vacuum itself. This is used to ensure that the isolating properties required for the cryogenics are assured.

- The optics and the sample in the system are monitored by a camera. This allows to take pictures of the sample and the laser focus at the beginning and end of each measurement part. The camera can also provide a live picture of the sample and the laser, which is used for the alignment, positioning, and troubleshooting of the optical system.

## 5.7. Software

Inspired by the software concept of Thomas Lorenz [Lor18], for each hardware device a separate program has been written. This structure has several benefits: Using the program as an interface that handles the hardware communication, the tedious details can be masked from the higher level programs. The state of each hardware device can be checked and logged irrespective of the current functions of the setup. The hardware can be connected to different computers allowing to use the benefits of different operating systems. The interface software can communicate with several high level programs at the same time which allows e.g. to read out the setup temperature by the software controlling the measurement while also showing the temperature in the setup monitoring software. Additionally, if one of the programs crashes, the rest of the setup can keep operating and the program can be restarted without unnecessary side effects.
Of course, this structured approach also has some drawbacks: Due to the use of the network to communicate, with the hardware via an interface program, additional computation power is needed. The extra step in the communication also generates lag times and increases the delay in the commands. In my setup the network interface is not used to

handle large amounts of data or information with critical timing, hence this does not pose a problem. The masking of the low level hardware communication also prevents some performance optimizations. Therefore there are some higher level programs which do not follow the structure perfectly by accessing the hardware directly. Another drawback is the sheer number of different programs with make the maintenance somewhat cumbersome.
The decision for Python as the programming language for the software was made for several reasons. I chose Python because it is an interpreted language. This means you execute the source code directly and do not have a compiled executable file where you lose track of which version of the source code it belongs to. The source code of exactly the program running can be quickly checked and changed. The scripting nature means that you can directly try different commands in a terminal which is especially useful for the debugging of the communication with the hardware. There are many easy to use libraries for interfacing with the hardware, the network, and different file formats like `.mat` or `.h5` and it is possible to create graphical user interfaces. Python is platform-independent thus I can run the same code on my different computers with different processor architectures and operating systems. Furthermore it is open-source so that I do not need to worry about all the problems with product keys and licence managers.
A few measurements are very short and simple. These are controlled by short terminal based programs that perform the measurement when started and end when the measurement is completed. However, most of the measurements are rather long and it is therefore nice to have some status feedback and some control over the process e.g. to pause the measurement for the transfer of coolant.
Controlling and monitoring are done by the so called "MeasurementController". This is a software with a graphical user interface that shows the progress, the elapsed and remaining measurement time, the state of the measurement, and the current operation. All measurement programs provide network commands on the same port for the controller to get their information. As an additional level of surveillance, the controller will check the time since the last update of the current operation. When the measurement software runs normally, it is updated at least every few seconds. Hence, the controller can spot if the measurement software is stuck and notify me via e-mail and set the measurement state to `stuck`. The state of the measurement can also be influenced by buttons to start, pause, halt, continue, and kill the measurement.

### 5.7.1. Measurement states

During the measurement, the software can be in different phases, so-called states. All states and their transitions are shown in figure 5.12. Changes from one state to another may be performed by the measurement software itself, or they may be externally triggered by network commands from the "MeasurementController". The possible states are:

- `continuing`
  In this state the software will perform the necessary operations to resume the measurement from the `paused` or `halted` state.

- `finished`
  This state is reached when the measurement is completed.

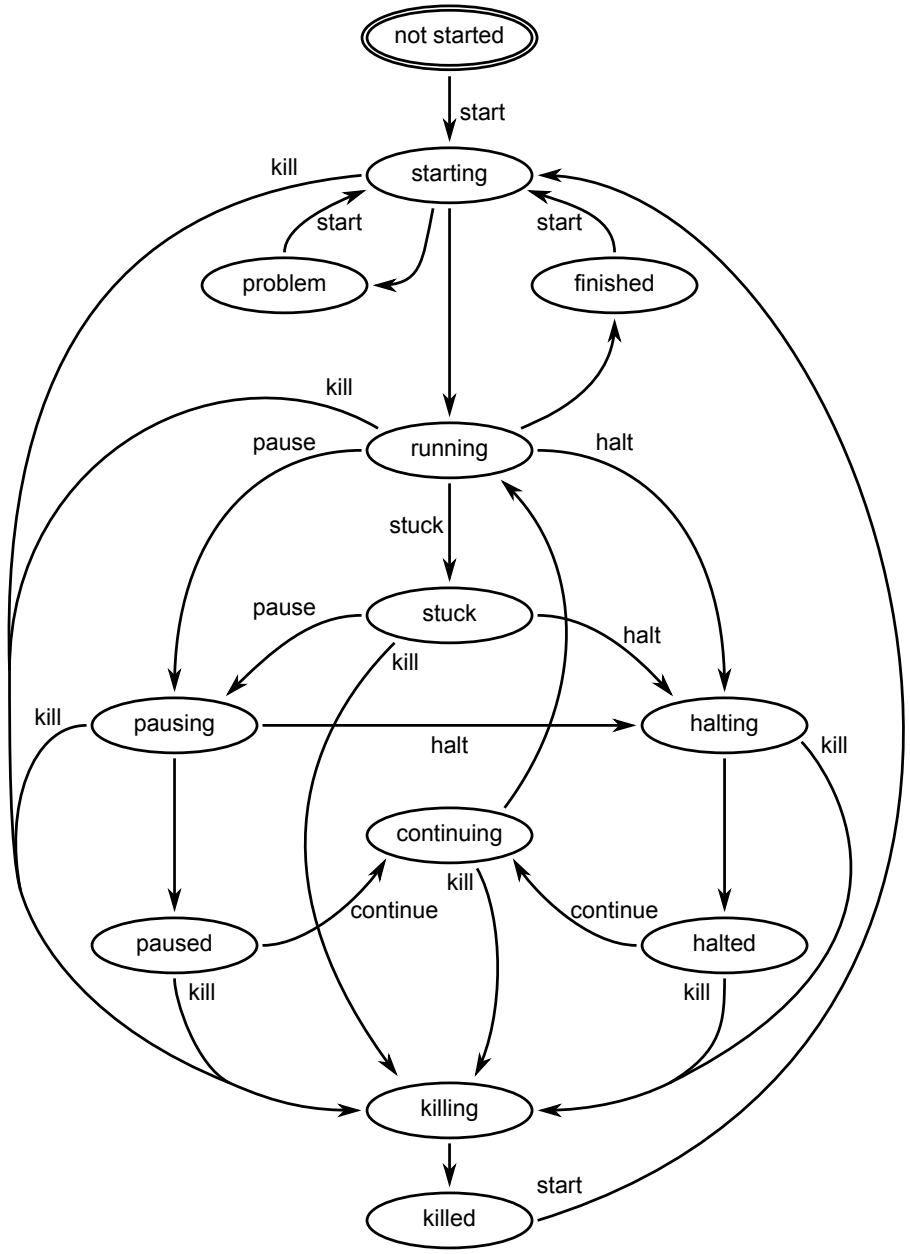



**Figure 5.12.:** This state diagram shows all possible measurement states. The labels on the arrows are the external commands that are used to change the state. The arrows without labels represent state changes without external input e.g. by completing tasks.

- `halted`
  The measurement is paused as quickly as possible without compromising or loosing any data. This is used to prevent the measurements from acquiring possibly compromised data during user interactions with the setup.

- `halting`
  This is a transition state in which the software makes all the necessary preparations for the halted state. These may be operations like stopping the motor,turning off the laser, stopping the data acquisition, and saving the data.

- `killed`
  The software is in this state when the commands for killing the measurement are executed.

- `killing`
  When the user sends the `kill` command the software changes to this transition state and executes the necessary actions. The killing of the measurement will stop the measurement as fast as reasonably possible. The acquired data is saved, however the measurement may end on incomplete dataset.

- `not started`
  This is the state of the measurement software after the program was launched. The measurement is not immediately started, since it may require some configuration.

- `paused`
  The measurement is paused in a save configuration. It is intended to be used for longer user interactions with the setup. To retain the integrity of the data, this pause will not interrupt the acquisition of datasets. This may lead to waiting times of several minutes between the `pause` command and the reaching of this state.

- `pausing`
  This is the transition state in which the software waits for the completion of the dataset and then sends the commands to prepare for the pause.

- `problem`
  This state is reached when the configuration verification at the beginning of the measurement is not successful e.g. the user is not happy with the laser position.

- `running`
  This is the normal state of the measurement, were data is collected.

- `starting`
  This transition state is reached at the beginning of the measurement by the `start` command. The hardware is initialized, the user verifies configuration and everything is prepared for the data acquisition.

- `stuck`
  The controller can set the measurement in this state if the timeout for the operation update was exceeded. This does not happen in the normal operation and indicates

some malfunction or communication problem. This has no effect on the software and is treated as if the state was `running`. It serves mainly as an indication for the user. If the program reaches the next operation update, the state is automatically changed back to `running`.

The network commands to change the state are

- `continue`
  This command changes the state to `continuing`. This is only allowed from the `paused` and `halted` state.
- `halt`
  This command changes the state to `halting`. This is only allowed from the `running`, `stuck`, and `pausing` states.
- `kill`
  This command changes the state to `killing`. This is allowed from all possible states except for `killing`, `killed`, `finished`, and `not started` since there is no point in killing the before it has started or after it has finished.
- `pause`
  This command changes the state to `pausing`. This is only allowed from the states `running` and `stuck`.
- `start`
  This command changes the state to `starting`. This is only allowed from the states `not started`, `problem`, `finished`, and `killed`.
- `stuck`
  This command changes the state to `stuck`. This is only allowed from the state `running`.

### 5.7.2. Important programs for the pulsed measurements

An overview of the programs used for the pulsed measurements is shown in figure 5.13. The program "Thermospannungsmessung_V3" measures the thermovoltage at different conductances and the program "Kartenaufzeichnen_V2" moves the laser over the sample and measures the resistances of the leads. The "Widerstandskalibrierung" program measures the same resistances at different temperatures and the "Probencharakterisierung" software measures the resistances of the four sample leads and the constrictions, thereby checking the complete wiring and the sample contacts. The programs are described in the following and the full technical details can be found in appendix E.

#### "Thermospannungsmessung_V3"

The main program for this measurement is the "Thermospannungsmessung_V3" program. It is controlled by the "MeasurementController" and performs the acquisition of the I-V-curves during the laser pulse sequence at different conductances. The software will open or close the constriction to sweep across the range from 0.2 to 20 $G_0$. To measure the

**Figure 5.13.:** This diagram shows the software used for a pulsed measurement. The computer is indicated by the IP-address and if the software supports network commands, its port is also shown. The arrows indicate the control flow.

conductance, a voltage of 20 mV is applied to the sample through the series resistors. The voltage drop across the sample is amplified and measured by the multimeter. The software uses this value to calculate the conductance.
Three different motor speeds are used when the measured conductance is bigger than 40, 80, and 160 $G_0$, or smaller than 0.1, 0.05, and 0.025 $G_0$. This reduces the necessary movement time if the conductance of the constriction is far from the measurement range. If the conductance is out of the measurement range or the motor is stopped by the limits in the software or hardware, the program will change the direction of motion.
As long as the sample conductance is within the measurement range, the bending stops if the conductance has changed by more than 2 % and a data point is measured. This begins by turning the applied voltage off, turning the laser slightly on, and taking a picture with the camera to document the position of the laser spot on the sample. Since the reflected white light needs to be sufficiently bright to see the structure of the sample, the laser power has to be very low to have the central parts of the focus correctly exposed. A picture with just the overexposed green laser focus and no visible references of the sample structure would be useless. After this picture, the laser is set to the pulsing pattern with the correct power. The program will wait for 5 s for the changes to be executed and to have a stable temperature distribution in the sample geometry. Then the source voltage is set to a 2 kHz sinusoidal alternating voltage with a peak-to-peak value of 27 mV. and the transient recorder is started to acquire the data for 45 s. After the data is recorded it is saved, the laser is turned off, the compression of the data is initiated, and the voltage to measure the conductance during the motor movement is applied. Then the motor starts moving to change the bending of the sample and everything repeats.
The program uses the "Multimeter" software to measure the voltage drop during the movement, the "Faulhaber" software to access the bending motor, the "DataCompression_IV" software to compress and analyse the data in real time after the data point has been recorded, and the "Laser" software to change the laser between the off state, the constant power necessary for the camera picture, and the pulsed mode for creating the temperature differences. The shutter of the camera for documenting the laser position is controlled by the "Kamera" software, and the "TransCOM" program is the interface to the API of the transient recorder for starting and stopping the data acquisition and for saving the data on the hard drive.

### "Kartenaufzeichnen_V2"

This software moves the laser spot across the sample while recording the time dependant resistance change of the leads. It requires the "ΔR" IceCube insert.
The program positions the laser on a grid on the sample. This grid is defined by a primary axis given by two points and two diagonal corners of the rectangular area. The number of steps can be set separately for the two orthogonal directions. During the measurement, the program moves the laser to every point on this grid. At each point the following actions are performed: The laser is turned on, but only faint, and the camera takes a picture to document the position. The laser is now set to pulsing with the power for the measurement and the program applies the voltages -200, -100, 0, 100, and 200 mV. For each voltage the transient recorder records five seconds of data. The measurement can

be halted at this point before the next voltage is applied. When data are recorded for all five voltages, they are compressed into one `mat`-file for the point. Now, the program can be paused before the laser is moved to the next point.
The "MeasurementController" software controls the state of the measurement performed by the "Kartenaufzeichnen_V2" software. This software uses the "Source" program to apply the different voltages, the "ThorlabsMotor" program for moving the laser, the "Laser" program to control the power and mode of the laser, the "Camera" software to take the pictures, and the "DataCompression" software for compressing and combining all the data for each point.

#### "Widerstandskalibrierung"

This program uses the "ΔR" IceCube insert to source different voltages of -100, -50, 0, 50, and 100 mV. The transient recorder measures the currents and voltages for five seconds at each voltage. This is repeated for different temperatures by changing the temperature controller setpoints after each measurement to the next temperature between 86 and 120 K in steps of 2 K. After the temperature change, the program waits until the temperature settled, which is determined checking if the temperature does not differ more than 0.05 K from the setpoint at any time during one minute.
The software uses the "TransCOM" program to control the transient recorder, to apply the voltages the "Source" program is used, and the "TemperatureLogger" program is needed to change and check the temperatures in the setup.

#### "Probencharakterisierung"

This terminal based program uses the multiplexer with the "breakout" IceCube insert to perform four wire resistance measurements through the eight wires connected to the sample. The Keithley 2400 Source Measure Unit sources 1 µA with a voltage limit of 100 mV trough different wires, the Agilent 34401A Multimeter measures the voltage drop. With the known current, the voltage is easily converted to a resistance by the software. Using the multiplexer to switch the source and the multimeter to different wires, the software measures the resistances of all four connections from the sample plug to the central structure of the sample, as well as the resistance of the constriction itself.
This characterisation of the sample is useful for checking the resistances at different temperatures, but it also involves the use of the complete wiring and all electrical contacts to the sample. Thus a correct reading of the five resistances also shows that all electrical connections to the sample are fine.
The multiplexer is controlled by the "Multiplexer" software, and the source and multimeter are directly accessed over the GPIB bus.

### 5.7.3. Important programs for the static measurements

Similar to the pulsed measurements, an overview of the software used for the static measurements is shown in figure 5.14. The main measurement is performed by "Thermospannungsmessung_statisch" or "Thermospannungsmessung_statisch_V2" in the more recent measurements. Both programs are very similar, except that the newer software does

not use the transient recorder to save the data and uses the digital values of the lock-in amplifier directly.

### "Laserkarte"

This program measures the resistance of the platinum thermometer at every position of a grid for different laser control voltages. The grid contains 80 by 80 points and at each point, a picture is taken, then the laser control voltage is set to all values between 0 and 6 V in steps of 0.5 V. After setting the voltage, the system is given one second to equilibrate and then the resistance is measured.

### "Thermospannungsmessung_statisch"

This is the main measurement program. It continually monitors the conductance, the motor position, and the motor speed. The motor speed is calculated from the measured conductance $G$ by the following formula:

$$1\,\mathrm{G_0}^2 \cdot G^{-2} + 0.1\,\mathrm{G_0}^{-2} \cdot G^2 \tag{5.1}$$

The speed is limited by a minimum speed which is set to 40 and the maximum speed that the motor is capable of which is 6000. This speed adjustment is useful because it prevents the setup from spending lots of time at motor positions where the junction is either completely open or closed.
During the measurement, traces are recorded under different conditions. The laser starts on the side with the thermometer. The laser is turned off and a trace is recorded with the motor opening the junction. Then the motor switches direction and another trace is recorded. Now the laser is turned on and again an opening and a closing trace are recorded. The laser is then moved to the side without the thermometer and another two traces are recorded without the laser heating and also with the laser heating. Then the laser moves back to the side with the thermometer and the process is repeated until it is stopped by the user.
The recording of every trace consists of the following steps: The laser is turned on at a low power and the camera takes a picture of the position of the laser focus. Then the laser is either turned off, or fully on to heat the sample. In both cases the software waits for 5 s for the temperature distribution in the sample to reach a stable equilibrium. The resistance of the platinum thermometer is measured, and the resistance measurement is switched to the continuous mode. The transient recorder starts the data acquisition and the bending starts in the appropriate direction. When the conductance has reached the limit which is either $20\,\mathrm{G_0}$ for closing traces or $0.2\,\mathrm{G_0}$ for opening traces, the motor is stopped, the transient recorder stops the data acquisition and saves the data. The resistance measurement is set back to single shot mode, and the resistance of the thermometer is measured. The laser is turned on at a low power again and another picture of the position of the laser focus is taken.
The program uses the "Faulhaber" software to control the motor, the "ThorlabsMotor" software to move the laser focus, the "MeanderResistance" software to read and configure the thermometer resistance measurements, the "Laser" software to set the appropriate laser power, the "Kamera" software to take the pictures, and the "TransCOM" software to access the API that controls the transient recorder functions.

#### "Thermospannungsmessung_statisch_V2"

This software is identical to "Thermospannungsmessung_statisch" in all aspects except for the way the measurement data is saved: The "Thermospannungsmessung_statisch" software uses the transient recorder to record the voltage and current at the constriction. It also records the voltage across the resistance and in order to have the same time base for all data, the lock-in signals to calculate the conductance are also recorded as analog signals coming from the auxiliary outputs from the lock-in amplifier. Unfortunately, the inputs of the transient recorder have offset drift. This means that the calculation of the conductance from the recorded data is prone to errors of up to 10 %. To avoid this problem, the new software uses only the digital data from the lock-in to also measure the voltage and current offsets generated by the thermopower.
In the software this leads to a small difference in the data acquisition: after the movement of the motor is started, the lock-in control software "LockInAcquisition" starts to write the data directly to the specified output file, and at the end of the measurement, the required data saving command for the transient recorder is no longer needed. Additionally the maximum motor speed is changed from 6000 to 200. The minimum motor speed may be adjusted depending on the sample and it be as low as 20. This means the motor will move slower during the interesting part of the trace and it will not move close that fast after the contact is opened. This reduced speed limit makes the measurement software fast enough to slow down the motor in order to acquire data points in the tunneling regime.

#### "Widerstandsmessung_Laserwechsel"

During the cool down or warm up process, this software measures the resistance of the platinum thermometer at different sample chamber temperatures. The laser is also used to heat the sample. To do that, the laser is moved to the side of the sample with the thermometer and the resistance is measured without and with the laser heating. The laser is moved to the other side and again, the resistance is measured without and with the laser heating. The laser is subsequently moved back to the side with the thermometer and the whole process is repeated until the user stops the program.

#### "Laserkalibrierung"

For this software, the laser powermeter needs to be placed in front of the cryostat window. It uses the source to increase the laser control voltage is small steps and records the resulting measured laser power.

### 5.7.4. Important programs for the molecule measurements

Since the IV-converter design uses up all measurement wires, no additional sample features can be recorded. This also means that there is only one useful measurement that can be performed with the setup in this configuration.

#### "Thermospannungsmessung_molekuele"

This is the main program that handles the complete measurement process. The measurement consists of repeated sets of eight traces. Every set starts with the laser positioned on

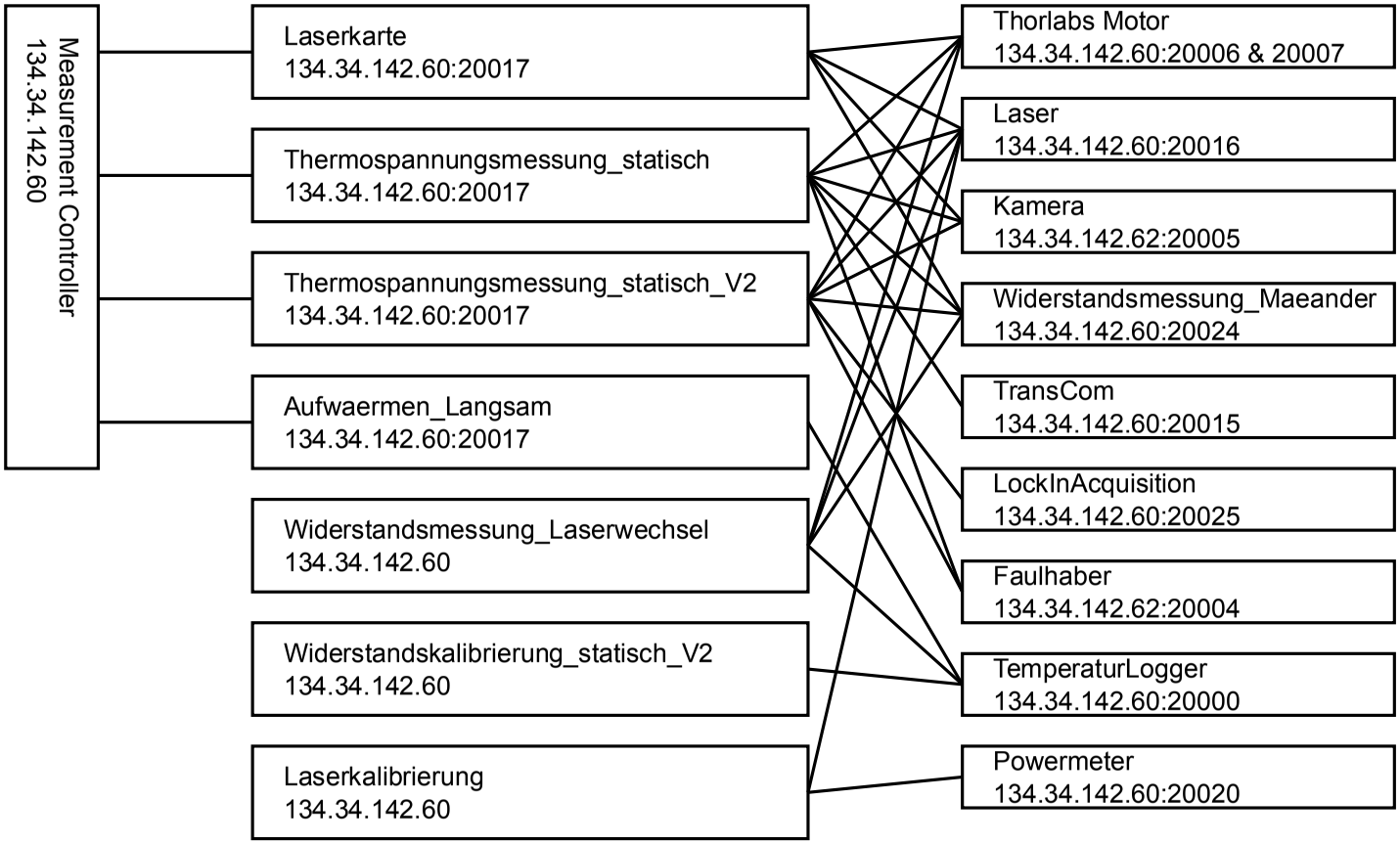



**Figure 5.14.:** This diagram shows the software used for a static measurement. The computer is indicated by the IP-address and if the software supports network commands, its port is also shown. The arrows indicate the control flow.

the sensor side of the sample. An opening and a closing trace are recorded with the laser being turned off. Then an opening and closing trace are recorded with the laser being turned on. After these four traces, the laser is moved to the other side of the constriction and opening and closing traces are recorded again with the laser turned off and on.
Each trace contains the following steps: At the beginning the laser is turned on at low power and the camera takes a picture of the sample to document the position and shape of the laser focus. Then the laser is set to either full power or off, depending on the desired laser state for the individual trace. The software waits for 5 s for the laser to settle. Then the motor starts to move in the direction given by the type of the trace at a constant speed and the data acquisition with the lock-in amplifier is started. At this stage the software continually checks the conductance of the sample until it reaches the limit of either $3\,G_0$ or $1 \cdot 10^{-7}\,G_0$ depending on the direction of the motion. Once the conductance has reached the limit the recording of the trace is stopped: the software stops the data acquisition, stops the motor, and stores the data. Additionally the speed of the motor is checked during the measurements and if the motor speed drops below 15 the motor has reached a limit given either by software or by the limit switches. This also stops the recording of the trace. Afterwards the laser is turned on at low power again and the camera takes another picture to document the shape and position of the laser focus at the end of the trace.

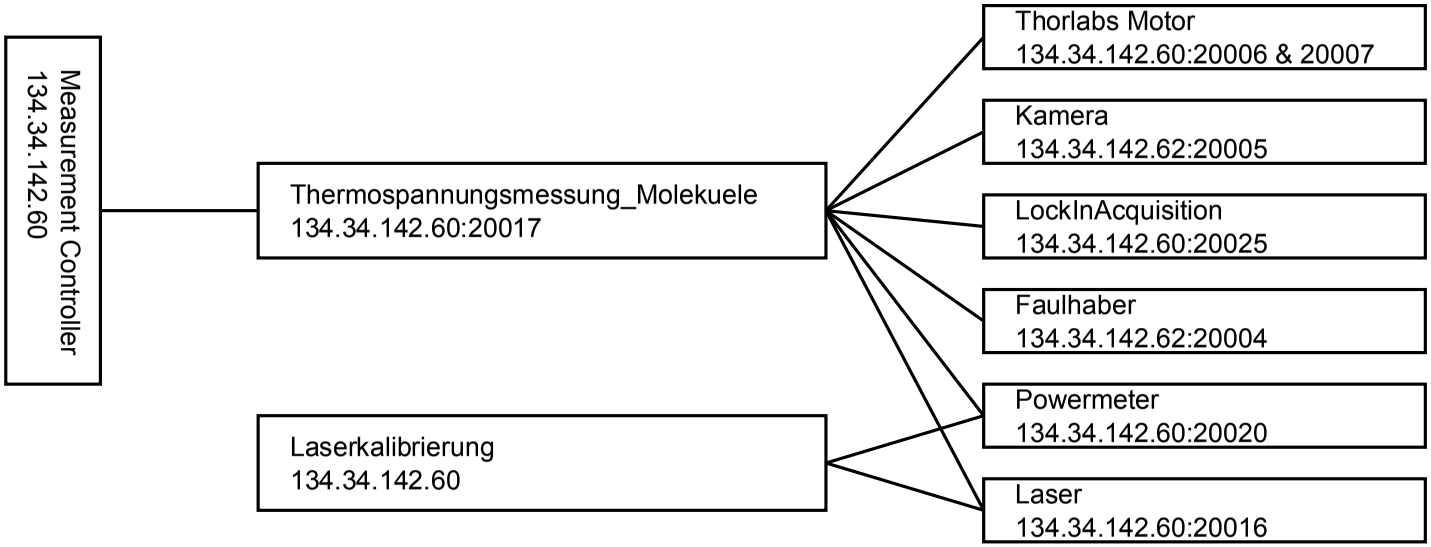



**Figure 5.15.:** This diagram shows the software used for a molecule measurement. The computer is indicated by the IP-address and if the software supports network commands, its port is also shown. The arrows indicate the control flow.

# 6. Pulsed Measurement

## 6.1. Temperature Measurements

Continuing the work of Bastian Kopp [Kop16], who started thermopower measurements on gold break junctions, I tried to verify the temperature simulations by moving the laser to different positions on the sample. The automation of the setup allowed to measure grids with up to 80 by 80 pixels on the sample. Since the influence of external interference was significantly reduced with the amplifiers in the IceCUBE, not only the change of the resistance could be measured, but the whole waveform was analyzed.
Figure 6.1 shows a typical curve of the resistance as function of time. The setup was configured as described in section 5.3.1. A bias voltage was applied, every 34 ms the laser was pulsing with a pulse 4 ms long with constant power, and five seconds of data were recorded for every bias voltage ($-100$,$-50$,0,50, and 100 mV). This resulted in five points in an IV-curve at every time, which were fitted with a linear curve. The slope of the curve is the resistance at the given time.
To compare these results with the simulations, five independent quantities were calculated:

- The equilibrium resistance reached just before the laser pulse, was calculated by fitting the last 4 ms of the data with a linear function and extrapolating this function to 34 ms. This quantity has the dimensions of a resistance.

- The first millisecond of the data was also fitted linearly to determine the slope at the beginning of the pulse. To reduce the correlation with the different parameters, the slope was divided by the resistance change. It has the dimensions of an inverse time.

- The fall of the resistance directly after the laser pulse was analyzed similarly, by evaluating the first millisecond after the pulse.

- Fitting the last millisecond of the laser pulse, the slope at the end of the pulse was determined in the same way.

- Evaluating the same linear function at the time of the end of the laser pulse gives the resistance change, which was divided by the equilibrium resistance to obtain a dimensionless quantity that does not depend on the exact details of the resistance wire.

The fit curves are also shown in figure 6.1. The results were well reproducible from one measurement to the next and similar values could be measured for the quantities that do not depend on the exact resistance. The reproducibility and the independence are demonstrated by the data in figures 6.3 to 6.7. In order to verify that the simulations

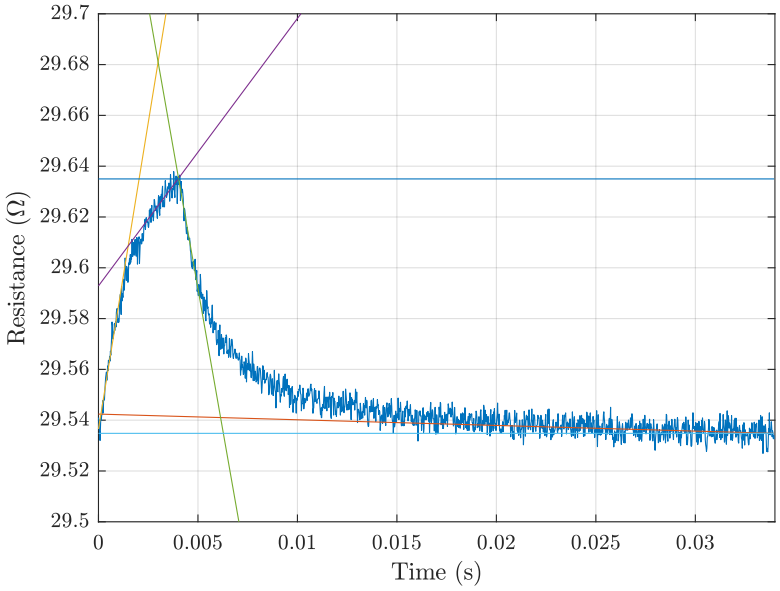
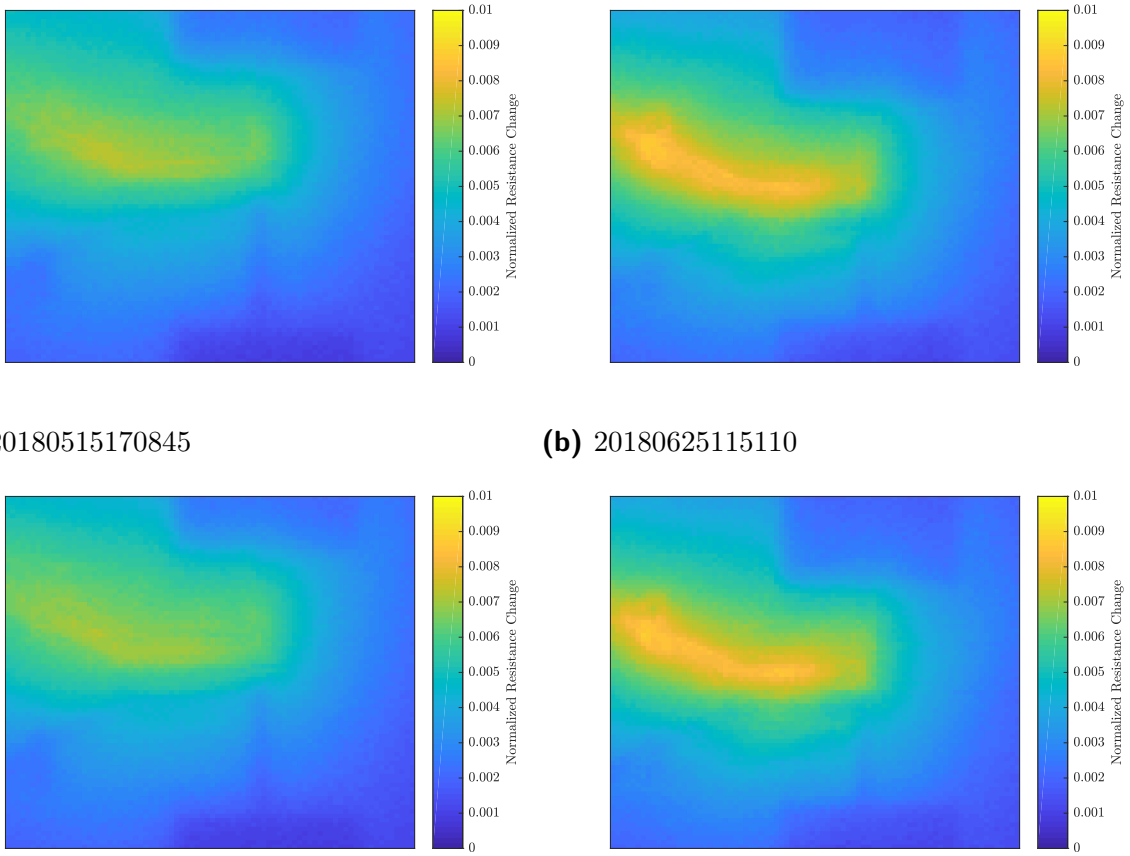



**Figure 6.1.:** This graph shows a typical time dependence of the resistance of the sensing leads. The laser pulse for the heating is 4 ms long, and starts at $t = 0$. Also shown are linear functions fitted to different sections of the pulse. The yellow line shows the slope at the beginning of the pulse, the purple one at the end of the pulse, the green one after the pulse, and the red fit function is used to determine the equilibrium resistance. They are used in the analysis to extract the slopes and the resistance change. The resistance at the end of the pulse (dark blue line), as well as the equilibrium resistance (bright blue line) are also shown. The data is taken from measurement 20180515170845 at the position $(x = 22, y = 34)$ and lead 2.

**Figure 6.2.:** The figure shows the area of the sample which is scanned by the laser focus during the measurements presented in figures 6.3 to 6.7.



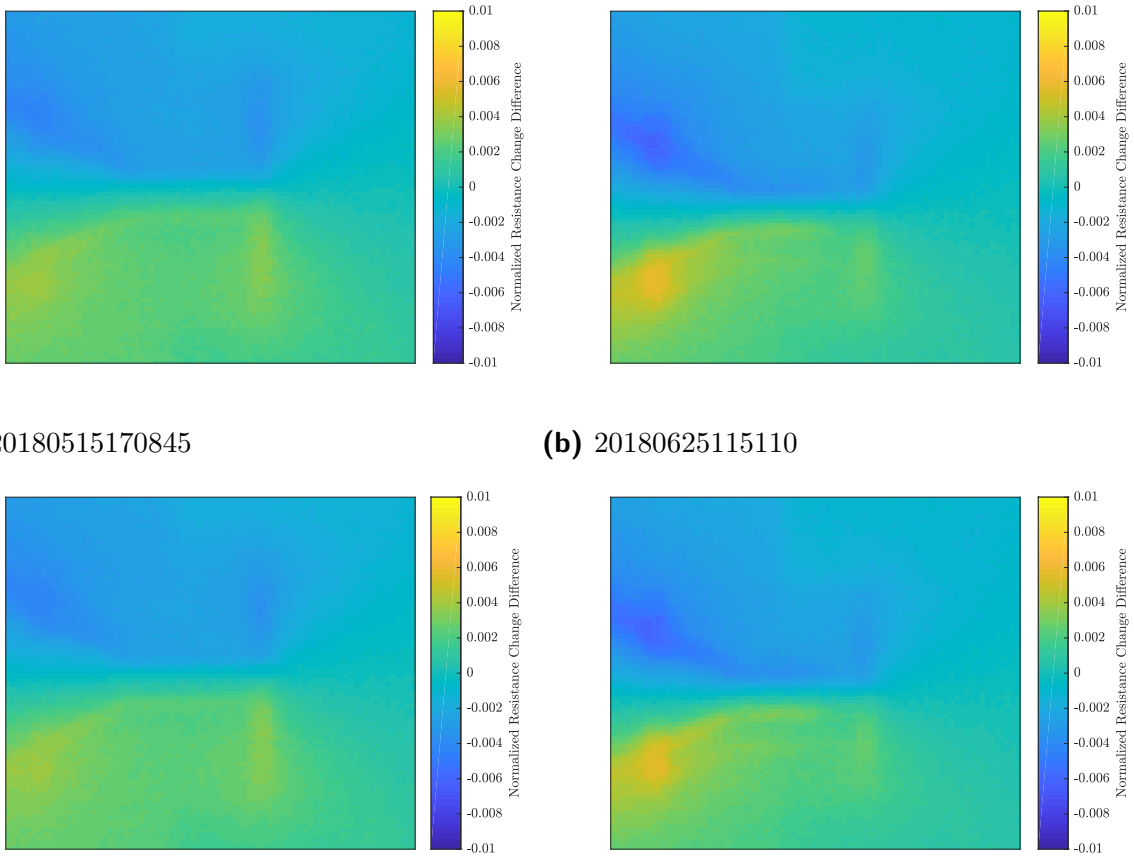

**(a)** 20180515170845

**(b)** 20180625115110

**(c)** 20180519082655

**(d)** 20180629082344

**Figure 6.3.:** The panels show the normalized resistance change as a function of the laser heating position. The scanned area is shown in figure 6.2. The coloring of each pixel represents the normalized resistance change at the given position. Panels a and c are measured on sample T0184M and b and d on T0176M, all at 86 K. The normalized resistance change is reproducible on the same sample, but is influenced by the series resistance and therefore slightly different from sample to sample. Similar data was obtained for the other lead.



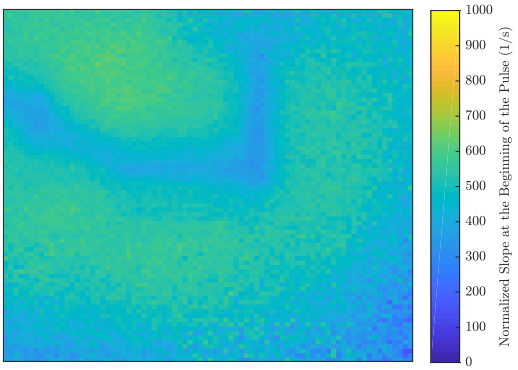

**(a)** 20180515170845

**(b)** 20180625115110

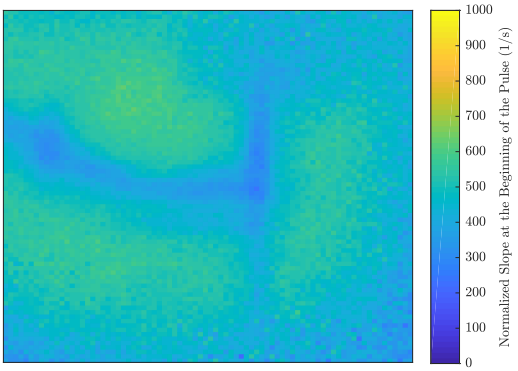
**(c)** 20180519082655

**(d)** 20180629082344

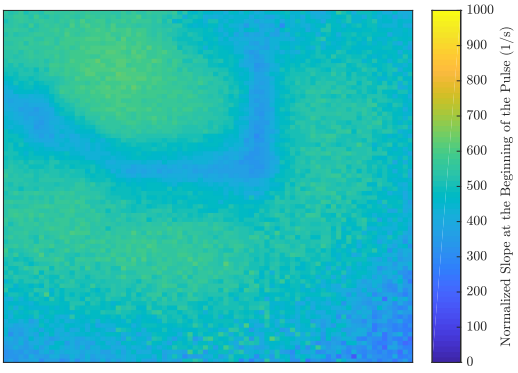
**Figure 6.4.:** The panels show the difference of the normalized resistance change as a function of the laser heating position. The scanned area is shown in figure 6.2. The coloring of each pixel represents the the difference of the normalized resistance change at the given position. Panels a and c are measured on sample T0184M and b and d on T0176M, all at 86 K. The difference of the normalized resistance change is reproducible on the same sample, but is influenced by the series resistance and therefore slightly different from sample to sample.



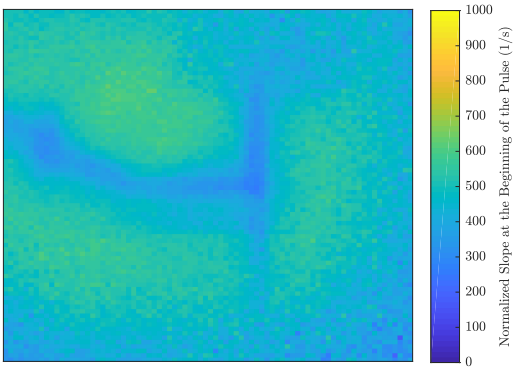

**(a)** 20180515170845




**(b)** 20180625115110

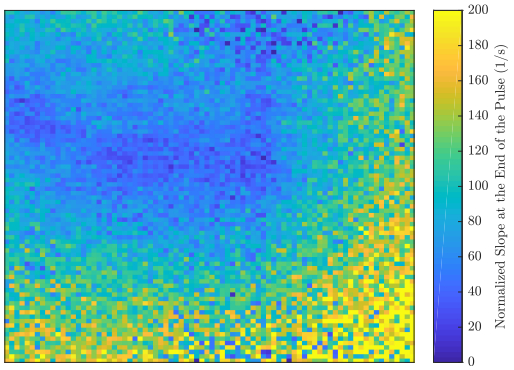



**(c)** 20180519082655

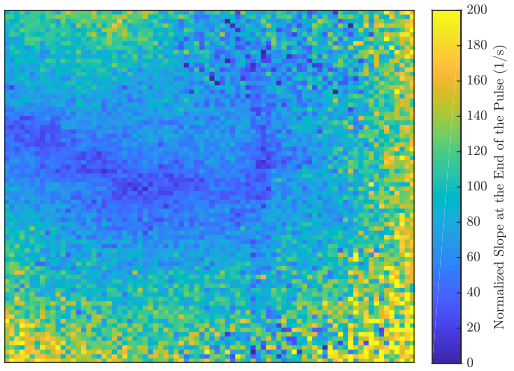



**(d)** 20180629082344

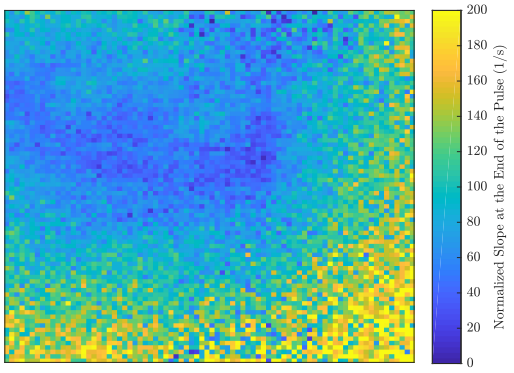
**Figure 6.5.:** The panels show the normalized slope at the beginning of the pulse as a function of the laser heating position. The scanned area is shown in figure 6.2. The coloring of each pixel represents the normalized slope at the beginning of the pulse at the given position. Panels a and c are measured on sample T0184M and b and d on T0176M, all at 86 K. The normalized slope at the beginning of the pulse is reproducible from sample to sample. Similar data was obtained for the other lead.



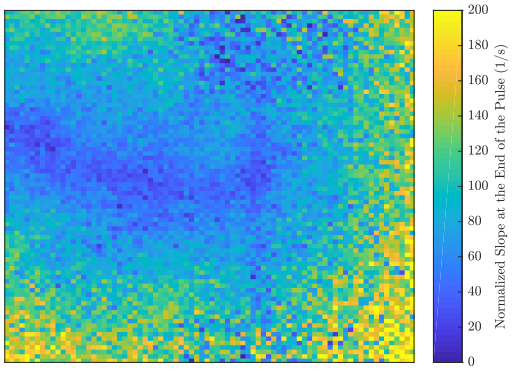

**(a)** 20180515170845




**(b)** 20180625115110

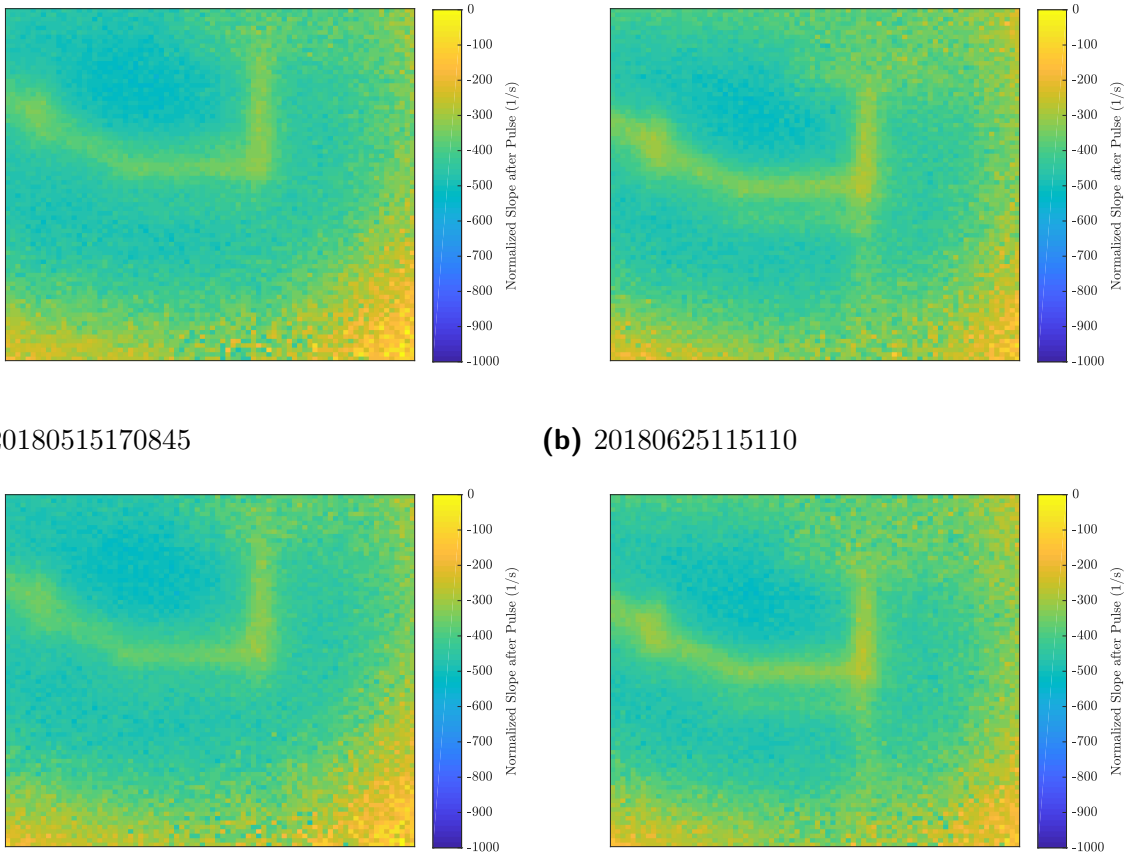



**(c)** 20180519082655




**(d)** 20180629082344

**Figure 6.6.:** The panels show the normalized slope at the end of the pulse as a function of the laser heating position. The scanned area is shown in figure 6.2. The coloring of each pixel represents the normalized slope at the end of the pulse at the given position. Panels a and c are measured on sample T0184M and b and d on T0176M, all at 86 K. The normalized slope at the end of the pulse is reproducible from sample to sample. Similar data was obtained for the other lead.




**(a)** 20180515170845

**(b)** 20180625115110

**(c)** 20180519082655

**(d)** 20180629082344

**Figure 6.7.:** The panels show the normalized slope after the pulse as a function of the laser heating position. The scanned area is shown in figure 6.2. The coloring of each pixel represents the normalized slope after the pulse at the given position. Panels a and c are measured on sample T0184M and b and d on T0176M, all at 86 K. The normalized slope after the pulse is reproducible from sample to sample. Similar data was obtained for the other lead.

| | | | |
|---|---|---|---|
| | density | 19 300 | $\mathrm{kg\,m^{-3}}$ |
| Gold | heat capacity | 128 | $\mathrm{J\,kg^{-1}\,K^{-1}}$ |
| | thermal conductivity | free parameter | |
| | density | 1420 | $\mathrm{kg\,m^{-3}}$ |
| Kapton$^{\mathrm{TM}}$ | heat capacity | 371.45 | $\mathrm{J\,kg^{-1}\,K^{-1}}$ |
| | thermal conductivity | 0.493 | $\mathrm{W\,m^{-1}\,K^{-1}}$ |
| | density | 1300 | $\mathrm{kg\,m^{-3}}$ |
| polyimide | heat capacity | 1100 | $\mathrm{J\,kg^{-1}\,K^{-1}}$ |
| | thermal conductivity | 0.15 | $\mathrm{W\,m^{-1}\,K^{-1}}$ |

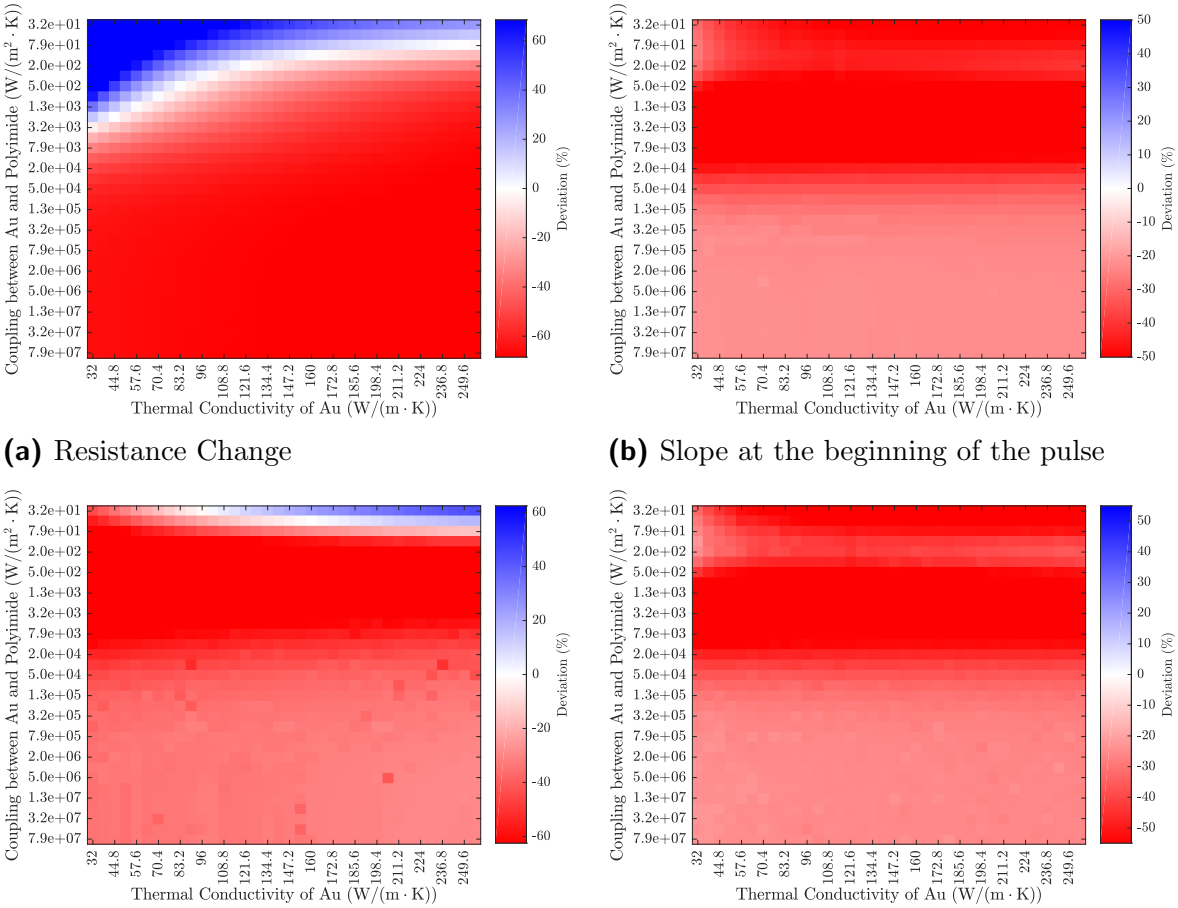
**Table 6.1.:** Material parameters used in the VD_11 simulations

accurately describe the samples, these values were then compared to the simulations[1] of the temperature distribution in a sample with similar geometry. The temperature dependence of the resistance of the sample during cool down was measured, approximated with an analytic function, and rescaled to be used as the temperature dependent resistivity of the gold film. Based on this resistivity, the simulated temperature distribution was used to simulate the current flow through the sample from which the simulated resistances of the wires could be derived. The assumed material properties in this simulation are listed in table 6.1.
The comparison with the experimental data is shown in figure 6.8. The simulation results were analyzed in exactly the same way as the experimental results. The comparison between simulation and experiment shows clearly that with only the thermal conductivity of the gold film and coupling between the gold and the polyimide layer suitable parameter combinations can be found for each parameter individually. An overlap of the areas of the parameter ranges would mean that the combination can describe the complete temperature characteristics. However, the areas of those parameter ranges do not overlap.
Different reasons might cause this mismatch between simulation and experiment. Assuming that the simulations are numerically correct, they are still only as good as the material properties and hidden assumptions in the data analysis.
The thermal properties of the materials were taken from the integrated material library [2]. The hidden assumptions are:

- The resistance changes are linear with the laser power. This was tested by varying the laser power. The results are shown in figure 6.9.
- The resistor has enough time to get to the equilibrium temperature. To test this, the repetition rate was changed. The results in figure 6.10 show that the equilibrium values do not change if the repetition rate is lowered.
- The slopes do not depend on the laser power. This is shown in figure 6.11.

The mismatch might be caused by assuming that the laser light is absorbed either at the surface of the gold film or at the surface of the Kapton$^{\mathrm{TM}}$ substrate, but various other

[1] VD_10_20180518121539
[2] It will be shown later that the included value for the thermal conductivity is wrong, but fixing this is still not enough to achieve good agreement.




**(a)** Resistance Change

**(b)** Slope at the beginning of the pulse

**(c)** Slope at the end of the pulse

**(d)** Slope after the pulse

**Figure 6.8.:** The figures show the deviations between the experimental values and the results of the simulations with different parameters. The laser is at ($x = 10\,\mu\text{m}, y = -10\,\mu\text{m}$) in the simulation VD_11_20180613094804. The experimental values are taken from 20180625115110 at $x = 0058, y = 0059$ and lead 2.

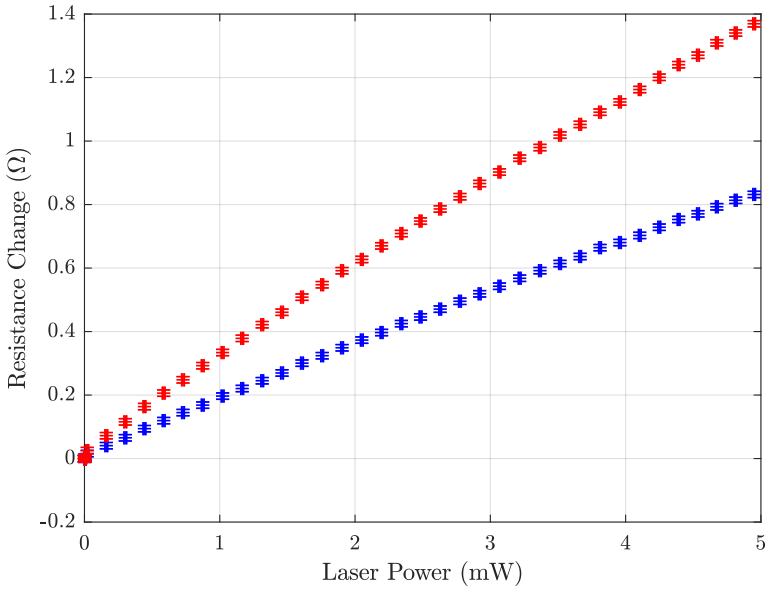



**Figure 6.9.:** The figure shows the change of the resistance as a function of the power of the laser heating pulse. The red data is the resistance change of the lead closer to the heating spot, the blue data the one further away. The change of the resistance of the two measurement leads is proportional to the used laser power. This indicates that the sample is still in the linear regime and the temperature changes are small enough to assume constant material parameters. The data is taken from 20180515170845.

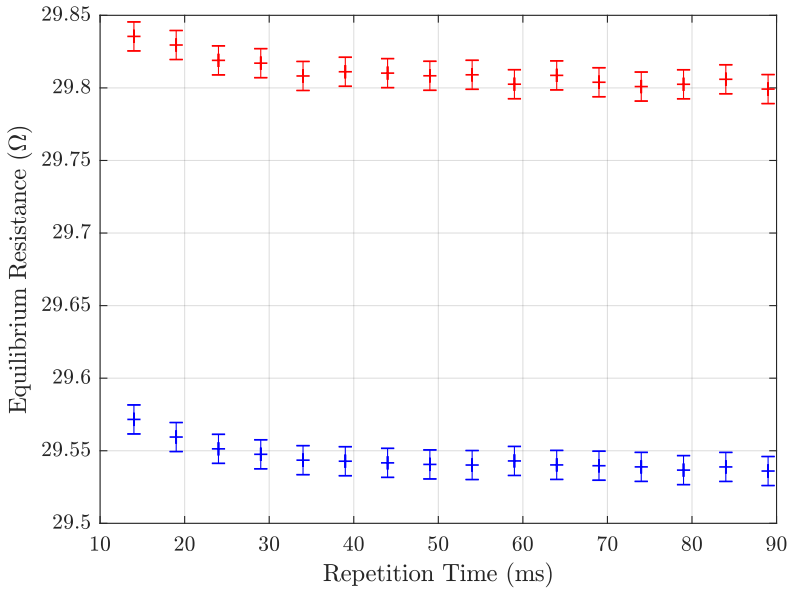
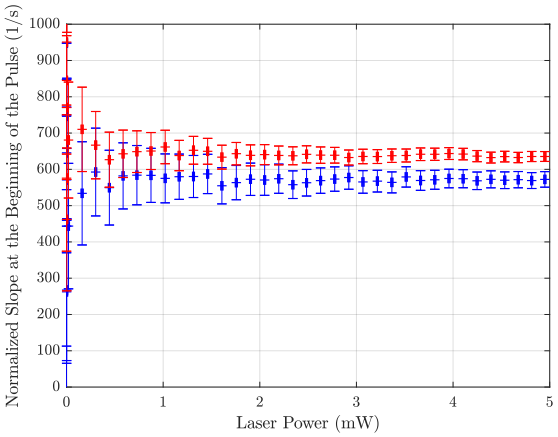
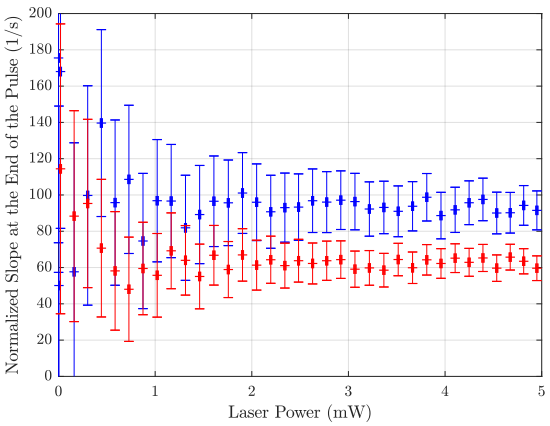


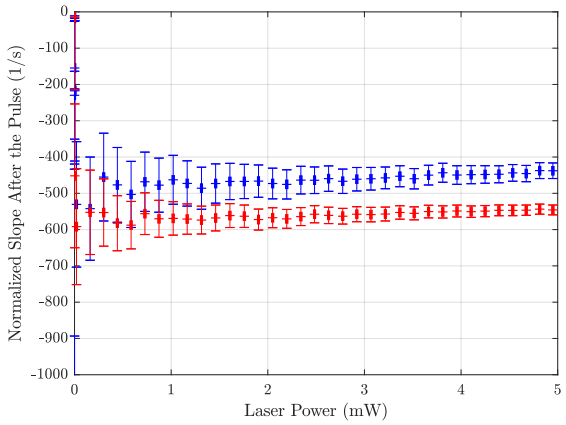

**Figure 6.10.:** The equilibrium resistance of the two leads (red on the heated side, blue on the other) is shown for different repetition times. For small times, the laser pulses are so frequent that the sample does not have enough time to cool completely before the next pulse. The time used in the normal measurements is 34 ms which is just long enough. The data is taken from 20180523085200.




**(a)** Normalized slope at the beginning of the pulse




**(b)** Normalized slope at the end of the pulse




**(c)** Normalized slope after the pulse

**Figure 6.11.:** The normalized slopes of both leads (red on the heated side, blue on the other) are independent of the laser power. For low powers, the resistance change is small and therefore the noise is large. The data is taken from 20180823121900 and the laser power from 20180529121351.

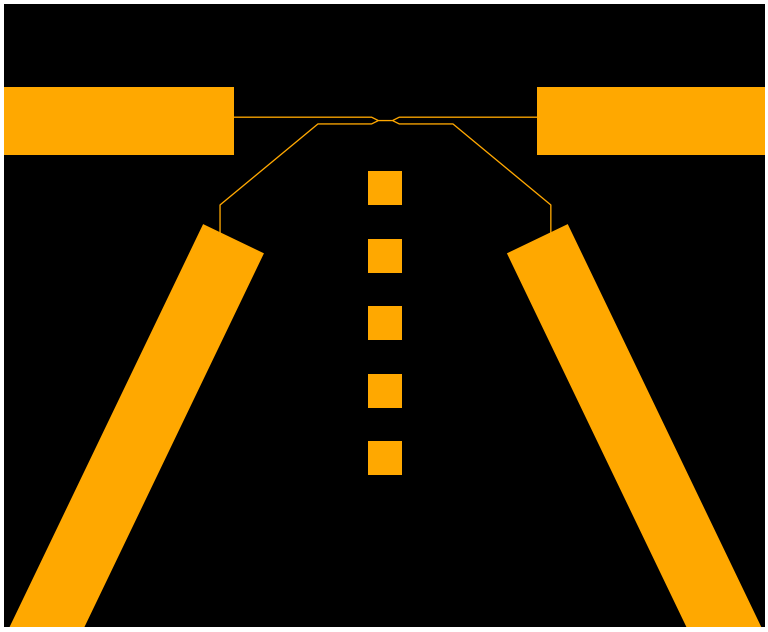


**Figure 6.12.:** The structure of the samples designed to simplify the simulation consists of a small gold wire and five squares. The squares have a side length of 10 µm and their centers are 20, 40, 60, 80, and 100 µm away from the center of the resistance. The geometry of the resistance is designed to be small enough that the temperature distribution inside the resistance can be approximated as homogeneous, but still large enough to be reliably fabricated.

material parameters might also be responsible. Varying all possible parameters is not reasonable because of the excessive computation times and the nearly unlimited number of combinations leading to possibly matching but unphysical parameter sets.

### Samples tailored for simulations

To reduce simulation times and to reduce the amount of necessary parameters, samples were created that simplify the simulation by allowing an easy measurement of the temperature. The design is a small gold wire with contacts for a four wire resistance measurement. By that, the temperatures from the simulation can be compared directly to the experimental values and the difficult derivation of the resistance from the temperature distribution in the geometry is no longer needed.
To simplify the simulations further, gold squares were placed next to the wire at different distances. The structure is shown in figure 6.12. The gold pads provide a well understood position for the laser to be absorbed on. Since the Kapton$^{TM}$ substrate appears black but gets translucent when ground to a wedge shape, the simulation assumption that all the laser power is absorbed at the Kapton$^{TM}$ surface is wrong. I was unable to find any optical constants for this material and the supplier did not answer any requests, therefore the absorption properties of the Kapton$^{TM}$ remain unknown. The gold pads were thus very useful since they allowed to isolate this unknown quantity and to perform the simulations with fewer free parameters. Also, the measurement and the simulation could be reduced

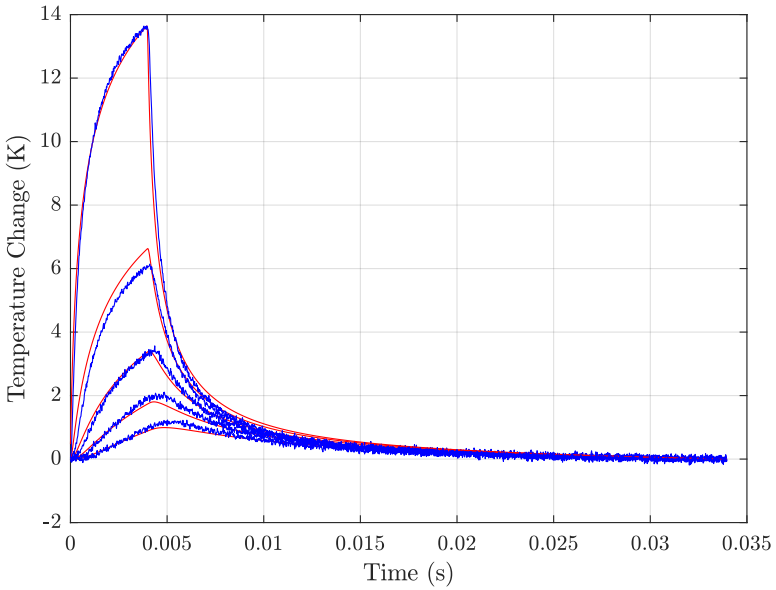



**Figure 6.13.:** The blue curves show the measured resistances from measurement 20190523133059 at $y = 24, 50, 73, 96, 123$. The measurement was performed on sample T0003T at a sample chamber temperature of 86 K with 2 mW of laser power in front of the cryostat. The red curves are the corresponding simulation results from VI_03_20190820121847. The simulations were scaled by 1.06 and use the thermal conductivity of polyimide as $2\,\mathrm{W\,m^{-1}\,K^{-1}}$ and a coupling between the gold and the polyimide of $63\,096\,\mathrm{W\,m^{-2}\,K^{-1}}$.

to only the five center points of the pads. It is therefore possible to compare not only the extracted slopes but the complete time evolution.
To match the simulations and the experiment, the data has to be slightly time shifted. The simulations do not include any mechanism to extract the heat, thus the temperature in the end is higher than the temperature at the beginning. This is corrected by subtracting a linear slope. In the experiment, the sample is at slightly elevated temperatures compared to the sample chamber. Since simulations start at exactly the sample chamber temperature, the resulting data is shifted to match the experimental starting temperature. The laser power may also vary by a few percent, hence the simulated temperatures are scaled to have matching maxima of the temperature on the closest gold pad. The results are shown in figure 6.13. The necessary scaling by 1.06 is easily within the errors of the laser power measurement, since the laser power cannot be measured inside the chamber, and therefore the attenuation by the chamber windows and the lens as well as the effects of the windows of the cryostat were measured separately. The good agreement with the experiment was achieved by using the thermal conductivity for Kapton™ from [Nat] instead of the values from the material library, and by having a thermal conductivity of $2\,\mathrm{W\,m^{-1}\,K^{-1}}$ for the polyimide layer formed from the Durimid 115A™. This value exceeds the known literature values for polyimides by an order of magnitude. These successful simulations were then used to also include the illumination of the Kapton™ with

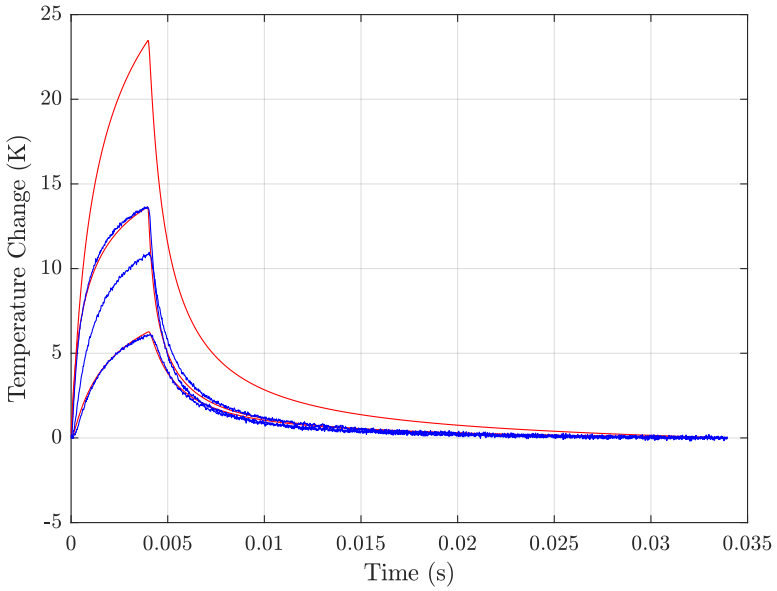



**Figure 6.14.:** The blue curves are the data taken from measurement 20100523133059 on sample T0003T at $y = 96, 109$ and 123 with 2 mW laser power measured in front of the cryostat and a sample chamber temperature of 86 K. The two extreme blue curves are measured with the laser on the two gold pads closest to the resistance. The curve in between is measured with the laser on the Kapton$^{TM}$ substrate. The red curves are the results of the simulation VJ_02_20191021173144 with the inverse absorption length of Kapton$^{TM}$ as $1 \cdot 10^6\,\mathrm{m}^{-1}$, the coupling between gold and polyimide as $63\,096\,\mathrm{W\,m^{-2}\,K^{-1}}$, and the thermal conductivity of the polyimide as $1.5\,\mathrm{W\,m^{-1}\,K^{-1}}$. The simulated temperatures were multiplied by a factor of 0.956.
This simulation can describe the time evolution of the temperature nicely when the laser is absorbed by the gold pads, but with the used absorption depth in the Kapton$^{TM}$ the simulated temperature changes are too big when the laser heats the substrate.

the absorption depth as a free parameter. It was found that the simulated temperature changes were either too big or too slow, as shown in figures 6.14 and 6.15.
Hence I doubt that the simulation results can be used for a reliable description of the complete geometry of the break junction sample style.

## 6.2. Voltage Measurements

Regardless of the unknown temperature distribution in the sample, thermovoltage measurements were performed. The dominating problem of these measurements is the fact that a complete measurement set takes about 25 s during which the contact has to remain stable. However, in the course of a single laser pulse the sample structures expand and contract significantly. This changes the contact geometry and the conductivity with every

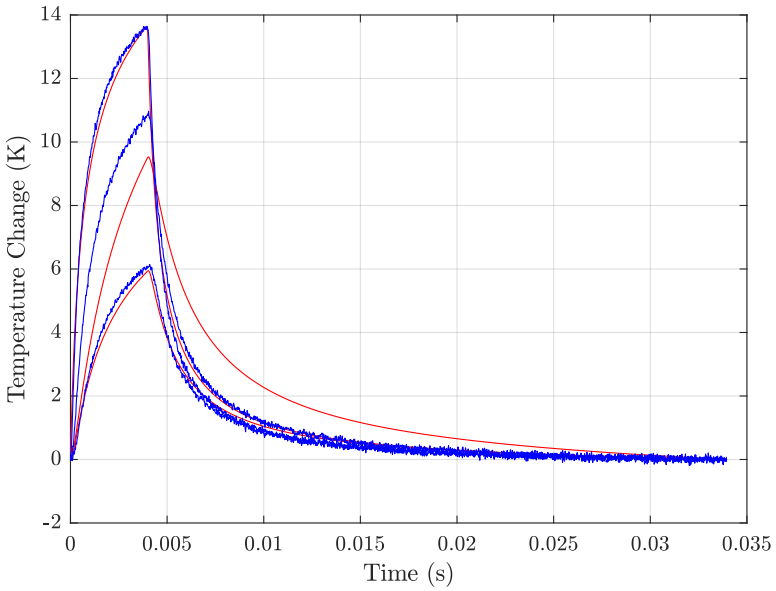



**Figure 6.15.:** The blue curves show the same measured data as in figure 6.14. The red curves are the results of the simulation VJ_02_20191029123024 with the inverse absorption length of Kapton$^{TM}$ as $2 \cdot 10^4\,\mathrm{m}^{-1}$, the coupling between gold and polyimide as $80\,000\,\mathrm{W\,m^{-2}\,K^{-1}}$, and the thermal conductivity of the polyimide as $1.2\,\mathrm{W\,m^{-1}\,K^{-1}}$. The simulated data was stretched by a factor of 0.885.
For slightly different parameters than before, a good agreement between simulation and experiment can still be achieved with the laser heating the gold pads. When the laser heats the substrate, the temperature changes have approximately the correct size. However, the simulated time evolution is clearly too slow.

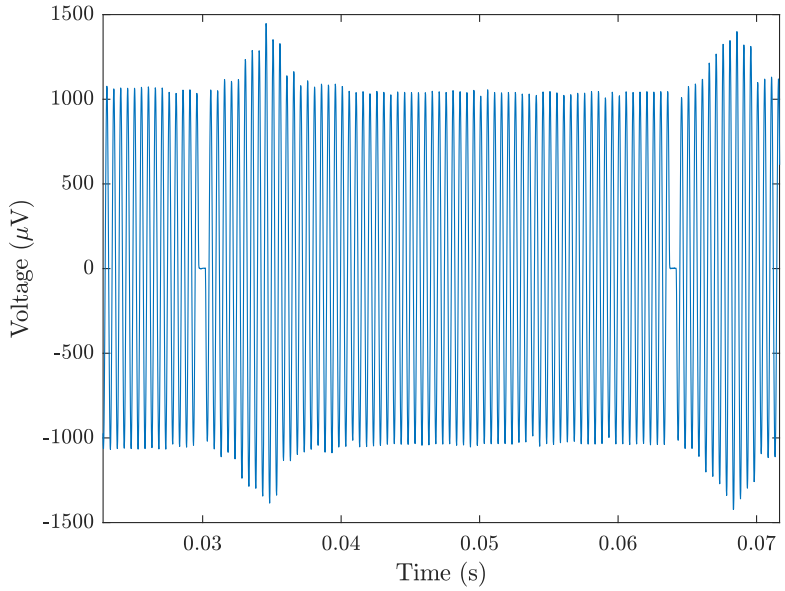



**Figure 6.16.:** The raw measured voltage as a function of time in measurement 20181116124756 (trace 20181117000054) on sample T0177M at room temperature shows the significant change of the conductance during the laser pulse.

pulse. During the cooling, the contact does not necessarily return into the same geometry as before. This explains why not a single contact remained unchanged during the measurement set. If the conductance changes during the measurement, the data points at higher bias currents do not fit to those at lower bias currents. This will influence the slope and position of the linear IV-fit. Because the position of this fit is the generated thermovoltage, the resulting data is completely unreliable and useless.

### 6.2.1. Fast Measurements

To get around the problems generated by the instability of the contact from pulse to pulse, the experimental approach was changed to measure the different points of the IV-curve within the same pulse. This means, the bias is modulated quickly so that the influence of the changing conductance during the pulse remains low.
The change of the measured voltage as a function of time is displayed in figure 6.16. The change of the amplitude during the pulse is clearly visible, which demonstrates the extent to which the contact is deformed due to the thermal expansion.
The data between 2 and 4 ms are used to generate an IV-curve which is fitted with a linear function. The same fitting is performed with the data directly before the pulse. The voltage at zero current is calculated for both fits. The voltage offset of the amplifier can be subtracted by taking the voltage before the pulse as reference. However, the bias currents of the amplifier generate conductance dependent offsets. These are also corrected. The result of two of these measurements is shown in figures 6.17 and 6.18. In the first measurement, the laser destabilizes the contact so that very few data points between 1

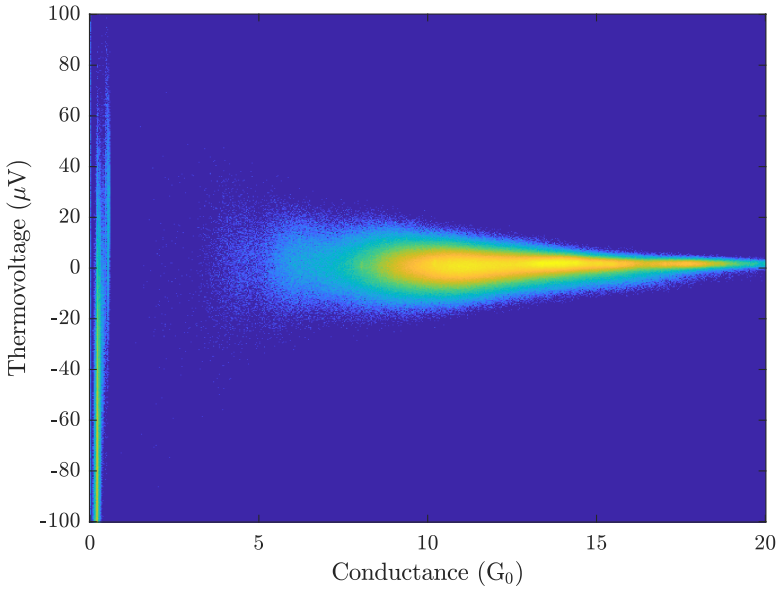



**Figure 6.17.:** This figure shows the measured thermovoltage as a function of the conductance. Regions which occurred often are shown in yellow, regions without data points are dark blue. The results of measurement 20181116124756 on sample T0177M at room temperature show no significant thermovoltage. The lack of data points in the few conductance quanta region shows the result of the instability of the contacts due to the laser induced thermal expansion.

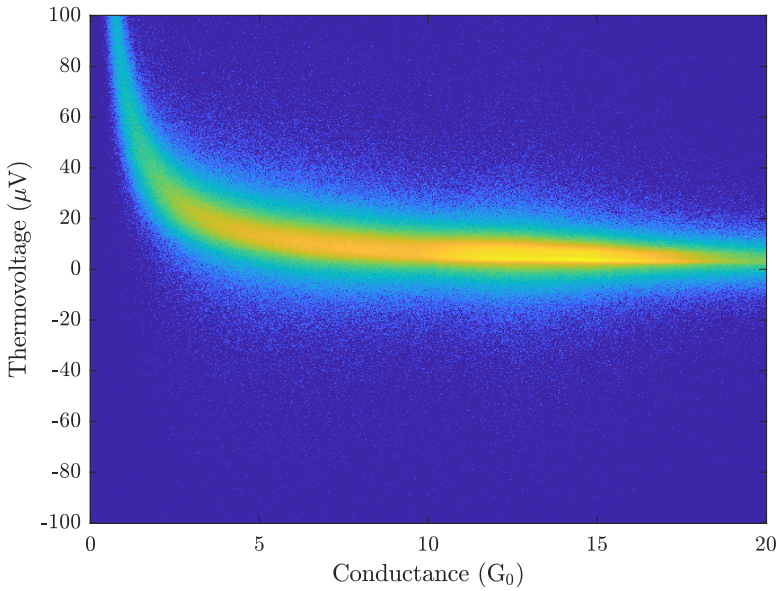



**Figure 6.18.:** This figure shows the measured thermovoltage as a function of the conductance. Regions which occurred often are shown in yellow, regions without data points are dark blue. The results of measurement 20181121111252 on sample T0181M at room temperature with 0.5 mW laser power measured in front of the cryostat show the characteristic shape of an incorrect compensation of the amplifier offset current.

and $4\,G_0$ can be observed. The second measurement does not show this problem, however, the shape of the curve is very similar to the shape created by an incorrect current offset. Thus this data seems not trustworthy.

### 6.2.2. Solutions for the Instability

To reduce the destabilization due to the laser pulses, the following concepts might be applied:
A substrate material might be found with a thermal expansion similar to gold. With such a substrate the expansion of the gold may be compensated. Since the samples need to have an electrically isolating layer between the gold electrodes and the substrate, the gold cannot be coupled directly to the substrate. This means that the temperature distributions in the gold and the substrate will have different time dependences. For a good compensation, the differences in the time dependences would have to be carefully controlled.
Using less heating power is obviously reducing the thermal expansion, but it also directly reduces the temperature difference at the constriction, thereby reducing the measured thermovoltage.
Performing the measurement at very low temperatures is another possibility to reduce the thermal expansion. However, the theory predicts that the thermopower diminishes with the absolute temperature. This also reduces the measured thermovoltage, which makes the measurement more difficult.
The last possibility is to have a contact which is insensitive to small geometric changes. This might be possible in molecular contacts. In metallic contacts, however, the thermopower is strongly dependent on the exact geometry of not only the central atom but also the banks on either side.
Even if any of these solutions to measure the thermovoltage are realized, accurate knowledge of the temperature difference is still necessary to calculate the thermopower. The lack of correct simulations for the temperature distribution makes this calculation of the thermopower impossible or at least very error-prone.

# 7. Static Measurement

The static thermopower measurements were designed to work around the expansion problems of the pulsed measurements. In contrast to the pulsed measurements, the static measurements have the laser continuously on or off for the complete bending process. Thus the thermal expansion does not change during an opening or closing trace. Therefore the contacts can remain stable. In the pulsed measurements, the offset of the amplifiers could be corrected by using the data points before the pulse as a reference. In the static measurements this is not possible, therefore traces with and without heating were compared. In order to reduce the dependence on the simulation results, a resistance thermometer was integrated onto the sample to get an estimate for the temperature.

## 7.1. Measurement

During the measurement successive opening and closing traces were recorded.
One trace comprised the following steps:

- The laser was turned on at a low power and a photo of the sample with the position of the laser focus was taken.
- The laser was configured to the measurement power
- The sample was given 5 s to equilibrate.
- The resistance of the thermometer on the sample was measured in a single offset corrected measurement with the Keithley 2400 SourceMeter SMU$^{TM}$. This value is used to determine the temperature in the analysis.
- The SourceMeter was reconfigured for free running successive measurements.
- The transient recorder was started to record the current and voltage across the constriction, the voltage across the resistance thermometer, the laser power measured by the photo-diode, and the X and Y values of the voltage and current channels of the lock-in amplifier.
  In the later versions of this measurement, the voltage and current are measured directly by the lock-in amplifier. Then at this point, the data acquisition of the lock-in amplifier is started, and the X and Y values as well as the dc values are stored. The dc values was obtained by setting the demodulation frequency to 0 Hz.
- The motor was started to change the bending of the sample in the desired direction.

- The digital output of the lock-in amplifier was evaluated every 0.1 s and the conductance calculated. This conductance was used to calculate the moving speed of the motor:

$$1\,\mathrm{G_0}^2 \cdot \frac{1}{G^2} + 0.1\,\mathrm{G_0}^{-2} \cdot G^2$$

  If this resulted in a value lower than the minimum speed of 40, the minimum motor speed was used instead. And in the later versions, a maximum motor speed of 200 was used. The minimum speed may be adjusted to suit the breaking behaviour of the sample.

- Once the conductance had reached the limit for the measurement, the motor was stopped.

- The data recording of the transient recorder was stopped and the acquired data was saved into a `.tpc5` file. In the later versions, the data acquisition with the lock-in was stopped.

- The Source Meter was configured to perform another single measurement of the resistance thermometer.

- Then the laser was set to low power and another photo of the position of the focus was taken, to document the movement of the sample during the bending.

The waiting for the conductance limit could be ended prematurely and deliberately if manual intervention like coolant transfer was necessary. In this case, the following trace would be executed with the same direction and laser power and position to continue where the trace before stopped.
The combination of an opening and a closing trace forms one cycle. The measurement started with the laser focus positioned on the side of the thermometer, but the laser was turned off. Then one cycle was recorded. After this, the laser was turned on to a power of about 10 mW measured in front of the cryostat windows. Then a second cycle was recorded which completed the measurement on one side. Afterwards, the laser was moved to the other side of the constriction and the complete process repeated. Thus a complete set of opening and closing traces with and without laser heating on each side of the constriction was acquired. This process was repeated over and over again to get one measurement with sufficient statistics.

## 7.2. Simulations

Simulations for a static temperature distribution in the sample were performed. The simulation geometry consisted of an $18 \times 3 \times 1.998$ mm block as the substrate. Another $18 \times 3 \times 0.0015$ mm block was placed on top to represent the polyimide which survived the etching process. On top of this were two layers with the same structure, a lower layer with a thickness of 500 nm representing the remaining polyimide under the gold, and an upper layer with 80 nm of thickness representing the structured gold film. This structure reflected the structure of the first lithography step, except the markers and labels left out.

| | | | |
|---|---|---|---|
| | density | 19 300 | $\mathrm{kg\,m^{-3}}$ |
| gold | heat capacity | 128 | $\mathrm{J\,kg^{-1}\,K^{-1}}$ |
| | thermal conductivity | extracted from [Wan13, Fig. 6] | |
| | density | 1420 | $\mathrm{kg\,m^{-3}}$ |
| polyimide | heat capacity | 371.45 | $\mathrm{J\,kg^{-1}\,K^{-1}}$ |
| | thermal conductivity | taken from [Nat] | |
| | density | 8820 | $\mathrm{kg\,m^{-3}}$ |
| bronze | heat capacity | 377 | $\mathrm{J\,kg^{-1}\,K^{-1}}$ |
| | thermal conductivity | taken from [Sim92, Eqn. 21-11] | |

**Table 7.1.:** These material parameters are used in the simulations for the static samples (VC_04). Since the simulation is static, only the thermal conductivities influence the result.

To create a free standing bridge, the structured polyimide layer was removed in the center below the constriction to leave 500 nm of free standing gold. The underetching of the rest of the structure was ignored, as well as all the structures of all further lithography steps. Three materials were used in the simulation: gold, polyimide, and bronze. The material properties used in the simulation are listed in table 7.1. Since the simulations were not time-dependent, only the thermal conductivities are relevant. The thermal conductivity for the polyimide layer was taken from Kapton$^{\mathrm{TM}}$ since the properties of cured Durimide 115A$^{\mathrm{TM}}$ could not be found.
All boundaries on the surfaces of the sample were assumed to be thermally isolated, except for a 0.2 mm strip under the constriction on the bottom of the sample to couple the sample to the chamber temperature. The constriction was assumed to be thermally isolating. The laser heating was implemented as a 10 µm square at an angle of 45° with homogeneous heat flow which corresponds to the absorbed power of the laser at 10 mW attenuated by the optical components to 79 %. The reflectivity of the gold film of 63.5 % [Joh72] was included.
For the analysis, the temperature was extracted at two points close to the constriction on either side. To get the temperature that would be measured by the thermometer a polygon was defined in the area of the wiggles of the Pt-wire. This area was then used to calculate the average temperature.
For the analysis of the experiments, the ratio of the temperature increase at the constriction and at the thermometer was used. This result is shown in figure 7.1. The sample layout is designed to minimize the influence of the exact material parameters on the measurement result. Therefore the simulation results depend weakly on the different assumed properties. This is shown in figures 7.2 to 7.8 where the laser heating position, the thermal conductivities of all materials, the coupling of the substrate to the bath, and the coupling between the materials are varied and their influence on the result of the simulation is shown. As can bee seen from the figures, the result depends mainly on two factors: The position of the heating spot and the ratio of the thermal conductivities of the polyimide and the gold. In the experiment, the position of the laser can be set to the required accuracy, thus only the ratio of the thermal conductivities remains. The uncertainties are estimated high enough to accommodate roughly 20 % variation. Physically, the fact that

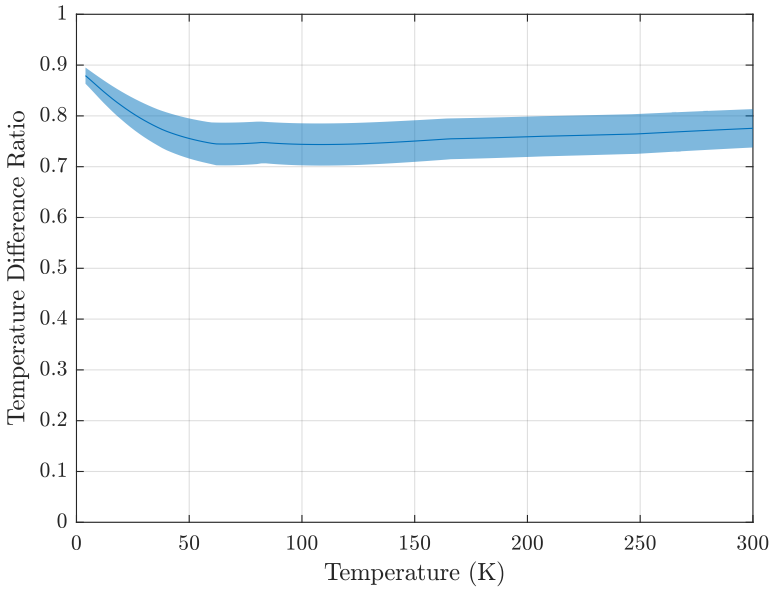



**Figure 7.1.:** The graph shows the simulated ratio of the temperature difference across the constriction to the temperature difference measured at the thermometer position. The area around the curve shows the assumed uncertainty. The results are taken from the simulations VC_04 and VC_05.

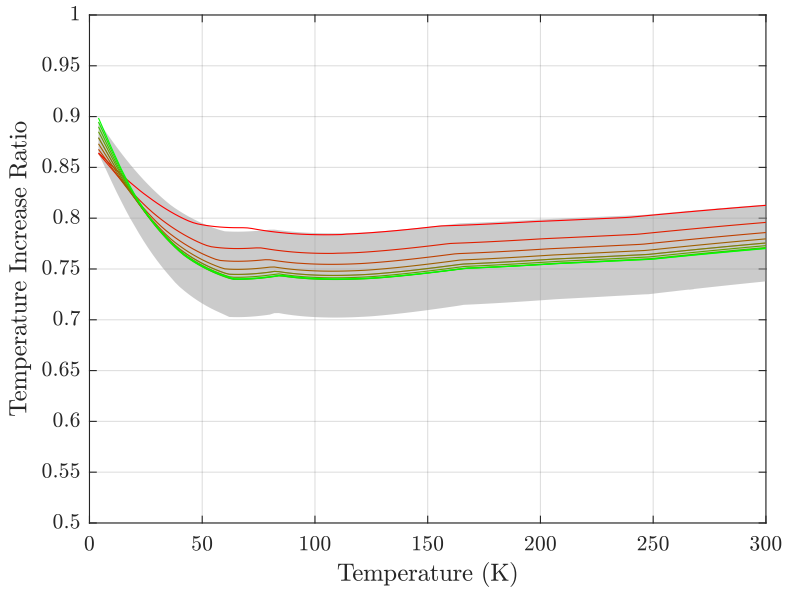



**Figure 7.2.:** The position of the laser heating spot is varied from a distance to the constriction of 30 µm (red) to 70 µm (green) in steps of 5 µm. The values with their uncertainties used for the analysis are shown in grey as a reference.

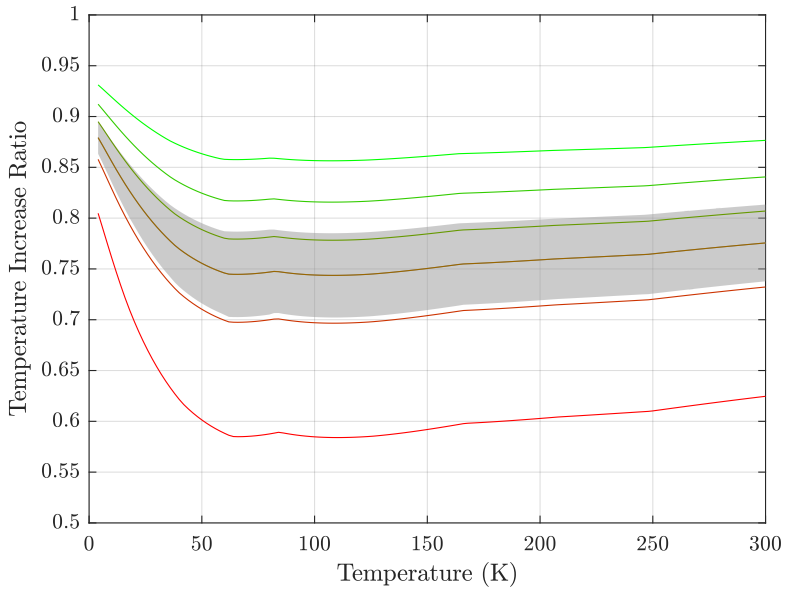



**Figure 7.3.:** The thermal conductivity of gold is changed by a factor of 0.5(red), 0.8, 1, 1.2, 1.5, and 2 (green). The values with their uncertainties used for the analysis are shown in grey as a reference.

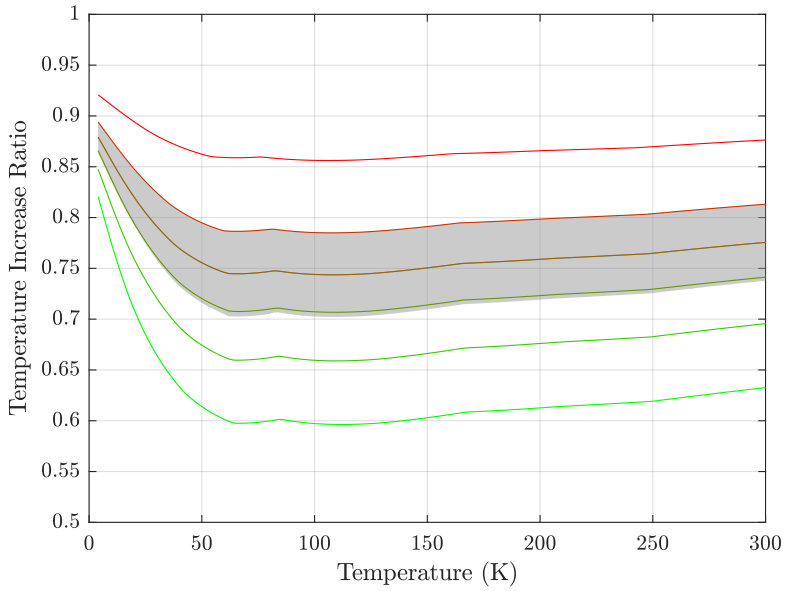



**Figure 7.4.:** The thermal conductivity of the polyimide is changed by a factor of 0.5(red), 0.8, 1, 1.2, 1.5, and 2 (green). The values with their uncertainties used for the analysis are shown in grey as a reference.

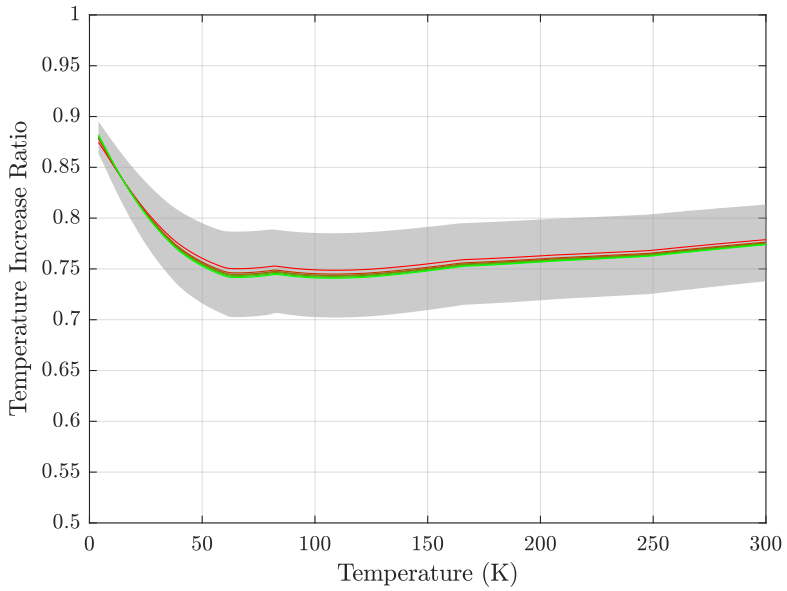



**Figure 7.5.:** The thermal conductivity of bronze is changed by a factor of 0.5(red), 0.8, 1, 1.2, 1.5, and 2 (green). The values with their uncertainties used for the analysis are shown in grey as a reference.

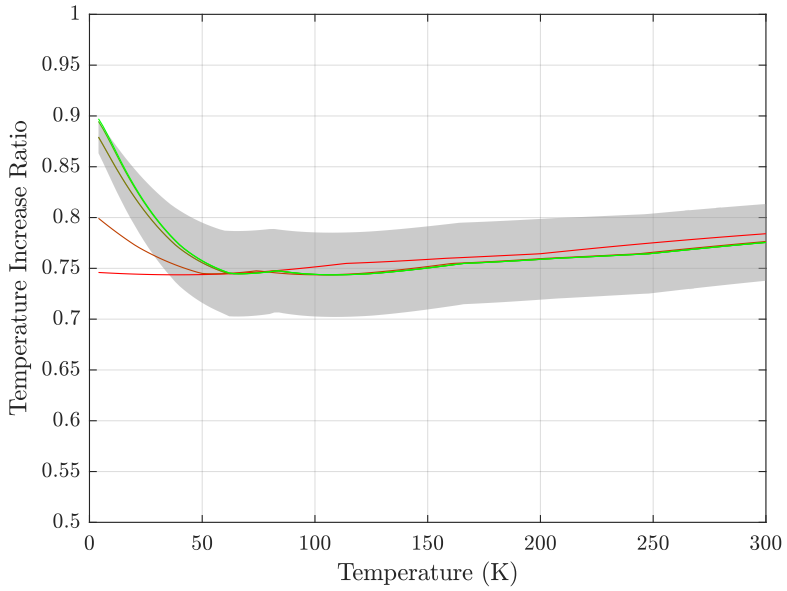



**Figure 7.6.:** The coupling to the thermal bath is set to $1 \cdot 10^2$ (red), $1 \cdot 10^3$, $1 \cdot 10^4$, $1 \cdot 10^5$, $1 \cdot 10^6$ $\mathrm{W\,m^{-2}\,K^{-1}}$ (green). The values with their uncertainties used for the analysis are shown in grey as a reference.

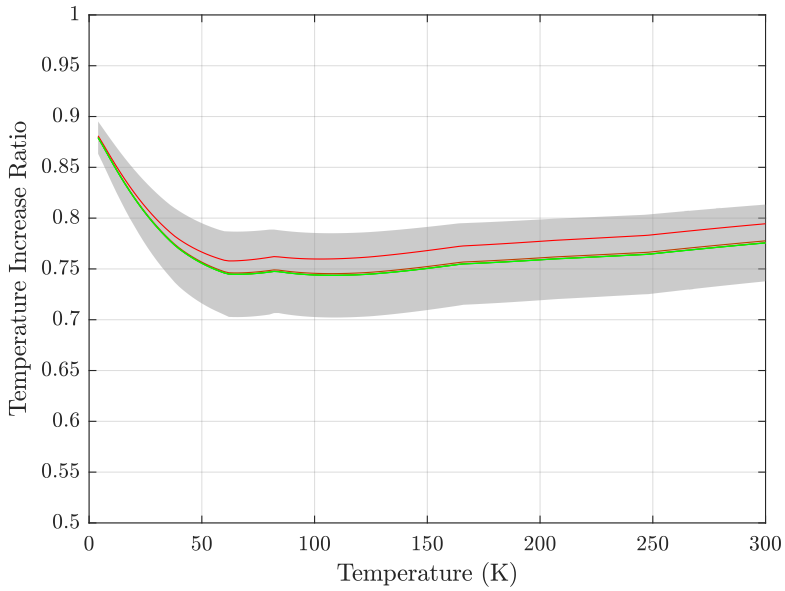



**Figure 7.7.:** The coupling between gold and polyimide is set to $1 \cdot 10^6$ (red), $1 \cdot 10^7$, $1 \cdot 10^8$, $1 \cdot 10^9$, $1 \cdot 10^{10}\,\mathrm{W\,m^{-2}\,K^{-1}}$ (green). The values with their uncertainties used for the analysis are shown in grey as a reference.

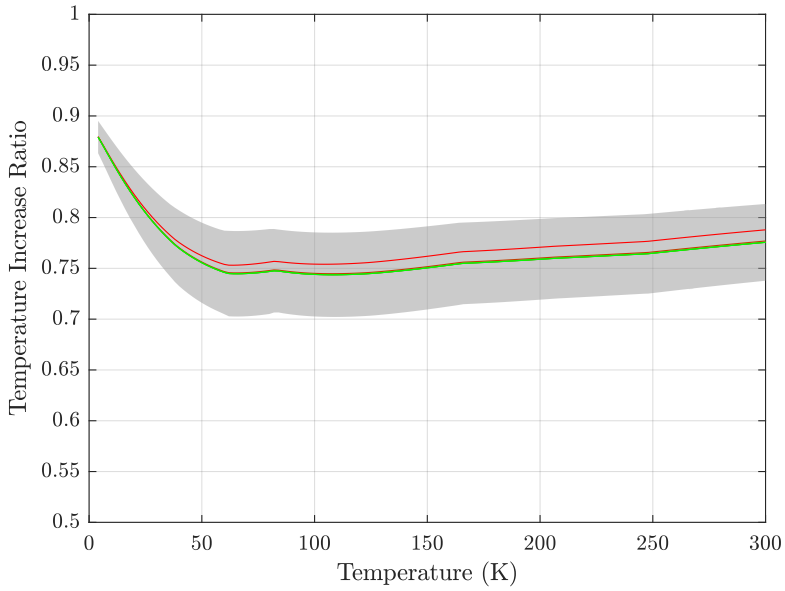



**Figure 7.8.:** The coupling between polyimide and bronze is set to $1 \cdot 10^6$ (red), $1 \cdot 10^7$, $1 \cdot 10^8$, $1 \cdot 10^9$, $1 \cdot 10^{10}\,\mathrm{W\,m^{-2}\,K^{-1}}$ (green). The values with their uncertainties used for the analysis are shown in grey as a reference.

the ratio is the main error source makes sense, since this directly influences the shape of the heat distribution by the length at which the heat dissipates into the substrate.

## 7.3. Analysis

Following the log file of the measurement, all the data for every trace were collected. This data were the number of the picture at the beginning and the end, the resistance of the thermometer at the beginning and the end, the state of the laser (on or off), which side the laser was heating, the temperature of the sample chamber during the measurement, and the data from the transient recorder. With the known time of the beginning and the end of the measurement, the temperature log of the cryostat was filtered to get only the temperatures during the measurement. The data file from the transient recorder was loaded and for every channel 10 ms of data were averaged to a single new data point. This filters out the lock-in modulation from the voltage and the current signal and also reduces the number of data points. The voltage and current signals were stored directly, the four lock-in signals were shifted to correct the drift of the system and the conductance was calculated from the ratio of the current and the voltage amplitude. The data from the lock-in amplifier stored in later versions of the measurement could be used directly, just the conductance needed to be calculated.
To enable a quick analysis, the data were further averaged in three stages, with each stage averaging over ten data points of the previous result. The original data and the three averaged versions were saved in a `.mat`-file for further analysis.
For the further analysis this file was loaded, and the desired data reduction was chosen. If the data contained multiple entries for the same opening or closing process due to interruption, these traces were combined. At this stage it is possible to filter out problematic traces. Then the sample chamber temperatures of all traces were combined and averaged. This gave the base temperature for the complete measurement.
For each trace the opening and closing thermometer resistances were collected as shown in figure 7.9, and their differences were calculated. The average of the differences was then used to subtract the influence of the bending from the open resistances. Then the corrected open and closed resistances were averaged for each trace. The values without the laser heating were subtracted from those with heating and averaged over the complete measurement to get the heating induced resistance change in the thermometer.
During the cooldown, the resistance of the thermometer as a function of the chamber temperature was measured with and without laser heating on both sides of the constriction. This data was now used to convert the obtained resistances into temperatures. Since this function is strongly nonlinear, this was performed in the following way: The average sample chamber temperature was used to calculate the corresponding resistance. Then the resistance change due to the heating was added and the resulting resistance was converted back into a temperature. This temperature of the thermometer was calculated for the heating on both sides.
The data from the simulations were now used to estimate the temperatures at the constriction. Here the heating on the side without the thermometer had to be treated differently, since the effects of a different heating position are not measured by the thermometer and have to be deduced from the photos and simulations.
Since the temperature difference across the junction was now known, the literature values for the bulk thermopower of gold [Cri70] could be used to calculate the bulk thermovoltage.
To correct the effects of the voltage offset and offset bias currents of the amplifiers, they

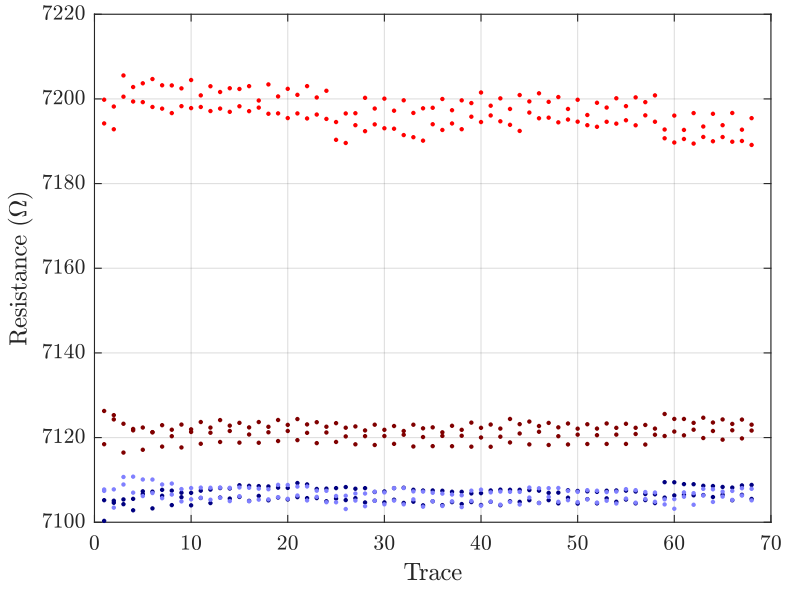



**Figure 7.9.:** The resistance of the thermometer depends on temperature, but also on the bending of the sample. The bright red resistances are measured with the laser heating the side of the constriction where the thermometer is, the dark red are measured with the heating on the other side. The bright blue resistances are measured with the laser on the thermometer side, but turned off, and the dark blue with the laser on the other side, but also turned off. Every trace has a pair of points, since the resistance is measured in the open and the closed state.
The data show that the influence of the bending is not negligible but small compared to the influence of temperature change due to the laser heating.

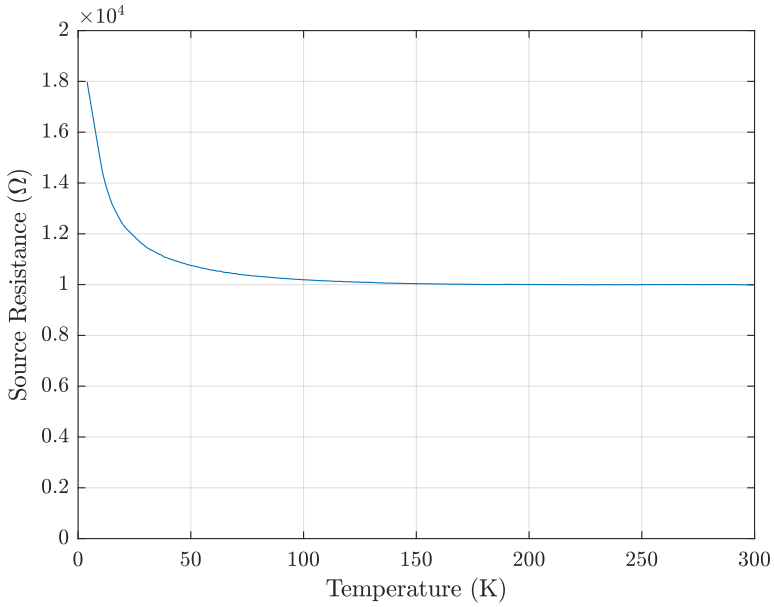



**Figure 7.10.:** This figure shows the temperature dependent value of the source resistance. This resistance is placed in series with the sample to limit the current when the sample is closed. To reduce the noise, it is mounted inside the sample chamber. Thus the temperature dependence has to be included in the data analysis.

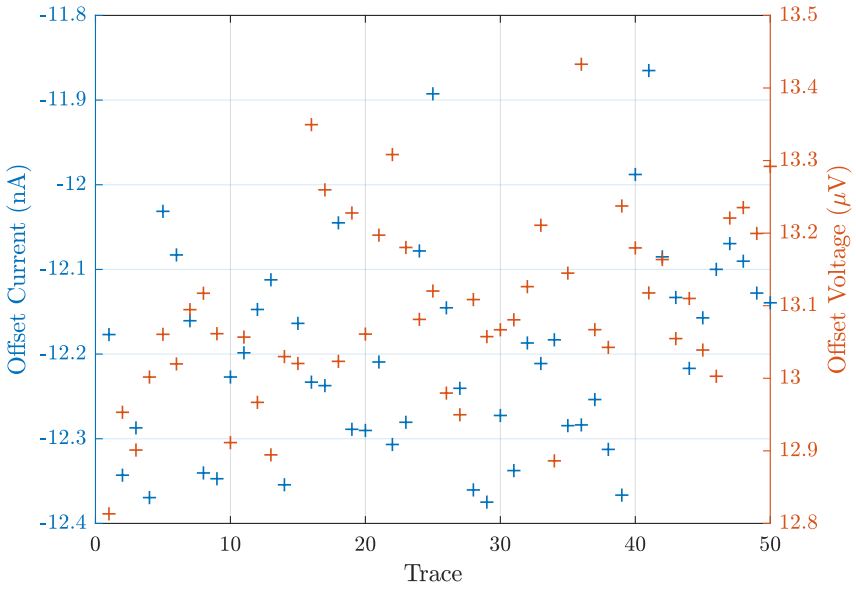
were described by the following classically derived formulae:

$$U_{\text{offset}}\left(I_{\text{offset current}}, U_{\text{offset voltage}}, G\right) = \frac{I_{\text{offset current}}}{\frac{1}{G+\frac{1}{R_{\text{source}}}} + R_{\text{input}}} + U_{\text{offset voltage}} \tag{7.1}$$

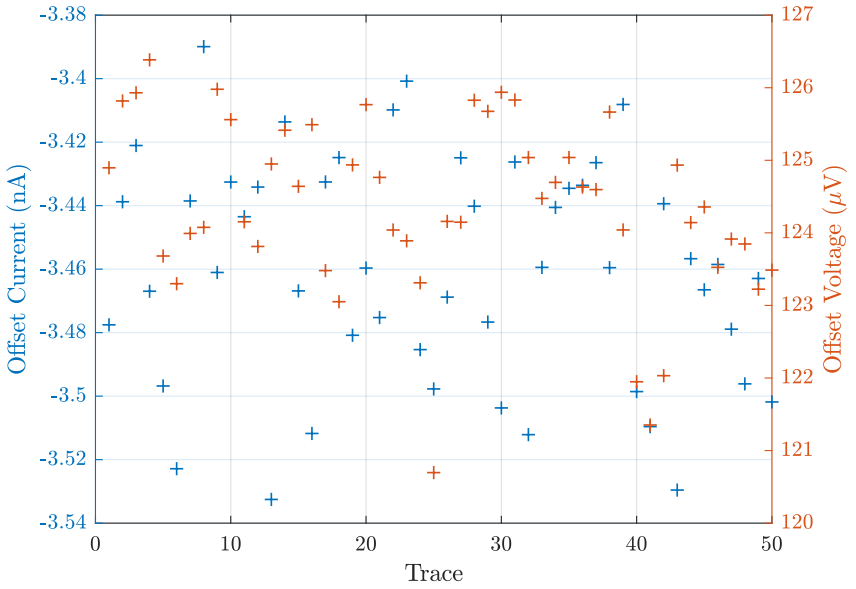
$$I_{\text{offset}}\left(I_{\text{offset current}}, U_{\text{offset voltage}}, G\right) = U_{\text{offset voltage}}\left(\frac{1}{R_{\text{source}}+\frac{1}{G}} + \frac{1}{A}\right) + I_{\text{offset current}} \tag{7.2}$$

with the conductance $G$, $R_{\text{input}} = 100\,\Omega$ as input impedance of the $IV$-converter, $A = 1 \cdot 10^6\,\text{V}\,\text{A}^{-1}$ as the transimpedance of the converter, and $R_{\text{source}}$ as the combined impedance of the bias source and the series resistor. $R_{\text{source}}$ is temperature dependent as shown in figure 7.10.
The offset bias current and the voltage offset are unknown for both amplifiers, but they can be derived from the data by fitting these functions to the average current and voltage measured without the laser heating. The typical results are shown in figure 7.11. These results are used to verify that no steady trend of the offsets is visible. This is important since such a trend would erroneously appear as a thermovoltage.
These functions generated for each trace without the heating were now used to correct the data with the heating by the following formulae:

$$I_{\text{corrected}} = I_{\text{measured}} - I_{\text{offset}}\left(G_{\text{measured}}, I_{\text{offset current}}, U_{\text{offset voltage}}\right) \tag{7.3}$$




**(a)** offsets of the voltage amplifier




**(b)** offsets of the IV-converter

**Figure 7.11.:** The graphs shows a typical behaviour of the offset voltage and offset bias current of the voltage amplifier and the IV-converter. The data is taken from measurement 20191128170541 at 86 K. Note that the offset can be shifted with the adjustment screws, therefore the static values are not useful. The graph is only used to verify that the offsets do not show a steady trend which would influence the data analysis.

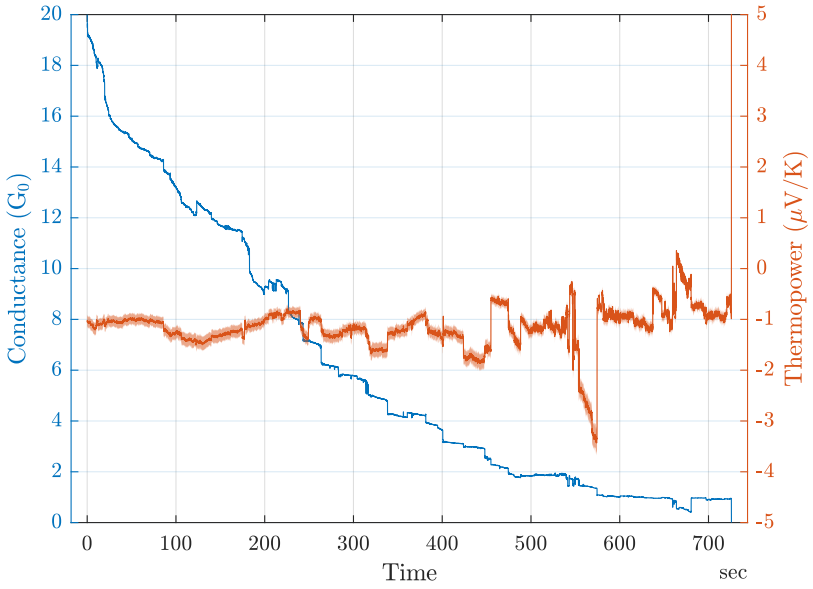



**Figure 7.12.:** The graph shows the conductance and the thermopower as function of time for a typical opening trace. It can be seen that when the conductance jumps because the contact rearranges, this also generates a sudden change in the thermopower. But since the thermopower is more sensitive to the exact atomic configuration, whereas the conductance is mainly defined by the orbitals at the tip, there are regions, e.g. at $1\,G_0$, where the conductance shows no change, the thermopower however shows jumps and gradients indicating the effects of the stretching and restructuring of the leads close to but not at the constriction.
The data is taken from trace 19 of the measurement 20200308121210 on sample T0143S with $T_{\mathrm{Hot}} = 76.5(3)\,\mathrm{K}$ and $T_{\mathrm{Cold}} = 70.83(2)\,\mathrm{K}$.

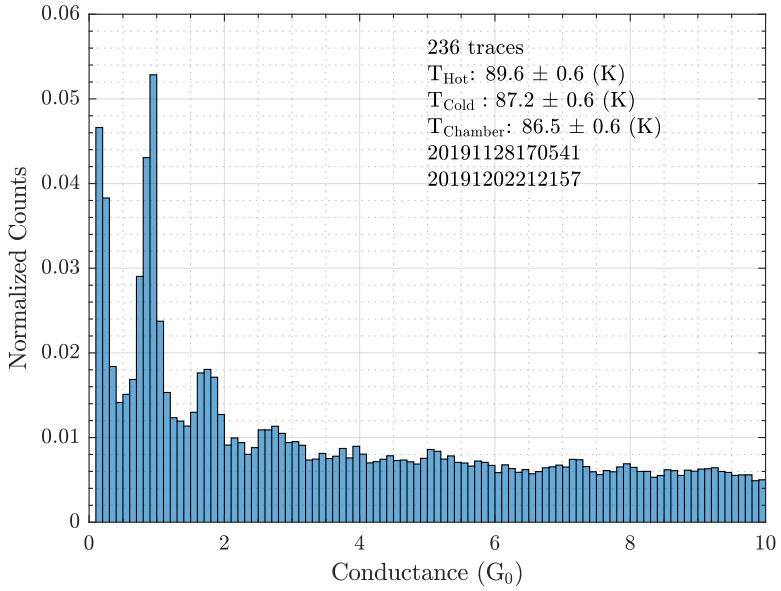
$$U_{\mathrm{corrected}} = U_{\mathrm{measured}} - U_{\mathrm{offset}}\left(G_{\mathrm{measured}}, I_{\mathrm{offset\ current}}, U_{\mathrm{offset\ voltage}}\right) + \frac{I_{\mathrm{corrected}}}{G_{\mathrm{measured}}} \tag{7.4}$$

$I_{\mathrm{offset\ current}}$ and $U_{\mathrm{offset\ voltage}}$ are different for the two amplifiers and for each trace.
Now the bulk thermovoltage could be subtracted, and the thermopower of the junction could be calculated with the known temperature difference for every point in every trace. To check for errors, the traces without heating were also divided by the temperature difference, to have the same scaling. The bulk thermovoltage was subtracted from these values as well, in order to show where the "no signal" position would be. Figure 7.12 shows the thermopower and the conductance of a single trace as function of the time.

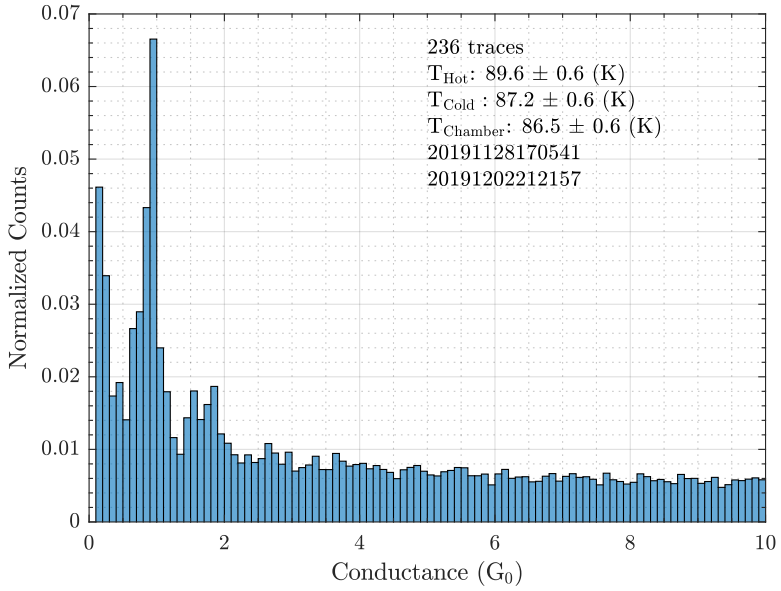
## 7.4. Results

Having calculated the thermopower and the conductance for every trace, the data can be analyzed in a statistical manner: The conductances were counted and collected into

histograms. Two exemplary histograms are shown in figure 7.13 where one diagram only includes the data with the laser heating the constriction on either side and the other one includes only the data without the laser heating.
The thermopower was also collected as a function of conductance and colored distribution maps were generated. An example is shown in figure 7.14. To verify that the result does not depend on the perfect subtraction of the offsets, the data was also analyzed with the correction not applied to the dataset to which it was fitted, but to the next dataset. As can be seen in figure 7.15, the noise increases since the correction is not as good, but the difference between the signals without heating and with heating are still clearly visible and similar to the results with the correct offset correction.
Since the thermometer is only on one side of the sample, the data with the laser heating the other side has no accurate temperature difference. Thus this data was disregarded in the following.
The thermopower is averaged over small conductance ranges, generating an average thermopower and the standard deviation. The uncertainties due to temperature differences are also averaged since they are not statistically independent. The result with the two types of uncertainties is shown in figure 7.16.
The thermopower values with the measurement uncertainties are averaged in certain ranges, and by combining the data from measurements at different temperatures, the figures 7.17 to 7.19 are generated, where the thermopower is shown as function of the average temperature at the junction.
The sample T0115S shows the same behaviour as the later measurements on sample T0143S and T0144S, but the thermopower is about $0.5\,\mu V\,K^{-1}$ higher. There are different reasons why this dataset may be different: The sample was measured with the transient recorder for the data acquisition instead of the lock-in amplifier used for the later samples. The sample was broken in air before the measurement which might have contaminated the constriction area. The sample showed a big difference in the resulting thermopower when comparing the heating on the different sides of the constriction. All these factors make the data less reliable than the data measured on samples T0143S and T0144S, where the contamination can be ruled out. The sample T0114 shows strong scattering of the values at $1\,G_0$. This scattering is due to the contact not being trained. The training forms the contacts and provides more reproducible atomic structures.
The results show that at low temperatures, the thermopower diminishes, which is in agreement with the theory [Pau11] and previous measurements [Lud99]. At room temperature my measurements show a value of $S = -1.1\,\mu V\,K^{-1}$. This does not exactly agree with the results of [Eva15] who found a value of $S = -0.75\,\mu V\,K^{-1}$. However, they measure at temperature differences of 20 and 40 K, whereas my measurements have a temperature difference of only 4 K. Therefore, my measurements were performed at an average temperature of 297 K whereas their measurements were at average temperatures in the range from 305 to 315 K. Given the rising trend in my data, their value agrees with an estimated extrapolation. Hence, my measurements do not contradict their results.
More remarkable is the unexpected behavior between these two ends: The thermopower drops to about $-2\,\mu V\,K^{-1}$ at a temperature of 150 to 250 K. This minimum is remarkable since it deviates from the linear dependence on the absolute temperature predicted by the Mott equation (equation (2.9)) if the transmission were featureless. Such a non-monotonous behavior is known in organometallic complex wires, where the transmission




**(a)** data with laser heating




**(b)** data without laser heating

**Figure 7.13.:** The two conductance histograms recorded on sample T0115S with and without laser heating show the characteristic $1\,G_0$ peak of gold. Additional peaks can be seen at $1.8\,G_0$ and around $3.7\,G_0$.

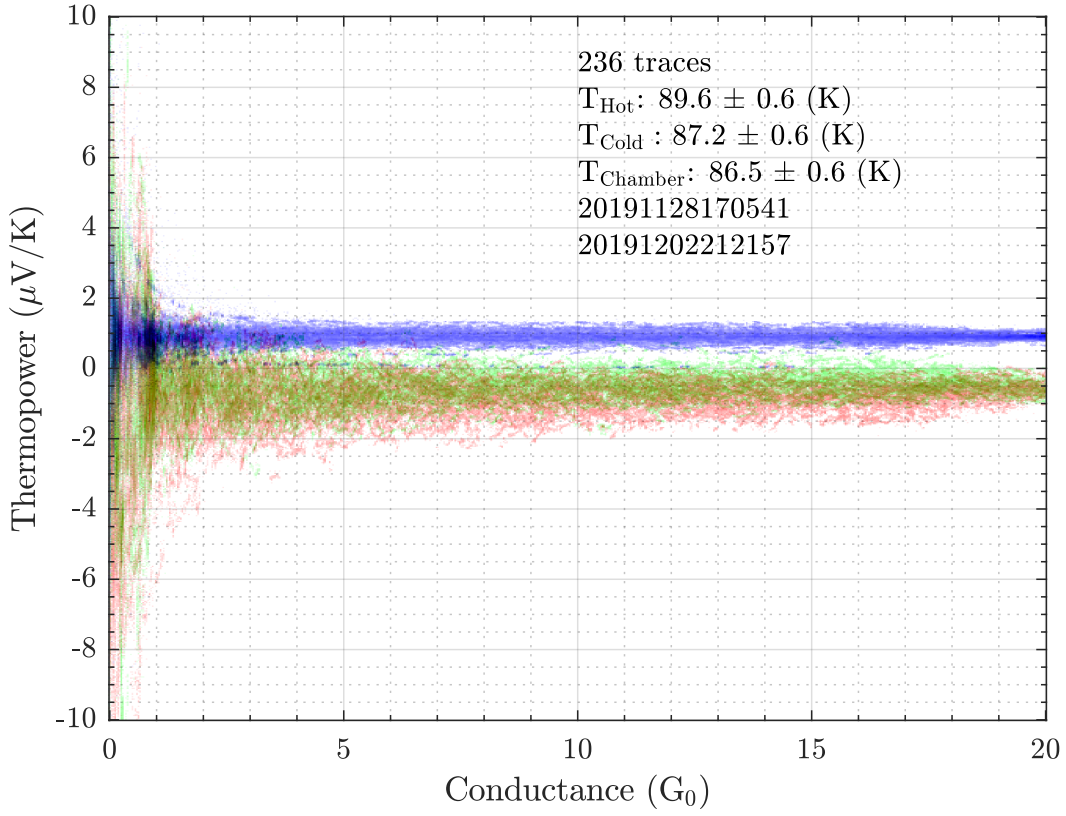



**Figure 7.14.:** This color plot shows the distribution of the thermopower and the corresponding conductance. The color intensity scales logarithmically with the number of data points.
The blue color distribution is derived from the data without laser heating. It is processed similar to the data with the heating: It is divided by the temperature difference obtained by the heating, and the bulk thermopower is subtracted.
The red coloring shows the thermopower calculated from the traces with the laser heating on the side of the constriction where the thermometer is. There, the temperature difference is reliable.
The green coloring shows the thermopower calculated from the traces where the laser heating was on the other side. Due to different distances to the constriction, the lack of the additional structures for the thermometry and the fact that the laser focus is optimized for the measurement on the thermometer side, the assumption of the analysis, that the temperature difference would be the same, is rather speculative. Ideally, if everything were symmetric, the green and the red coloring would coincide.

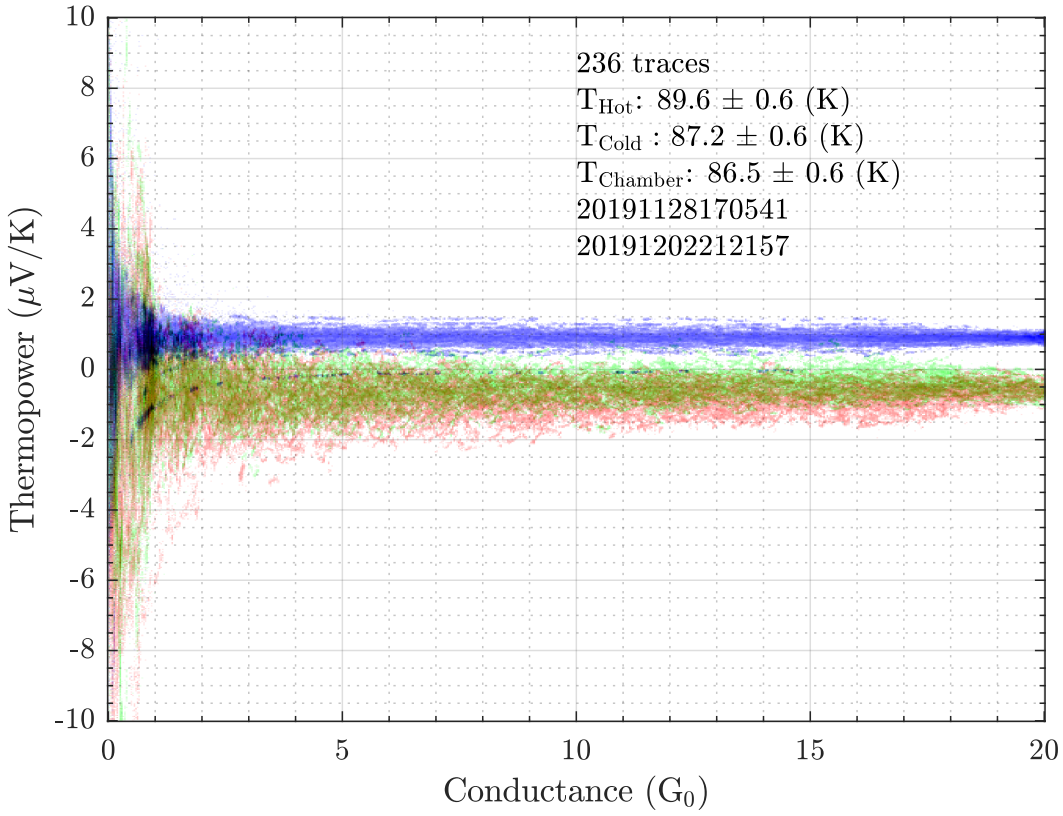



**Figure 7.15.:** This is similar to figure 7.14. However, the offset correction derived from the data without the heating are not applied to the dataset they were generated from, but to the next dataset. This allows to verify that the results do not depend on the exact correction, but can be reproduced with slight variations. The non perfect correction is the reason for the increased noise visible in the data.

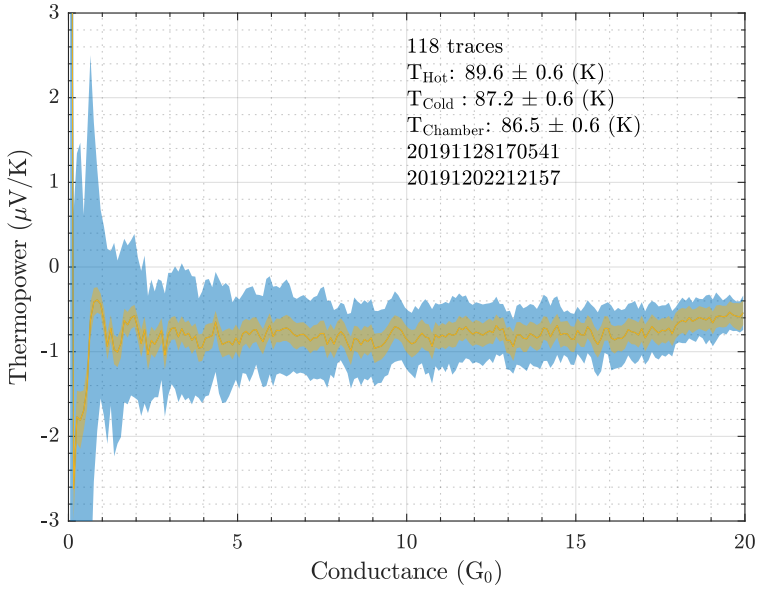



**Figure 7.16.:** This graph shows the averaged thermopower as function of the conductance. The blue area indicates the standard deviation of the data and the yellow area indicates the uncertainties arising from the temperature difference.

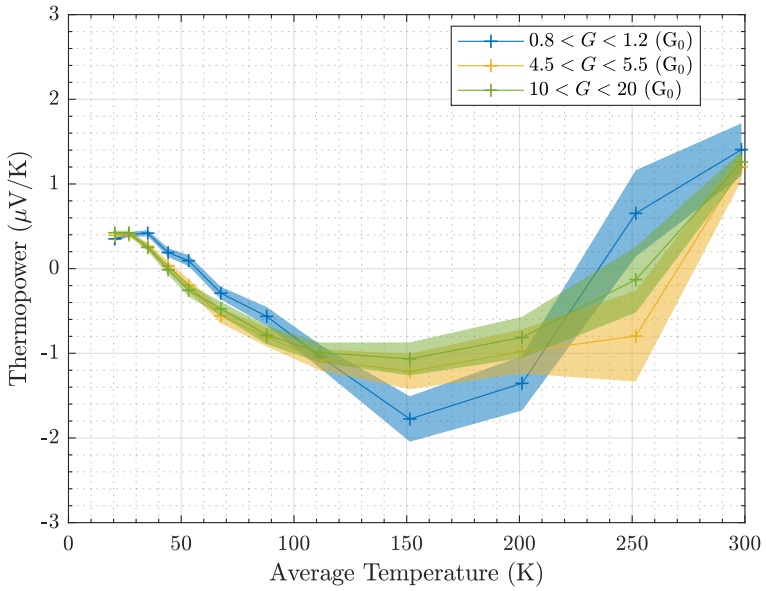



**Figure 7.17.:** The thermopower from measurements at different temperatures is averaged in three different conductance regions. These values with the uncertainty estimates based on the temperature errors are shown as function of the average temperature of the junction.
The data was measured on sample T0115S which was broken in air before the measurement.

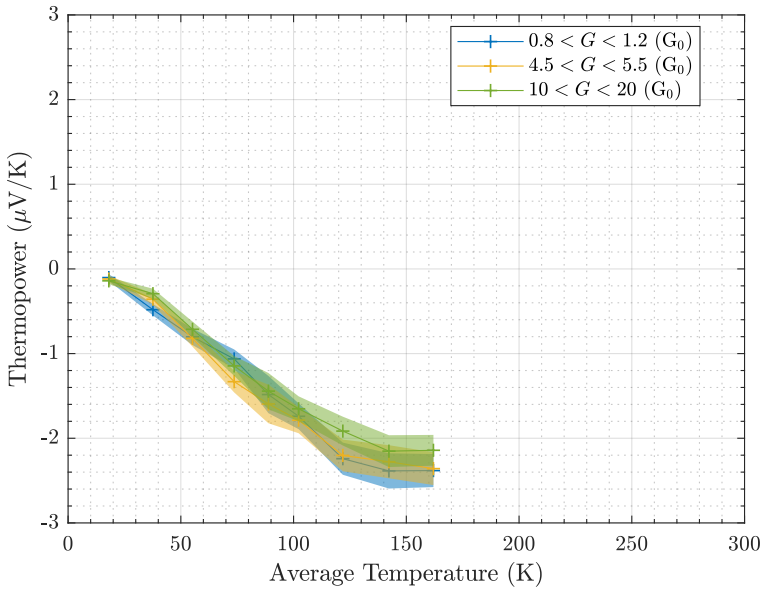



**Figure 7.18.:** This graph is generated similar to figure 7.17.
The data was measured on sample T0143S which was broken in vacuum at ambient temperatures before the measurement.

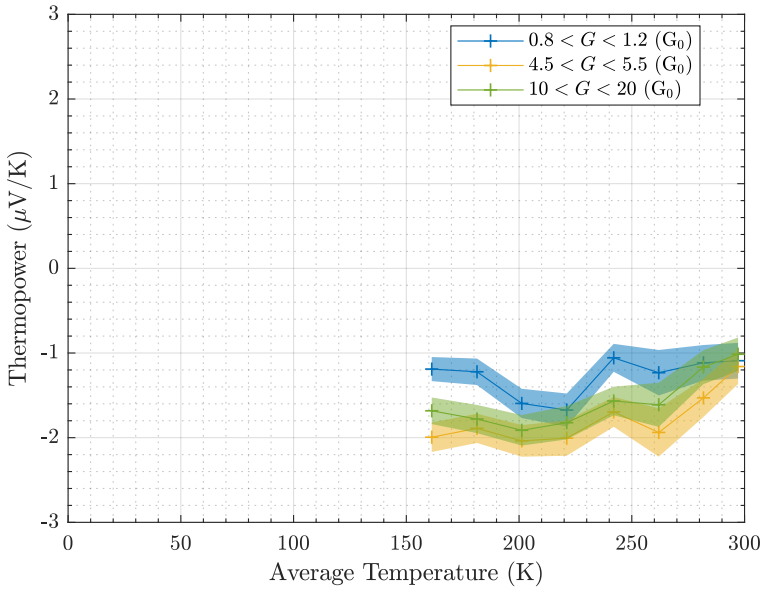



**Figure 7.19.:** This graph is generated similar to figure 7.17.
The data was measured on sample T0144S which was broken in vacuum at ambient temperatures before the measurement.

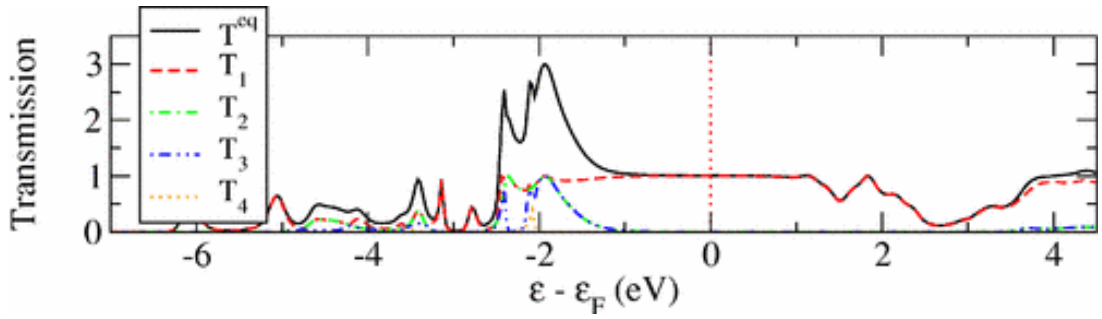



**Figure 7.20.:** The energy dependence of the transmission is shown and broken down in the four most influential channels [Vil07, Fig. 4]. The data shows the flat and featureless transmission around the Fermi energy being dominated by the singe channel of the s-orbital.

can have many peaks around the Fermi energy due to resonant states [Nak13]. Theoretical calculations for the transmission in gold contacts, however, show a very flat and featureless transmission around the Fermi energy as shown in figure 7.20. Therefore, no interesting behavior would be expected for the thermopower of atomic gold contacts. The measured data shows a minimum, hence, these simple models are not sufficient to describe the measured thermopower. Possible explanations might be magnetic impurities in the gold or phonon drag. The effects of magnetic impurities are only seen at low temperatures and not in the range of 150 to 250 K, therefore, their effects can be excluded. Phonon drag is a known contribution to the bulk thermopower, but in ballistic contacts, only the electron occupations and the transmission are incorporated in the simple theoretical models. More complex theoretical predictions for the temperature dependence of the thermopower of a gold atomic contact could not be found.
It can be excluded that the minimum is an artifact of the measurement: The thermopower is calculated from the temperature difference and the measured voltage. If the minimum were an artifact of an incorrect temperature difference, the temperature difference would need to be at at least 50 % higher than what I assume in the analysis. This would mean that the temperature of the thermometer would need to be lower than the temperature of the constriction. This is impossible because the thermometer is located between the heating spot and the constriction.
The voltage measurement is verified not only by the quantitative agreement at the temperature extremes. Due to the design of the measurement the voltage cannot be influenced by unaccounted additional thermovoltages: During the measurement, data is recorded with and without heating. The data without the heating is used to subtract any offsets which also eliminates any static thermovoltages of the setup. Further, the sample is heated on both sides during the measurement. The agreement of the thermopower measurement on both sides rules out any unaccounted thermovoltages which might occur when heating.
Similar to the temperature dependence of the thermopower, the standard deviation of the thermopower also shows nonlinear characteristics. This temperature dependence of the standard deviation is shown in figures 7.21 to 7.23. Except for the one data point measured at room temperature on sample T0115S at 1 $G_0$, the deviation of the thermopower of all samples show quantitatively similar behavior: The deviation increases from the low

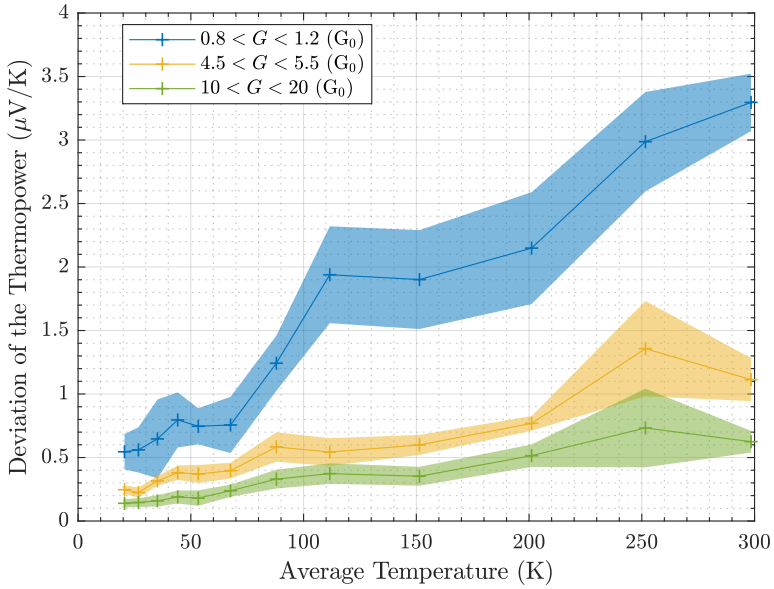



**Figure 7.21.:** The standard deviation of the thermopower from measurements at different temperatures is averaged in three different conductance regions. These values with the uncertainty estimates based on the deviation of the standard deviation are shown as function of the average temperature of the junction. The data was measured on sample T0115S which was broken in air before the measurement.

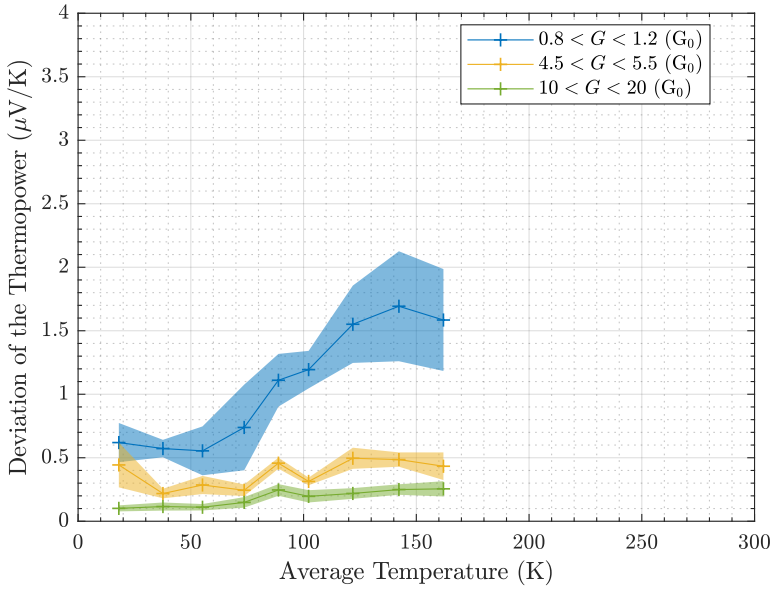



**Figure 7.22.:** This graph is generated similar to figure 7.21.
The data was measured on sample T0143S which was broken in vacuum at ambient temperatures before the measurement.

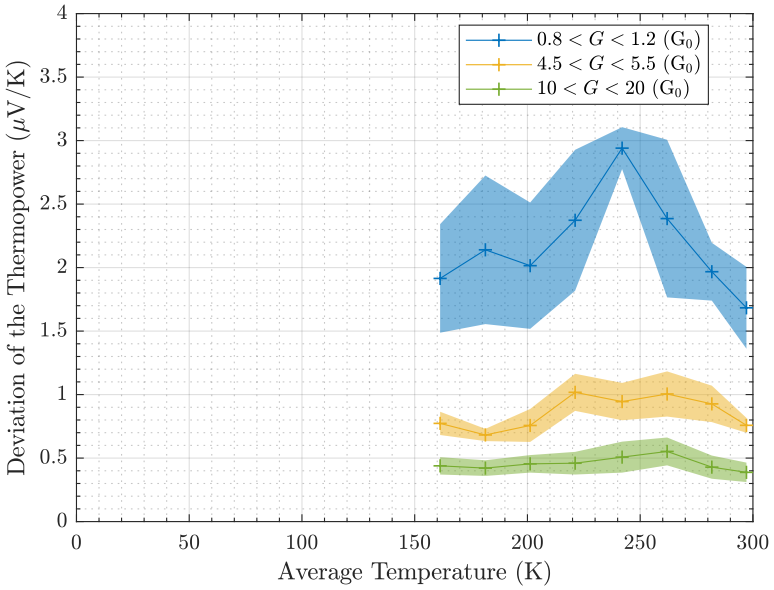



**Figure 7.23.:** This graph is generated similar to figure 7.21.
The data was measured on sample T0144S which was broken in vacuum at ambient temperatures before the measurement.

temperatures up to 240 K and decreases again towards room temperature. This trend is observable in all evaluated conductance ranges.
The standard deviation was verified to be not influenced by the temperature dependence of the temperature difference. It is also not influenced by the number of traces measured at each given temperature.
The reduction of the standard deviation at temperatures above 240 K might be linked to the minimum of the thermopower below that temperature since the absolute value of the thermopower also starts to decrease at roughly this temperature. Similar to the thermopower, no theoretical explanation for the temperature dependence of the standard deviation of the thermopower could be found.
The deviation from the simple theoretical model imply a temperature dependence of the transmission. To explain this temperature dependence of the transmission, complete theoretical simulations need to be performed using temperature dependent DFT (density-functional theory) calculation coupled with NEGF (non-equilibrium Green's function) theory calculations.

# 8. Molecule Measurements

For measuring the thermovoltage generated by a single molecule, different electronics are needed, because the resistance of a molecule is orders of magnitude higher than the resistance of a metallic contact.
The resistances are so high that a direct measurement of the voltage across the molecule is not possible with the amplifiers used before. Therefore, the new approach is to rely solely on the change of the current through the molecule due to the thermovoltage. If one neglects that the thermopower might be temperature dependent, one might expect to have a molecular contact with about $10\,\mu V\,K^{-1}$. For a typical temperature difference of 3 K, this results in a thermovoltage of 30 µV. To measure the voltage with reasonable accuracy, the setup needs to resolve voltages of 6 µV. At a molecular conductance of about $1 \cdot 10^{-4}\,G_0$, this results in a current of 46 fA. This current is pretty small, thus an IV-converter with a transimpedance of 1 GΩ is used. With a cooled feedback resistor, the theoretical noise of an ideal amplifier is $2.2\,fA/\sqrt{Hz}$. The feedback resistance was also chosen to be this large, since it reduces the influence which the offset voltage of the amplifier has on the measured current. To keep the influence of leakage currents, interference and parasitic capacities low, an IV-converter with a discrete JFet front-end was built (see appendix C.6). This also reduces the noise of the amplifier, since it allows to mount the feedback resistor in the cold and this is the dominant noise source.
With all the electronics for measuring the small currents it is not possible to implement the additional temperature measurement. There are several reasons for that: The IV-converter takes up all the measurement wiring installed in the cryogenic insert. The molecules in the samples are not as reliable as clean gold contacts, therefore the effort to produce many samples with the five steps needed for the thermometry is excessive. The current applied during the measurement of the resistance thermometer may upset the current measurements for the thermovoltage. Therefore, the temperature was not directly measured on the molecule samples. Instead, the value for the temperature difference was taken from the gold contacts. The temperature of the thermometer was measured at different laser powers at different spots on the sample with the sample chamber at 86 K. This was used to generate a map which is shown in figure 8.1. This map shows the sensitivity to the laser heating as function of the position of the laser focus.
Analyzing this map at the positions used for the molecule measurements, and multiplying with the used laser power and the correction factor from the simulations gives a generated temperature difference of 3.3(10) K. Since this temperature is just an estimate rather large uncertainties have to be assumed.
The modulation of the bias voltage and the resulting modulation of the current was measured. This allows to calculate the conductance of the constriction with the known input impedance of the IV-converter. The dc part of the current is the result of the thermovoltage. Thus, with the known conductance the thermovoltage can be calculated

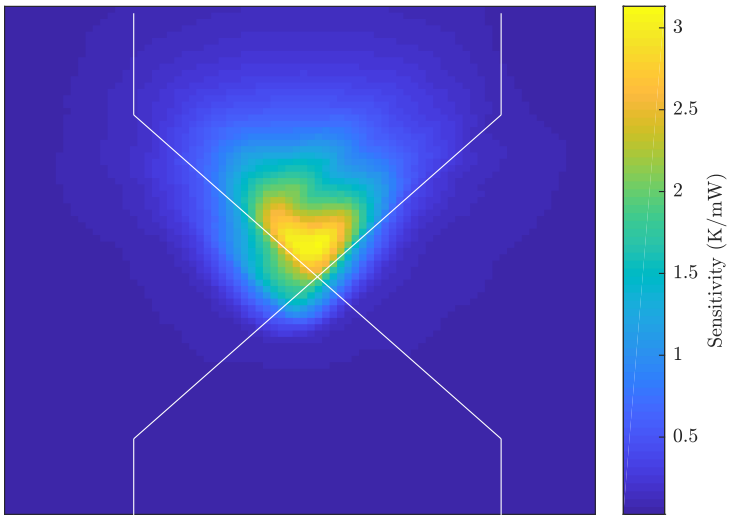



**Figure 8.1.:** This map shows the sensitivity to the laser heating as function of the position of the laser focus: The measured temperature at the thermometer is sensitive to the position of the heating spot. Knowing the laser power, this measurement allows to calculate the temperature for the similar sample layouts of the molecule samples without the direct measurement.
The structure of the sample is indicated with white lines.

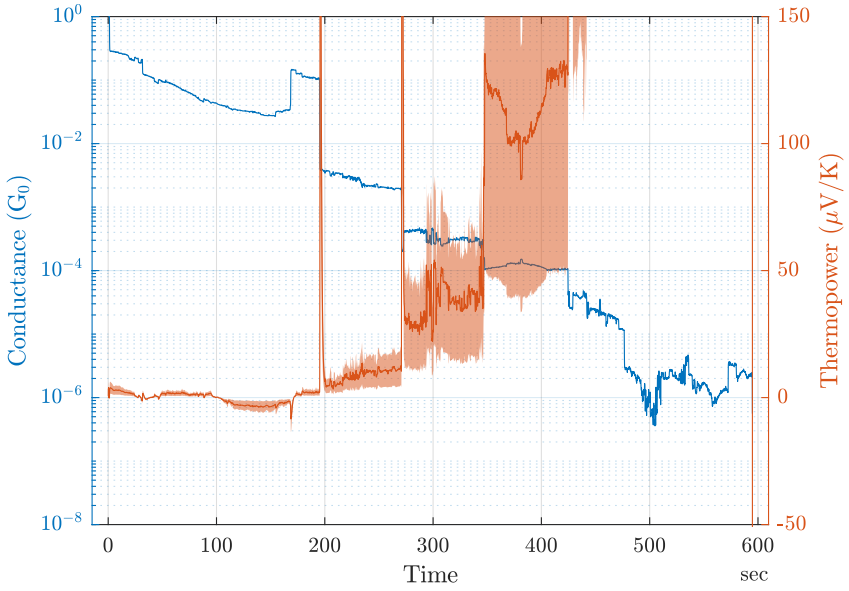



**Figure 8.2.:** The conductance and the thermopower were measured simultaneously during the opening of the contact. The measurement shows data for terphenyldithiol from trace 3 of measurement 20200218121738 on sample T0007C. The measurement was performed at 86 K and a temperature difference of 4(2) K was assumed.

from the dc current according to the formula

$$U_{\mathrm{Th}} = I\left(\frac{1}{G} + R_{\mathrm{setup}}\right)$$

with $R_{\mathrm{setup}}$ as the lumped resistance of the setup. This resistance includes the resistive divider, the impedance of the IV-converter, and the series resistance to limit the current. The latter is the dominant term.
Having calculated the thermovoltage, the thermopower is easily derived. A typical opening trace is shown in figure 8.2 with the conductance and the thermopower. The data of a single trace is not very useful, therefore a more statistical approach is performed with all the data of the opening traces. The conductance is shown as a conventional histogram in figure 8.3 and since the molecule is not visible above the noise, a two dimensional histogram is created showing the conductance and the time directly proportional to the displacement. The end of the $1\,G_0$ plateau is taken to be at $t = 0$ to shift all traces to the same starting position. The result is shown in figure 8.4.
This histogram shows some indication of molecular contacts, but the data measured at 86 K is far from clear. This is due to the small number of traces combined in the histogram. Unfortunately no additional traces could be measured, since the constriction degraded to the point where it could not be closed any more.
The thermopower was also analyzed statistically and colored histograms were created, similar to those generated from the data of the gold contacts. Due to the temperature variations induced by the electrical interference generated by the motor motion, the volt-

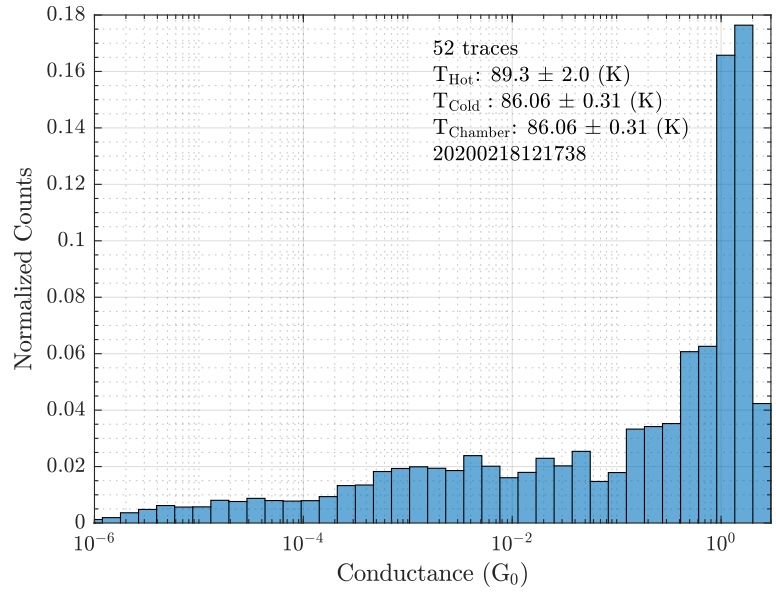



**Figure 8.3.:** The histogram generated from the opening traces without laser heating of sample T0007C with terphenyldithiol.

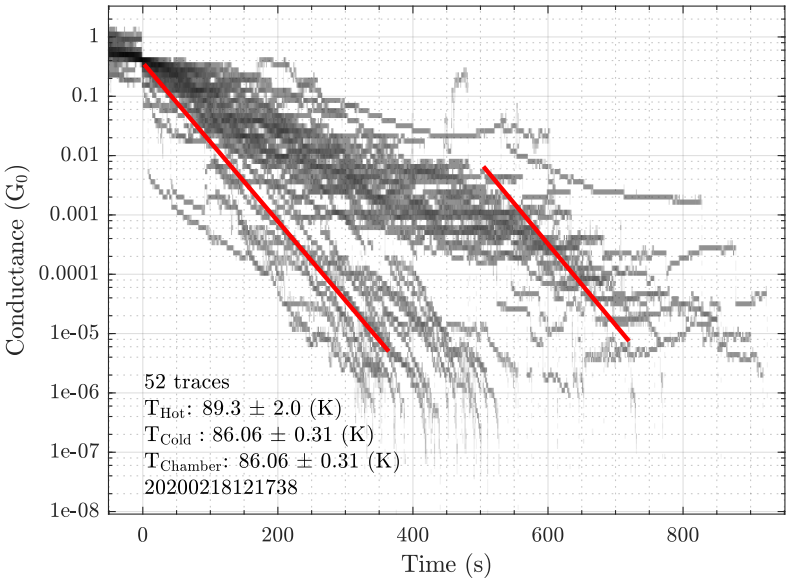



**Figure 8.4.:** The 2D histogram generated from the opening traces without laser heating of the sample T0007C with terphenyldithiol. The histogram shows two tunneling regimes, the direct tunneling and the tunneling after the molecular contact is broken, thereby indicating that molecular contacts were formed. The two red lines serve only as a guide to the eye showing these two tunneling regimes. All traces are shifted horizontally to have the end of the $1\,G_0$ plateau at the same time.

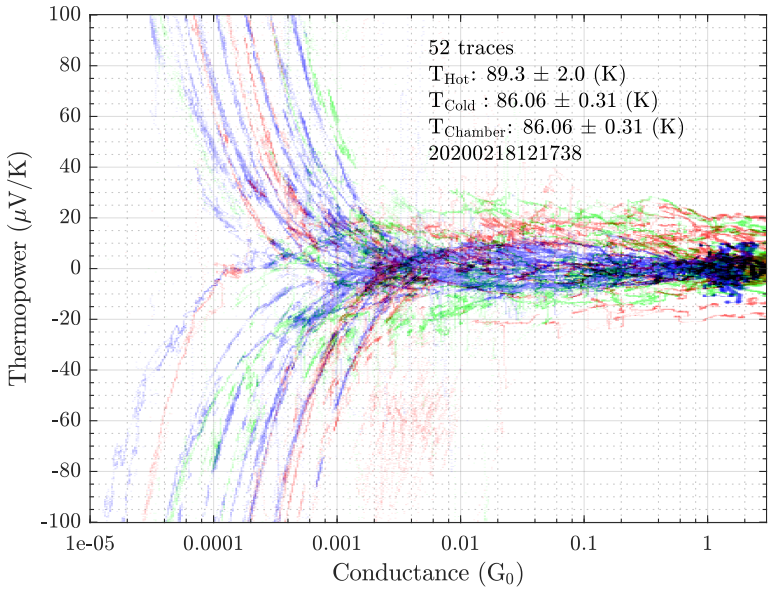



**Figure 8.5.:** The color histogram generated from the thermopower of the opening traces of sample T0007C with terphenyldithiol. The blue color are the traces without the laser heating, the red color are the traces where the IV-converter side of the junction is heated, and the green color are the traces with the heating on the other side.
Unfortunately the offset voltage drifts during the measurement are too big to see any effects from the heating.
The errors of the offsets show up more pronounced the lower the conductance.

age drifts of the IV-converter front-end are too big to discern between the data with and without the laser. Therefore no conclusions can be made from the results shown in figure 8.5. A different set of measurement electronics was built (see appendix C.7) to reduce to voltage drift, and the motor was further decoupled, but no measurements were performed with these improvements.

# 9. Outlook

The measurements on gold contacts are reproducible and fairly complete. However, the theoretical understanding may be the subject of further theoretical DFT an NEGF studies.
Having shown the feasibility and precision of the developed measurement approach, this method should be expanded to study different metals with more complicated electronic structures. Possible candidates would be aluminium, silver and platinum. This fabrication is not suitable for metals where the oxidation in air is not self passivating, since the metallic properties of the thin film would not be preserved during the many necessary fabrication steps.
The measurements on the molecule contacts highlight the increased electronic requirements since the resistance of the junctions is orders of magnitude higher than the resistances of the metallic contacts. A new electronic approach e.g. with a completely cold IV-converter might address some of the problems, but the low probability of the formation of molecular contacts will remain. To improve on the measurements presented before, the drifting of the electronics needs to be prevented or at least significantly reduced, and the number of traces needs to be orders of magnitude higher to allow a clear detection of the molecule in the conductance histograms. With these requirements fulfilled, it would be possible to measure the thermopower of individual molecules and thereby deduce information of the position of the HOMO and LUMO orbitals. This can be taken further by systematically analyzing groups of similar molecules, where the energy levels are shifted by side chains with different electron affinity.

# 10. Summary and Conclusion

This thesis shows that using a pulsed laser as a heating source in MCBJs might be an appealing solution from the electronical point of view since it solves the problems of amplifier drift. However, the mechanical deformation induced by the thermal expansion destroys the stability of the contact inherent in the MCBJ approach at chamber temperatures between 10 K and room temperature.
Therefore the setup was changed to work with a static temperature distribution inside the sample. To measure the temperature difference created by the laser, a platinum resistance thermometer was integrated into the sample structure. This allowed to have the temperature information with an accuracy of below 20 %. The problem of the amplifier drifts was solved by measuring opening and closing traces without the laser heating, and any additional offsets would show up in the curves where the heating is taking place on the other side of the junction. Using these correction mechanisms, accurate thermovoltage measurements are possible, which can be converted into thermopower with the known temperature difference. The results are shown in figure 10.1. The thermopower is negative and close to zero at low temperatures, and at room temperature a value of about $-1.1\,\mu V\,K^{-1}$ was found. Between the two temperatures, the data shows an unexpected clear deviation from any linear temperature dependance: The thermopower continuously approaches the minimal value of $-2\,\mu V\,K^{-1}$ between 150 and 250 K.
The measurements of the thermopower of molecules were also performed with static heating, but they needed different electronics due to the different resistances of molecular contacts. Unfortunately, these electronics were highly susceptible to temperature changes. Therefore no usable thermovoltage measurements could be performed on the molecular contacts yet.
Even though this thesis does not show usable thermopower measurements of molecules, it lays a thorough foundation for further experiments: The pulsed measurement approach was extensively studied and tried on metallic contacts, since it is experimentally easy compared to later solutions and very appealing from an electronical point of view, because measuring a repeating voltage change at about 30 Hz is not problematic even with signals in the range of a few micro volts.
However, this approach created two problems in metallic contacts: due to the time dependence of the temperature, the simulations had many unknown parameters and these simulations would have been the only way to estimate the temperature difference across the junction in the samples for this measurement. This put strong requirements for quantitative reliability on the simulations, which could not be met, even by using structures designed to facilitate the simulations in order to limit the number of unknown parameters. The second problem in metallic contacts was of experimental nature: The repeated mechanical deformation of the metallic junction due to the thermal expansion ruined the otherwise exceptional stability of the metallic contact in the MCBJ at cryogenic temperatures. Thus the metallic contact geometry is different on every laser pulse and the

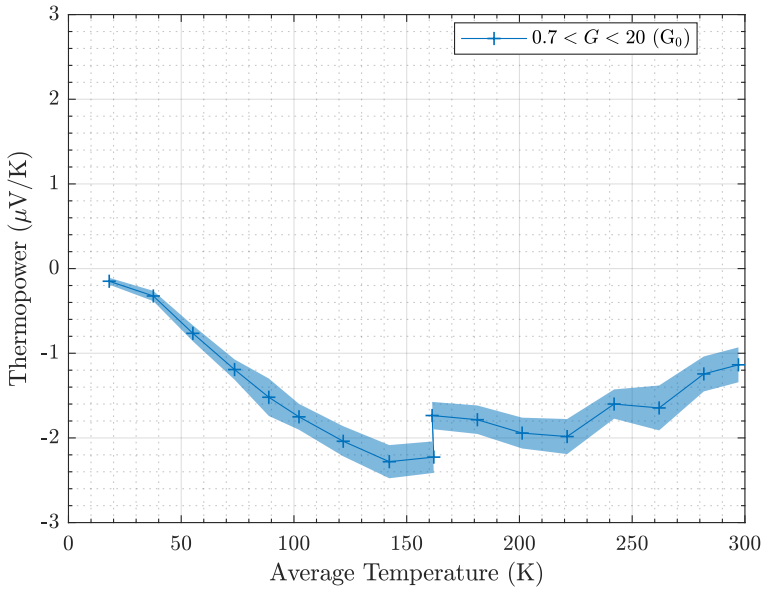



**Figure 10.1.:** The thermopower of an atomic gold contact has a strongly nonlinear temperature dependence.

voltage signal is not repeated with every pulse. This hinders the electronic benefits of having a repeating signal, and also limits the usable temperature differences in metallic contacts since large temperature differences lead to large deformations during the pulse and to large uncertainties in the conductance of metallic contacts. These two systematic problems in metallic contacts render the pulsed approach unusable for thermopower measurements employing my specific sample design of the MCBJs at 86 K.
In molecular contacts, the molecule bridges the gap and the conductance is defined by the molecule. For such a configuration, the molecule does not need to be anchored at the very apex of the breaking metallic contact. When such a contact is moved due to thermal expansion, the contact points of the molecule to the leads can move. This motion changes the geometry of the anchoring sites but the molecule might be unaffected. Still, the stress on the molecule and the changes in the contacts can affect the thermopower. The influence of the thermal expansion on the thermovoltage was observed by [Mor14] on a molecular contact of benzenedithiol in a gold break junction.
More important is that a precisely known temperature difference is required to make measurements of the thermopower of molecules. In the pulsed approach no suitable parameters for the description of the sample could be found, hence, the simulations do not provide reliable information about the temperature difference across the constriction. Without a known temperature difference the pulsed approach is unusable for thermopower measurements on molecules.
So a completely different approach to the thermopower measurements was developed, that completely eliminated the problem of thermal expansion. By integrating a thermometer into the lithographic structure, it was not only possible to measure the temperature directly, but the requirements of the simulations could be relaxed. By performing static simulations with only the ratio of the temperature difference across the junction to the

temperature increase in the thermometer, the simulation results are primarily affected by the thermal conductivities of the gold film and the polyimide layer. But even with comfortable uncertainties in these two parameters, the resulting ratio was found to be accurate to within 20 % since the geometry of the sample is designed to minimize these influences.
Developing this static approach was critical to have accurate information about the generated temperature difference across the junction. This approach was then used to measure the thermovoltage of an atomic gold contact as function of the conductance and at different temperatures. Here, the static heating allowed to measure normal opening and closing traces while simultaneously detecting the thermovoltage and with the known temperature difference also the thermopower. This allows to make not only statistical statements, but shows how the thermopower reacts to each individual reconfiguration of the contact during the bending.
With this foundation, reproducible measurements of gold contacts at different temperatures were performed and an unexpected temperature dependence was found. Nevertheless, the data is in full agreement with previous work at the two temperature extremes of the measurements.

# 11. Acknowledgment

First I need to thank Prof. Dr. Elke Scheer for giving me the opportunity for my PhD and the freedom to change my setup in quite significant ways. Secondly, I want to thank Prof. Dr. Johannes Boneberg for his support and experience not only with experiments but also with some more strategic considerations, and of course with "everything laser". Third, I thank Prof. Dr. Paul Leiderer for the many discussions on how some of the problematic unknowns could be measured as part of a "quick experiment". These discussions stimulated the outside the box thinking that led to some of the systematic changes to the experiment.
I am grateful for Dr. Sergii Snegir supplying the molecules and his expertise in the choice of solvents as well as his experience with experiments in general and his support of my experiment especially.
Deep gratitude goes to the technicians and the mechanical workshop, who had the unenviable task of realizing some of my weird ideas. I want to mention Louis Kukk for taking the time to solve the many small problems on incredibly short notice and within a few hours, without compromising his urge for accuracy and elegance.[1] Ansgar Fischer introduced me to the wonderful world of 3D modelling which lead to him having the endless task of ground my "perfect ideas" with remarks of what is and is not possible for the workshop. Simon Haus produced many of my samples and introduced me to the lithography process and the necessary devices. Without this knowledge I would have been unable to fabricate the overly complicated samples necessary for the static measurements.
Last but not least I would like to thank my friends and family for enduring me during the difficult phases and their continuous support which provided the motivation especially for some of the more unconventional approaches implemented as part of this thesis.[2]

[1] "Das Auge misst mit." (Louis Kukk)
[2] You know who you are, and what you have done...

# Bibliography


[Bah08] K. Baheti, J. A. Malen, P. Doak, P. Reddy, S.-Y. Jang, T. D. Tilley, A. Majumdar and R. A. Segalman, *Probing the Chemistry of Molecular Heterojunctions Using Thermoelectricity*, Nano Letters 8(2) pages 715–719 (2008).

[Cri70] R. S. Crisp and J. Rungis, *Thermoelectric power and thermal conductivity in the silver-gold alloy system from 3-300°K*, The Philosophical Magazine: A Journal of Theoretical Experimental and Applied Physics 22(176) pages 217–236 (1970), URL https://doi.org/10.1080/14786437008228219, https://doi.org/10.1080/14786437008228219.

[Cue10] J. C. Cuevas and E. Scheer, *Molecular Electronics*, volume 1 of *World Scientific Series in Nanascience and Nanotechnology*, World Scientific (2010).

[Elb05] M. Elbing, R. Ochs, M. Koentopp, M. Fischer, C. von Hanisch, F. Weigend, F. Evers, H. B. Weber and M. Mayor, *A single-molecule diode*, Proceedings of the National Academy of Sciences 102(25) pages 8815–8820 (2005).

[Eva15] C. Evangeli, M. Matt, L. Rincón-García, F. Pauly, P. Nielaba, G. Rubio-Bollinger, J. C. Cuevas and N. Agraït, *Quantum Thermopower of Metallic Atomic-Size Contacts at Room Temperature*, Nano Letters 15(2) pages 1006–1011 (2015), URL https://doi.org/10.1021/nl503853v, pMID: 25607343, https://doi.org/10.1021/nl503853v.

[Gim87] J. Gimzewski, R. Möller, D. Pohl and R. Schlittler, *Transition from tunneling to point contact investigated by scanning tunneling microscopy and spectroscopy*, Surface Science 189-190 pages 15–23 (1987).

[Guo13] S. Guo, G. Zhou and N. Tao, *Single Molecule Conductance, Thermopower, and Transition Voltage*, Nano Letters 13(9) pages 4326–4332 (2013).

[Int19] Intel, *Specifiactions for i9-10980XE* (2019), accessed 04.04.2020.

[Jan] Janis Research Company, Inc., 2 Jewel Drive P.O. Box 696 Wilmington, MA 01887-0696, *Operation Instructions for the Janis Model SVT Research Cryostat.*

[Joh72] P. B. Johnson and R. W. Christy, *Optical Constants of the Noble Metals*, Phys. Rev. B 6 pages 4370–4379 (1972), URL https://link.aps.org/doi/10.1103/PhysRevB.6.4370.

[Kap41] P. L. Kapitza, *The Study of Heat Transfer in Helium II*, J. Phys. (Moscow) 4 pages 181 (1941), URL https://ci.nii.ac.jp/naid/10028199917/en/.

[Kop16] B. Kopp, *Thermopower of Atomic-Size Contacts at Low Temperature*, Ph.D. thesis, Universität Konstanz, Konstanz (2016).

[Lor18] T. Lorenz, *Wechselspiel von Vielteilchen-Transport und Ladungseffekten in supraleitenden Einzelelektronentransistoren*, Ph.D. thesis, Universität Konstanz, Konstanz (2018).

[Lud99] B. Ludoph and J. M. v. Ruitenbeek, *Thermopower of atomic-size metallic contacts*, Phys. Rev. B 59 pages 12290–12293 (1999), URL https://link.aps.org/doi/10.1103/PhysRevB.59.12290.

[Lud00] B. Ludoph and J. M. van Ruitenbeek, *Conductance fluctuations as a tool for investigating the quantum modes in atomic-size metallic contacts*, Physical Review B 61(3) pages 2273–2285 (2000).

[Mat15] R. Matsushita, S. Kaneko, S. Fujii, H. Nakamura and M. Kiguchi, *Temperature dependence of the thermopower and its variation of the Au atomic contact*, Nanotechnology 26(4) pages 045709 (2015), URL https://doi.org/10.1088%2F0957-4484%2F26%2F4%2F045709.

[Mia18] R. Miao, H. Xu, M. Skripnik, L. Cui, K. Wang, K. G. L. Pedersen, M. Leijnse, F. Pauly, K. Wärnmark, E. Meyhofer, P. Reddy and H. Linke, *Influence of Quantum Interference on the Thermoelectric Properties of Molecular Junctions*, Nano Letters 18(9) pages 5666–5672 (2018).

[Moo65] G. E. Moore, *Cramming more components onto integrated circuits*, Electronics 38(8) pages 114 ff (1965).

[Mor14] T. Morikawa, A. Arima, M. Tsutsui and M. Taniguchi, *Thermoelectric voltage measurements of atomic and molecular wires using microheater-embedded mechanically-controllable break junctions*, Nanoscale 6 pages 8235–8241 (2014), URL http://dx.doi.org/10.1039/C4NR00127C.

[Nak13] H. Nakamura, T. Ohto, T. Ishida and Y. Asai, *Thermoelectric Efficiency of Organometallic Complex Wires via Quantum Resonance Effect and Long-Range Electric Transport Property*, Journal of the American Chemical Society 135(44) pages 16545–16552 (2013).

[Nat] National Institute of Standards and Technology, *Material Properties: Polyimide (Kapton)*, URL https://trc.nist.gov/cryogenics/materials/Polyimide%20Kapton/PolyimideKapton_rev.htm, accessed 20.03.2020.

[Oso08] E. A. Osorio, T. Bjørnholm, J.-M. Lehn, M. Ruben and H. S. J. van der Zant, *Single-molecule transport in three-terminal devices*, Journal of Physics: Condensed Matter 20(37) pages 374121 (2008).

[Pau11] F. Pauly, J. K. Viljas, M. Bürkle, M. Dreher, P. Nielaba and J. C. Cuevas, *Molecular dynamics study of the thermopower of Ag, Au, and Pt nanocontacts*, Phys. Rev. B 84 pages 195420 (2011), URL https://link.aps.org/doi/10.1103/PhysRevB.84.195420.

[Red07] P. Reddy, S.-Y. Jang, R. A. Segalman and A. Majumdar, *Thermoelectricity in Molecular Junctions*, Science 315(5818) pages 1568–1571 (2007), URL `https://science.sciencemag.org/content/315/5818/1568`, `https://science.sciencemag.org/content/315/5818/1568.full.pdf`.

[Ree97] M. A. Reed, C. Zhou, C. J. Muller, T. P. Burgin and J. M. Tour, *Conductance of a Molecular Junction*, Science 278(5336) pages 252–254 (1997).

[Sim92] N. J. Simon, E. S. Drexler and R. P. Reed, *Properties of Copper and Copper Alloys at Cryogenic Temperatures (Monograph 177)* (1992).

[TSM19] TSMC, *TSMC Unveils 6-nanometer Process* (2019), URL `https://www.tsmc.com/uploadfile/pr/newspdf/THWQWQTHTH/NEWS_FILE_EN.pdf`, accessed 04.04.2020.

[Tsu13] M. Tsutsui, T. Morikawa, A. Arima and M. Taniguchi, *Thermoelectricity in atom-sized junctions at room temperatures*, Scientific Reports 3 pages 3326 (2013), URL `https://doi.org/10.1038/srep03326`.

[Vil07] J. K. Viljas and J. C. Cuevas, *Role of electronic structure in photoassisted transport through atomic-sized contacts*, Physical Review B 75(7) (2007).

[Wan13] H. Wang, J. Liu, X. Zhang and K. Takahashi, *Breakdown of Wiedemann–Franz law in individual suspended polycrystalline gold nanofilms down to 3K*, International Journal of Heat and Mass Transfer 66 pages 585 – 591 (2013), URL `http://www.sciencedirect.com/science/article/pii/S0017931013006200`.

[Wid11] J. R. Widawsky, P. Darancet, J. B. Neaton and L. Venkataraman, *Simultaneous Determination of Conductance and Thermopower of Single Molecule Junctions*, Nano Letters 12(1) pages 354–358 (2011).

[Wid13] J. R. Widawsky, W. Chen, H. Vázquez, T. Kim, R. Breslow, M. S. Hybertsen and L. Venkataraman, *Length-Dependent Thermopower of Highly Conducting Au–C Bonded Single Molecule Junctions*, Nano Letters 13(6) pages 2889–2894 (2013).

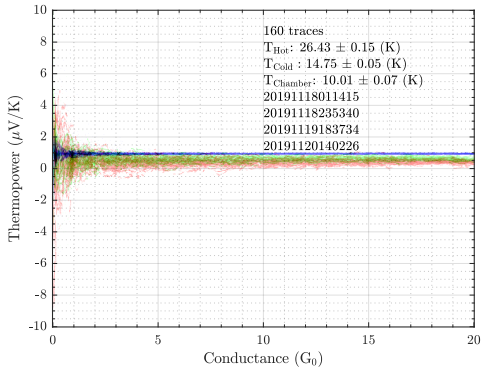
[Yza18] G. Yzambart, L. Rincón-García, A. A. Al-Jobory, A. K. Ismael, G. Rubio-Bollinger, C. J. Lambert, N. Agraït and M. R. Bryce, *Thermoelectric Properties of 2,7-Dipyridylfluorene Derivatives in Single-Molecule Junctions*, The Journal of Physical Chemistry C 122(48) pages 27198–27204 (2018).

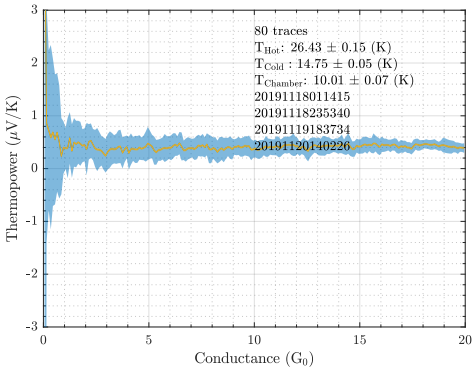
# A. Complete Measurement Results

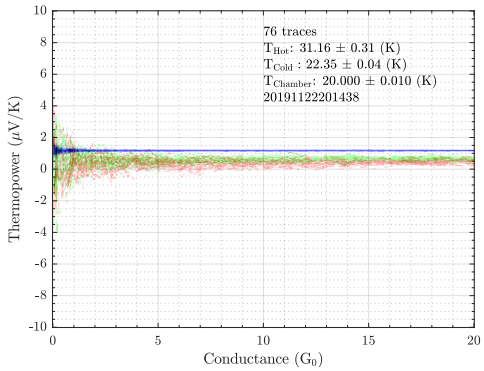
In the following the results of the statistical analysis of the static thermopower measurements are shown for every measurement included in figures 7.17 to 7.19. The graphs are generated in the same manner as figures 7.14 and 7.16: In the 2D histogram plots, the blue data is without laser heating, the red data is with the laser heating the side of the thermometer, and the green data is with the heating on the other side. The temperature difference used for the analysis of the data with the laser heating the other side is taken from the temperature difference when the laser is heating the side of the thermometer. This assumption neglects several sources for asymmetry: The position might not be symmetric with regards to the constriction. The laser power might be different since the laser takes a different path through the optics in front of the cryostat. The heat diffusion might be affected by the thermometer structures. And finally the laser focus is optimized for the side with the thermometer, not for the other side. Therefore, this data can deviate significantly from the results of the data where the temperature increase is measured directly.
In addition to the 2D histograms, the statistical results of the data with the laser heating the side of the thermometer are shown. Here the blue areas demonstrate the standard deviation and the yellow areas show the uncertainties due to the temperature difference.

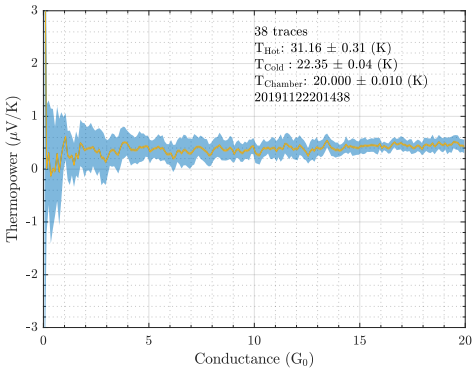
## A.1. T0115S

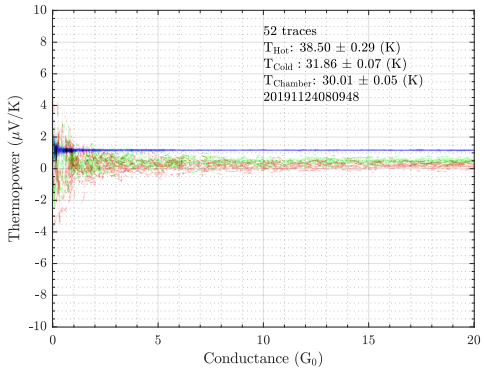
Sample T0115S was measured from 10 to 295 K with the "Thermospannungsmessung_statisch" software (see appendix E.40 on page 354). Due to experimental problems, this sample was broken in ambient conditions before the measurements.

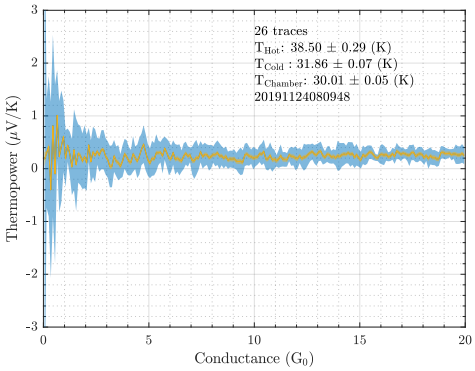

160 traces
$T_{Hot}$: 26.43 ± 0.15 (K)
$T_{Cold}$ : 14.75 ± 0.05 (K)
$T_{Chamber}$: 10.01 ± 0.07 (K)
20191118011415
20191118235340
20191119183734
20191120140226
Thermopower ($\mu V/K$)
Conductance ($G_0$)

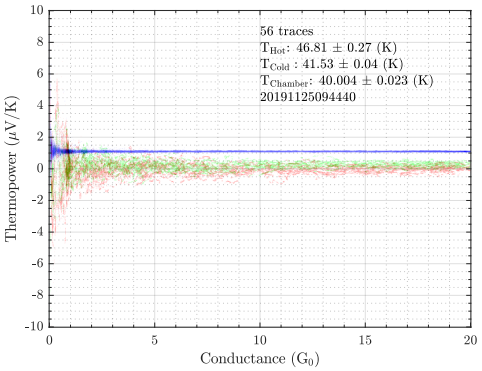

80 traces
$T_{Hot}$: 26.43 ± 0.15 (K)
$T_{Cold}$ : 14.75 ± 0.05 (K)
$T_{Chamber}$: 10.01 ± 0.07 (K)
20191118011415
20191118235340
20191119183734
20191120140226
Thermopower ($\mu V/K$)
Conductance ($G_0$)

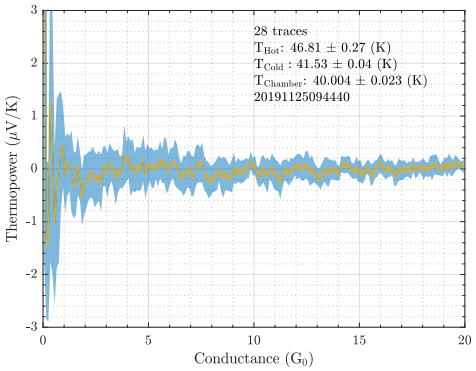

76 traces
$T_{Hot}$: 31.16 ± 0.31 (K)
$T_{Cold}$ : 22.35 ± 0.04 (K)
$T_{Chamber}$: 20.000 ± 0.010 (K)
20191122201438
Thermopower ($\mu V/K$)
Conductance ($G_0$)

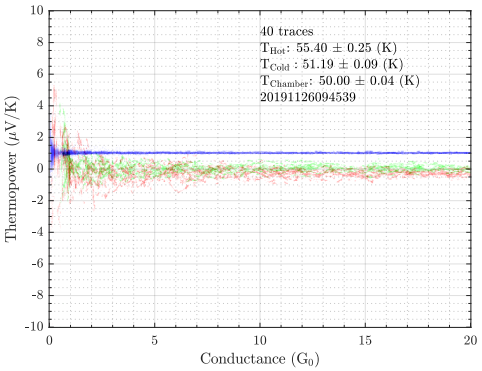

38 traces
$T_{Hot}$: 31.16 ± 0.31 (K)
$T_{Cold}$ : 22.35 ± 0.04 (K)
$T_{Chamber}$: 20.000 ± 0.010 (K)
20191122201438
Thermopower ($\mu V/K$)
Conductance ($G_0$)

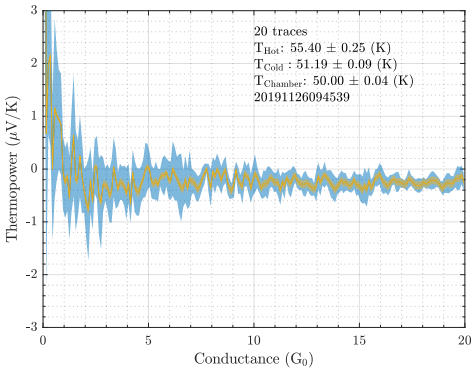

52 traces
$T_{Hot}$: 38.50 ± 0.29 (K)
$T_{Cold}$ : 31.86 ± 0.07 (K)
$T_{Chamber}$: 30.01 ± 0.05 (K)
20191124080948
Thermopower ($\mu V/K$)
Conductance ($G_0$)

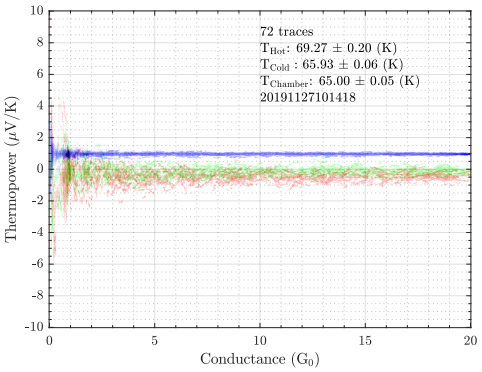

26 traces
$T_{Hot}$: 38.50 ± 0.29 (K)
$T_{Cold}$ : 31.86 ± 0.07 (K)
$T_{Chamber}$: 30.01 ± 0.05 (K)
20191124080948
Thermopower ($\mu V/K$)
Conductance ($G_0$)

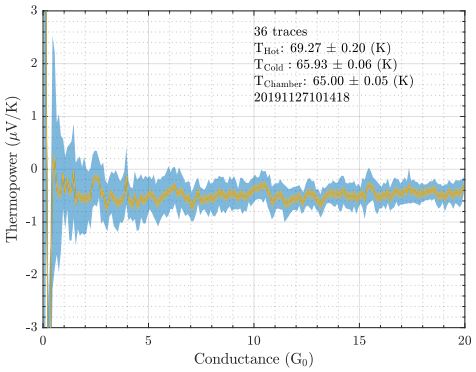

56 traces
$T_{Hot}$: 46.81 ± 0.27 (K)
$T_{Cold}$ : 41.53 ± 0.04 (K)
$T_{Chamber}$: 40.004 ± 0.023 (K)
20191125094440
Thermopower (μV/K)
Conductance ($G_0$)

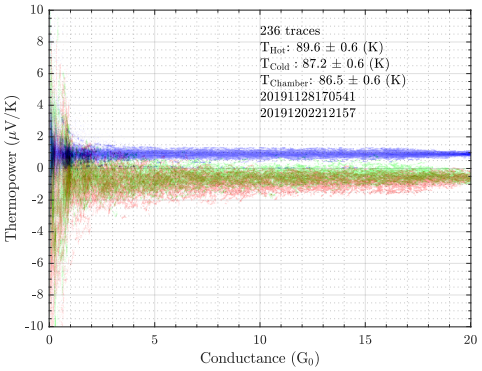

28 traces
$T_{Hot}$: 46.81 ± 0.27 (K)
$T_{Cold}$ : 41.53 ± 0.04 (K)
$T_{Chamber}$: 40.004 ± 0.023 (K)
20191125094440
Thermopower (μV/K)
Conductance ($G_0$)

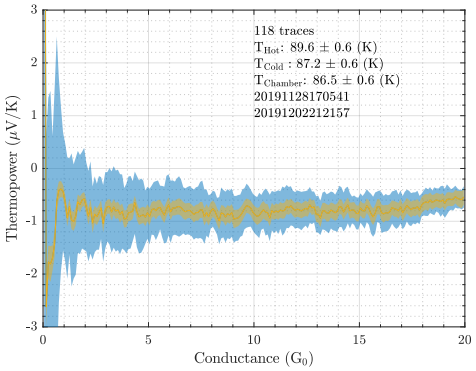

40 traces
$T_{Hot}$: 55.40 ± 0.25 (K)
$T_{Cold}$ : 51.19 ± 0.09 (K)
$T_{Chamber}$: 50.00 ± 0.04 (K)
20191126094539
Thermopower (μV/K)
Conductance ($G_0$)

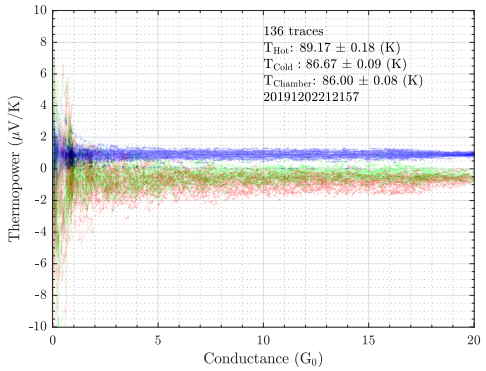

20 traces
$T_{Hot}$: 55.40 ± 0.25 (K)
$T_{Cold}$ : 51.19 ± 0.09 (K)
$T_{Chamber}$: 50.00 ± 0.04 (K)
20191126094539
Thermopower (μV/K)
Conductance ($G_0$)

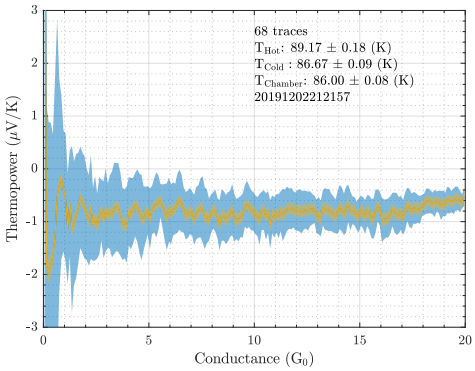

72 traces
$T_{Hot}$: 69.27 ± 0.20 (K)
$T_{Cold}$ : 65.93 ± 0.06 (K)
$T_{Chamber}$: 65.00 ± 0.05 (K)
20191127101418
Thermopower (μV/K)
Conductance ($G_0$)

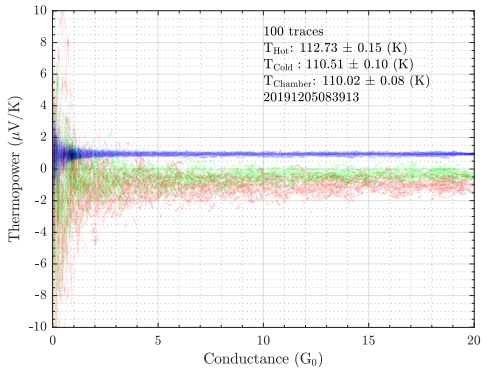

36 traces
$T_{Hot}$: 69.27 ± 0.20 (K)
$T_{Cold}$ : 65.93 ± 0.06 (K)
$T_{Chamber}$: 65.00 ± 0.05 (K)
20191127101418
Thermopower (μV/K)
Conductance ($G_0$)

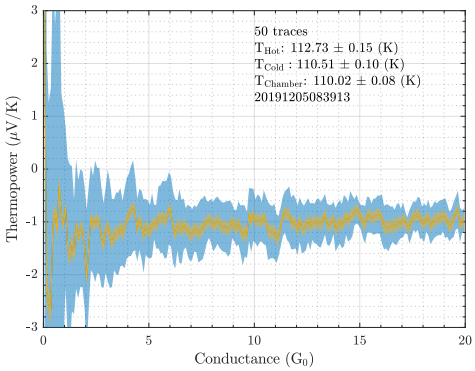

236 traces
$T_{Hot}$: 89.6 ± 0.6 (K)
$T_{Cold}$ : 87.2 ± 0.6 (K)
$T_{Chamber}$: 86.5 ± 0.6 (K)
20191128170541
20191202212157
Thermopower (μV/K)
Conductance ($G_0$)

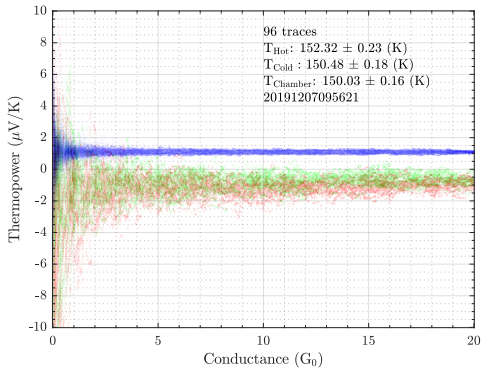

118 traces
$T_{Hot}$: 89.6 ± 0.6 (K)
$T_{Cold}$ : 87.2 ± 0.6 (K)
$T_{Chamber}$: 86.5 ± 0.6 (K)
20191128170541
20191202212157
Thermopower (μV/K)
Conductance ($G_0$)

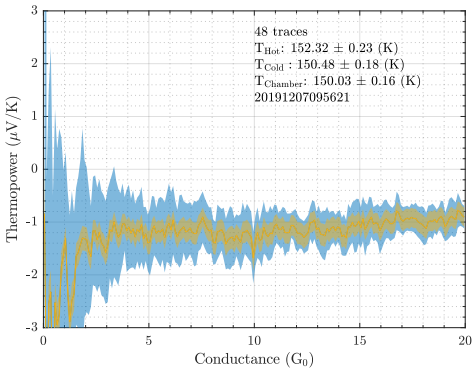

136 traces
T_Hot: 89.17 ± 0.18 (K)
T_Cold : 86.67 ± 0.09 (K)
T_Chamber: 86.00 ± 0.08 (K)
20191202212157
Thermopower (μV/K)
Conductance (G0)

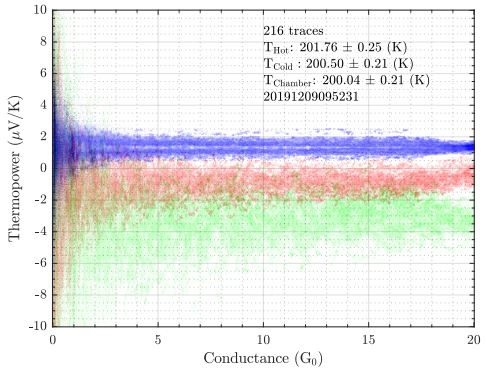

68 traces
T_Hot: 89.17 ± 0.18 (K)
T_Cold : 86.67 ± 0.09 (K)
T_Chamber: 86.00 ± 0.08 (K)
20191202212157
Thermopower (μV/K)
Conductance (G0)

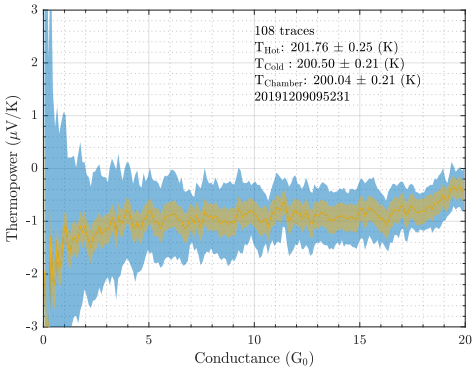

100 traces
T_Hot: 112.73 ± 0.15 (K)
T_Cold : 110.51 ± 0.10 (K)
T_Chamber: 110.02 ± 0.08 (K)
20191205083913
Thermopower (μV/K)
Conductance (G0)

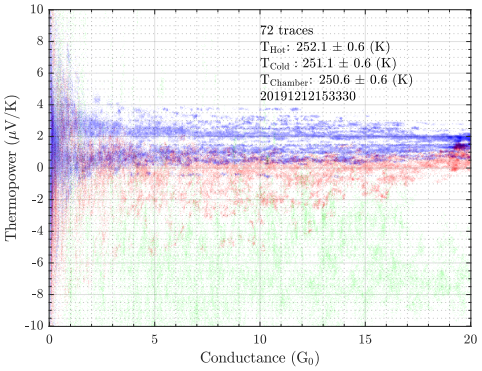

50 traces
T_Hot: 112.73 ± 0.15 (K)
T_Cold : 110.51 ± 0.10 (K)
T_Chamber: 110.02 ± 0.08 (K)
20191205083913
Thermopower (μV/K)
Conductance (G0)

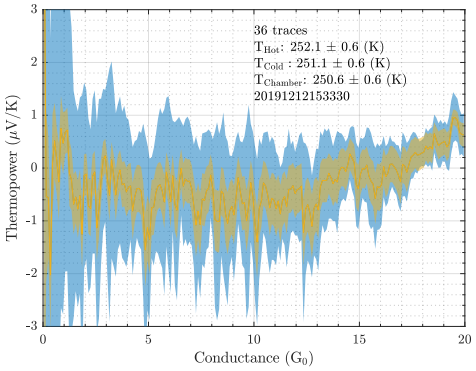

96 traces
T_Hot: 152.32 ± 0.23 (K)
T_Cold : 150.48 ± 0.18 (K)
T_Chamber: 150.03 ± 0.16 (K)
20191207095621
Thermopower (μV/K)
Conductance (G0)

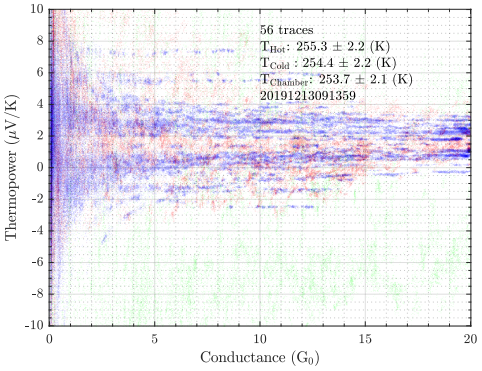

48 traces
T_Hot: 152.32 ± 0.23 (K)
T_Cold : 150.48 ± 0.18 (K)
T_Chamber: 150.03 ± 0.16 (K)
20191207095621
Thermopower (μV/K)
Conductance (G0)

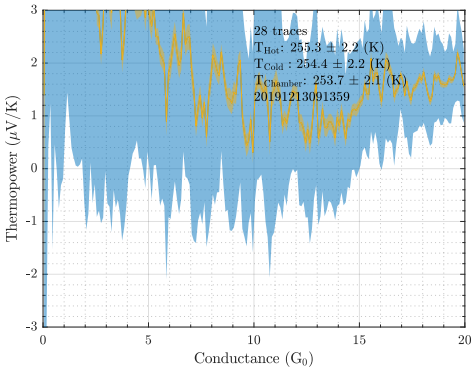

216 traces
T_Hot: 201.76 ± 0.25 (K)
T_Cold : 200.50 ± 0.21 (K)
T_Chamber: 200.04 ± 0.21 (K)
20191209095231
Thermopower (μV/K)
Conductance (G0)

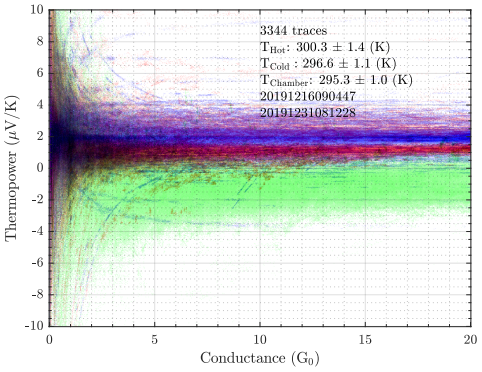

108 traces
T_Hot: 201.76 ± 0.25 (K)
T_Cold : 200.50 ± 0.21 (K)
T_Chamber: 200.04 ± 0.21 (K)
20191209095231
Thermopower (μV/K)
Conductance (G0)

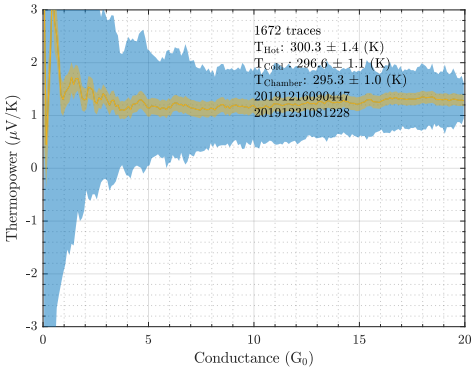

72 traces
T_Hot: 252.1 ± 0.6 (K)
T_Cold : 251.1 ± 0.6 (K)
T_Chamber: 250.6 ± 0.6 (K)
20191212153330
Thermopower (μV/K)
Conductance (G_0)


36 traces
T_Hot: 252.1 ± 0.6 (K)
T_Cold : 251.1 ± 0.6 (K)
T_Chamber: 250.6 ± 0.6 (K)
20191212153330
Thermopower (μV/K)
Conductance (G_0)


56 traces
T_Hot: 255.3 ± 2.2 (K)
T_Cold : 254.4 ± 2.2 (K)
T_Chamber: 253.7 ± 2.1 (K)
20191213091359
Thermopower (μV/K)
Conductance (G_0)

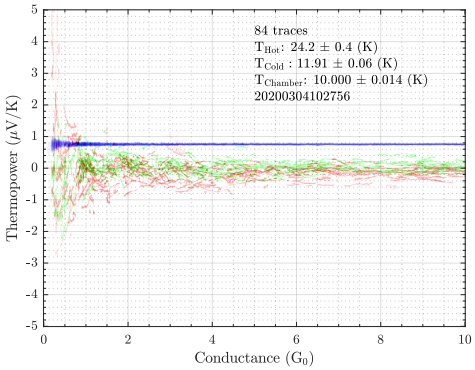

28 traces
T_Hot: 255.3 ± 2.2 (K)
T_Cold : 254.4 ± 2.2 (K)
T_Chamber: 253.7 ± 2.1 (K)
20191213091359
Thermopower (μV/K)
Conductance (G_0)

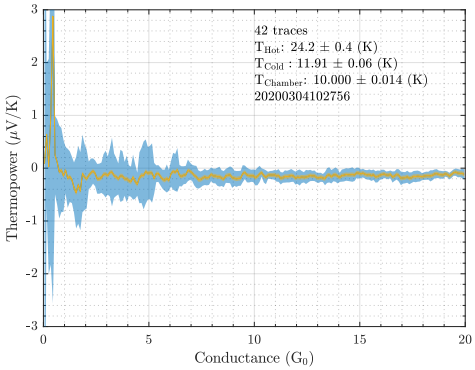

3344 traces
T_Hot: 300.3 ± 1.4 (K)
T_Cold : 296.6 ± 1.1 (K)
T_Chamber: 295.3 ± 1.0 (K)
20191216090447
20191231081228
Thermopower (μV/K)
Conductance (G_0)

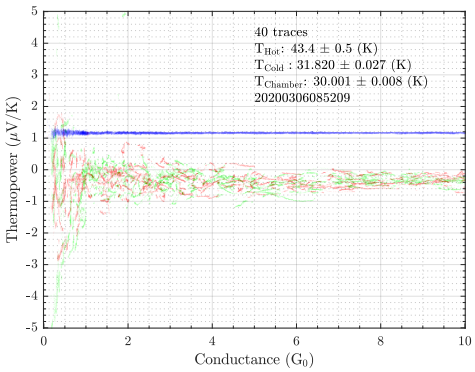

1672 traces
T_Hot: 300.3 ± 1.4 (K)
T_Cold : 296.6 ± 1.1 (K)
T_Chamber: 295.3 ± 1.0 (K)
20191216090447
20191231081228
Thermopower (μV/K)
Conductance (G_0)

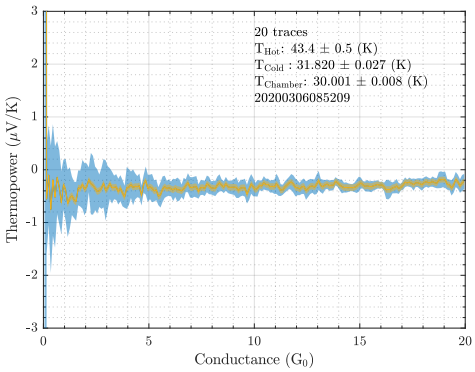
## A.2. T0143S

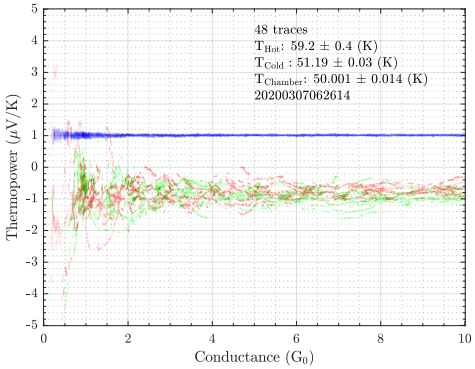
Sample T0143S was measured from 10 to 160 K with the "Thermospannungsmessung_statisch_V2" software (see appendix E.41 on page 357). This sample was broken in vacuum before the measurements to verify that the sample worked as expected. Unfortunately, the sample could not be closed any more during the measurements at 160 K, therefore no data at higher temperatures could be recorded.

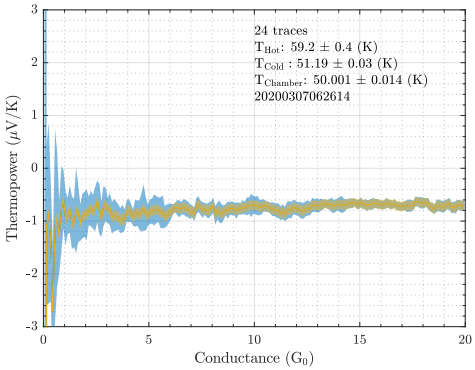


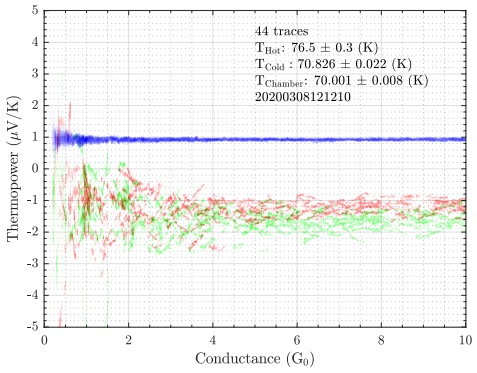



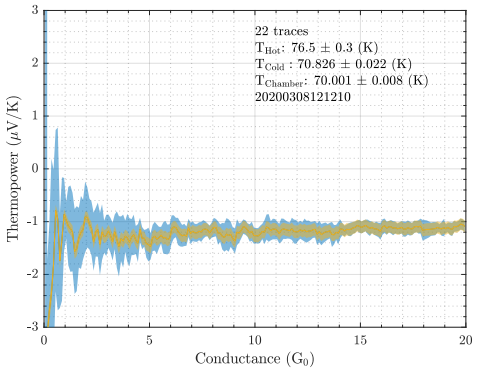



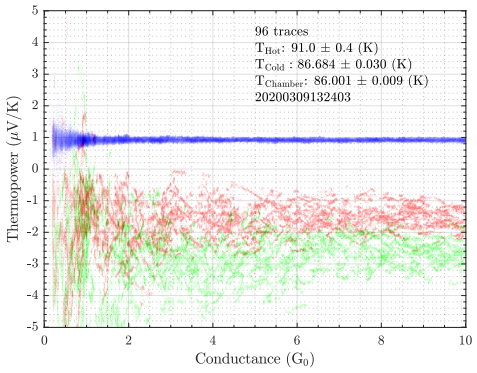



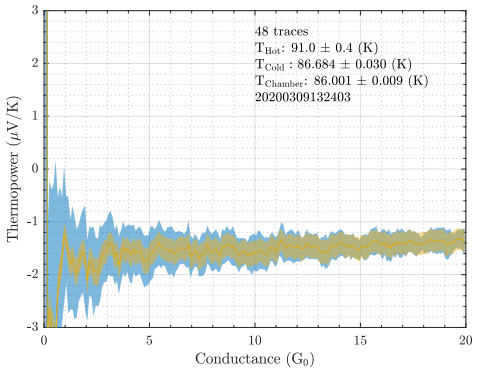



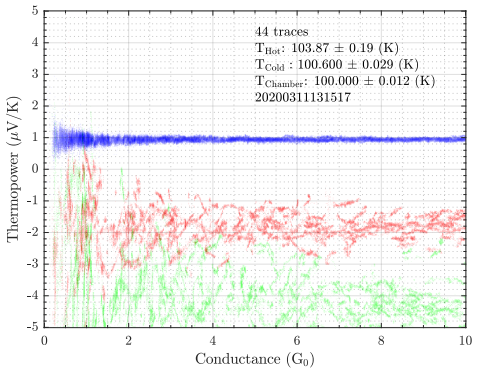



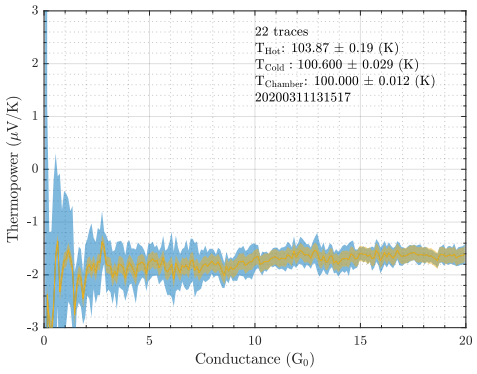

44 traces
$T_{Hot}$: 76.5 ± 0.3 (K)
$T_{Cold}$ : 70.826 ± 0.022 (K)
$T_{Chamber}$: 70.001 ± 0.008 (K)
20200308121210
Thermopower ($\mu V/K$)
Conductance ($G_0$)

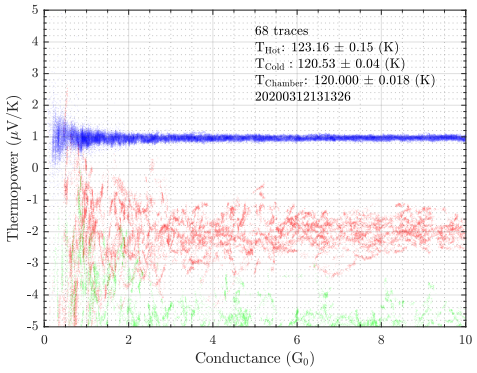

22 traces
$T_{Hot}$: 76.5 ± 0.3 (K)
$T_{Cold}$ : 70.826 ± 0.022 (K)
$T_{Chamber}$: 70.001 ± 0.008 (K)
20200308121210
Thermopower ($\mu V/K$)
Conductance ($G_0$)

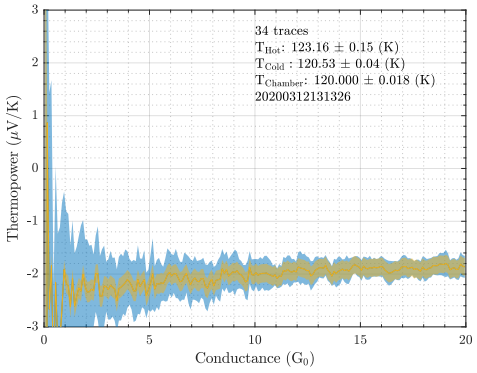

96 traces
$T_{Hot}$: 91.0 ± 0.4 (K)
$T_{Cold}$ : 86.684 ± 0.030 (K)
$T_{Chamber}$: 86.001 ± 0.009 (K)
20200309132403
Thermopower ($\mu V/K$)
Conductance ($G_0$)

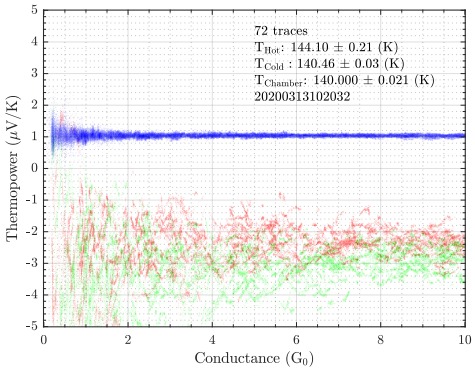

48 traces
$T_{Hot}$: 91.0 ± 0.4 (K)
$T_{Cold}$ : 86.684 ± 0.030 (K)
$T_{Chamber}$: 86.001 ± 0.009 (K)
20200309132403
Thermopower ($\mu V/K$)
Conductance ($G_0$)

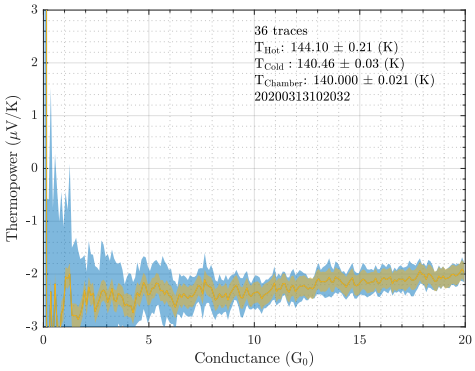

44 traces
$T_{Hot}$: 103.87 ± 0.19 (K)
$T_{Cold}$ : 100.600 ± 0.029 (K)
$T_{Chamber}$: 100.000 ± 0.012 (K)
20200311131517
Thermopower ($\mu V/K$)
Conductance ($G_0$)

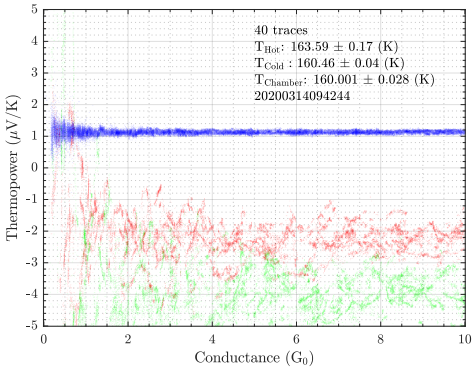

22 traces
$T_{Hot}$: 103.87 ± 0.19 (K)
$T_{Cold}$ : 100.600 ± 0.029 (K)
$T_{Chamber}$: 100.000 ± 0.012 (K)
20200311131517
Thermopower ($\mu V/K$)
Conductance ($G_0$)

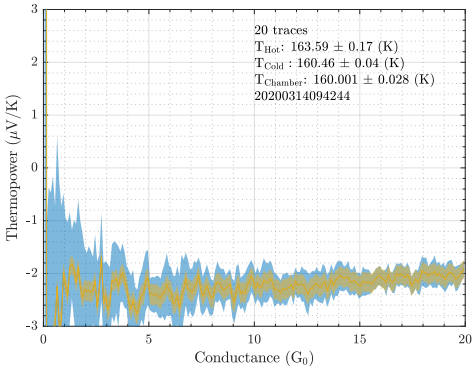

68 traces
$T_{Hot}$: 123.16 ± 0.15 (K)
$T_{Cold}$ : 120.53 ± 0.04 (K)
$T_{Chamber}$: 120.000 ± 0.018 (K)
20200312131326
Thermopower ($\mu V/K$)
Conductance ($G_0$)


34 traces
$T_{Hot}$: 123.16 ± 0.15 (K)
$T_{Cold}$ : 120.53 ± 0.04 (K)
$T_{Chamber}$: 120.000 ± 0.018 (K)
20200312131326
Thermopower ($\mu V/K$)
Conductance ($G_0$)


72 traces
T_Hot: 144.10 ± 0.21 (K)
T_Cold : 140.46 ± 0.03 (K)
T_Chamber: 140.000 ± 0.021 (K)
20200313102032
Thermopower (μV/K)
Conductance (G0)

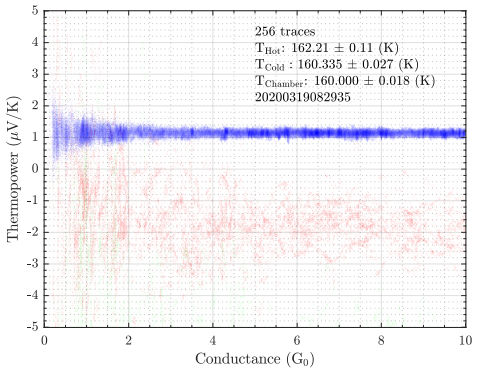

36 traces
T_Hot: 144.10 ± 0.21 (K)
T_Cold : 140.46 ± 0.03 (K)
T_Chamber: 140.000 ± 0.021 (K)
20200313102032
Thermopower (μV/K)
Conductance (G0)

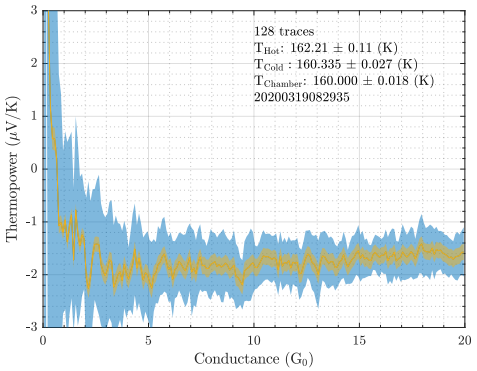

40 traces
T_Hot: 163.59 ± 0.17 (K)
T_Cold : 160.46 ± 0.04 (K)
T_Chamber: 160.001 ± 0.028 (K)
20200314094244
Thermopower (μV/K)
Conductance (G0)

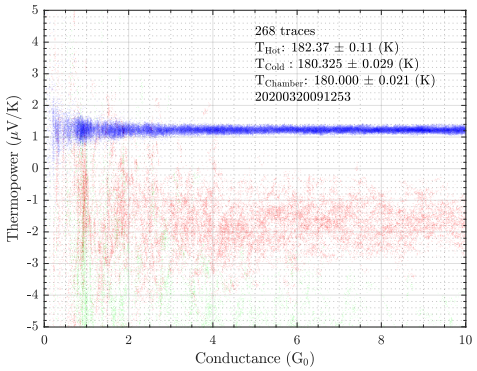

20 traces
T_Hot: 163.59 ± 0.17 (K)
T_Cold : 160.46 ± 0.04 (K)
T_Chamber: 160.001 ± 0.028 (K)
20200314094244
Thermopower (μV/K)
Conductance (G0)

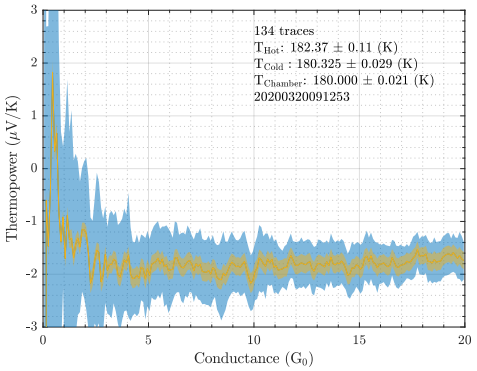
# A.3. T0144S

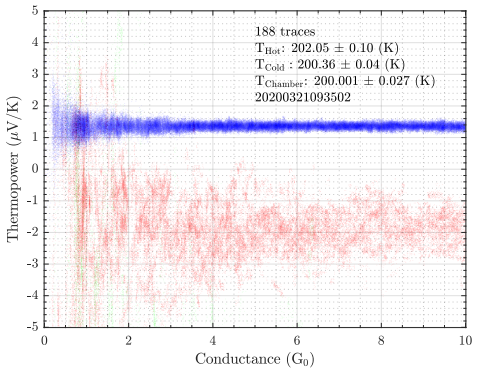
Sample T0144S was measured from 160 to 295 K with the "Thermospannungsmessung_-statisch_V2" software (see appendix E.41 on page 357). This sample was broken in vacuum before the measurements to verify that the sample worked as expected.

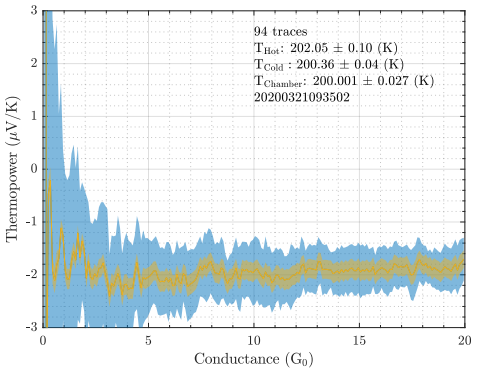


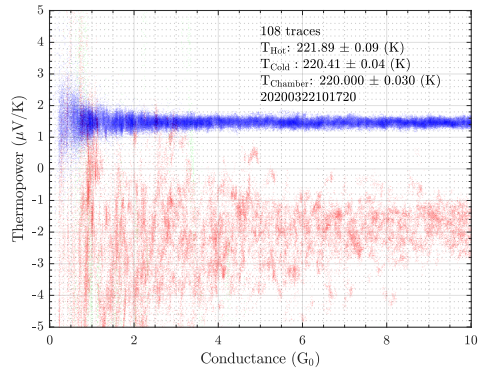



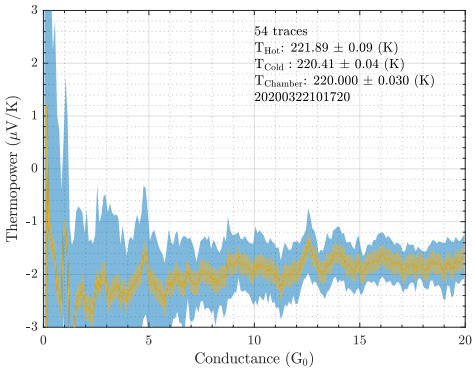



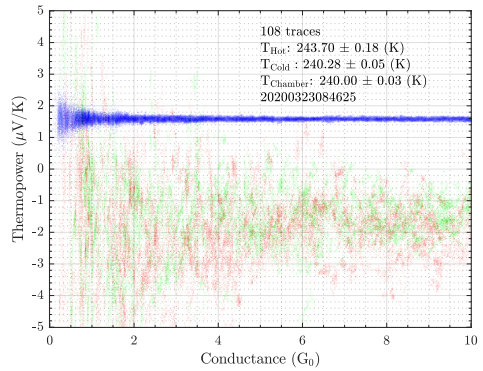



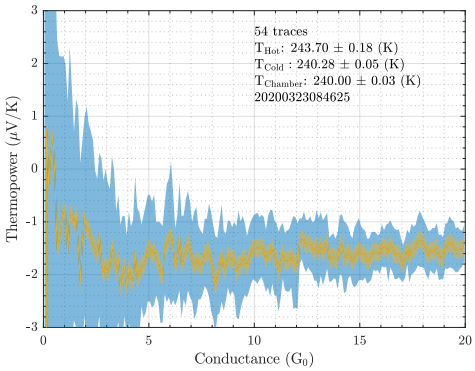



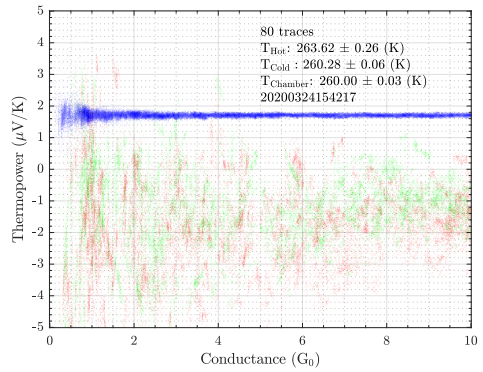



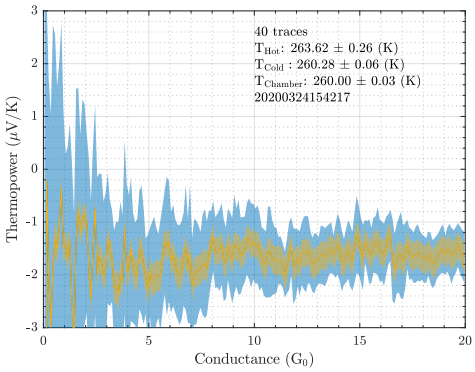

108 traces
$T_{Hot}$: 221.89 ± 0.09 (K)
$T_{Cold}$ : 220.41 ± 0.04 (K)
$T_{Chamber}$: 220.000 ± 0.030 (K)
20200322101720
Thermopower (μV/K)
Conductance ($G_0$)

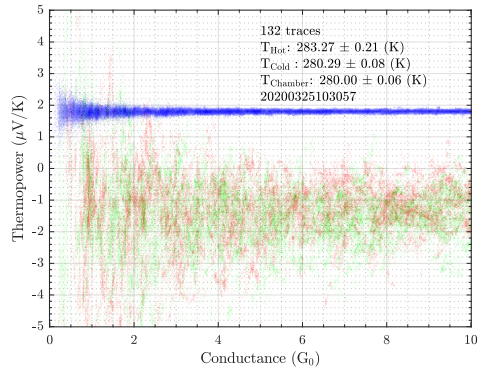

54 traces
$T_{Hot}$: 221.89 ± 0.09 (K)
$T_{Cold}$ : 220.41 ± 0.04 (K)
$T_{Chamber}$: 220.000 ± 0.030 (K)
20200322101720
Thermopower (μV/K)
Conductance ($G_0$)

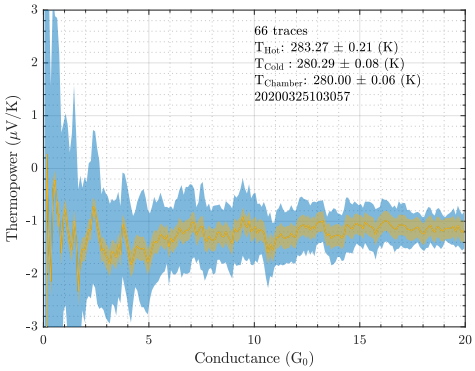

108 traces
$T_{Hot}$: 243.70 ± 0.18 (K)
$T_{Cold}$ : 240.28 ± 0.05 (K)
$T_{Chamber}$: 240.00 ± 0.03 (K)
20200323084625
Thermopower (μV/K)
Conductance ($G_0$)

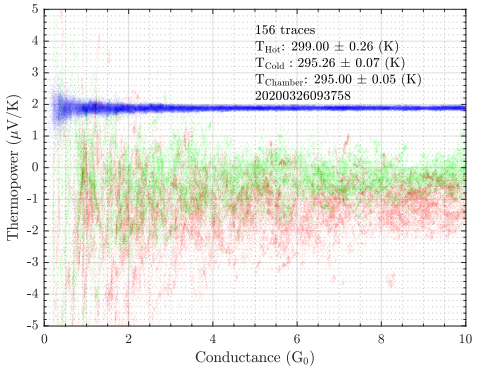

54 traces
$T_{Hot}$: 243.70 ± 0.18 (K)
$T_{Cold}$ : 240.28 ± 0.05 (K)
$T_{Chamber}$: 240.00 ± 0.03 (K)
20200323084625
Thermopower (μV/K)
Conductance ($G_0$)

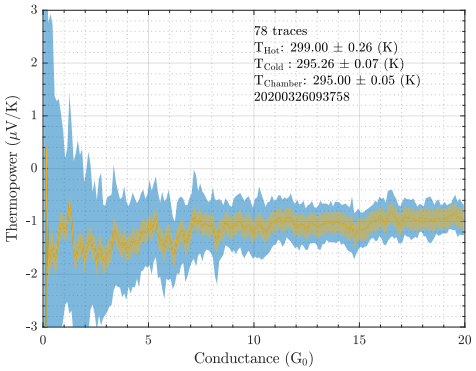

80 traces
$T_{Hot}$: 263.62 ± 0.26 (K)
$T_{Cold}$ : 260.28 ± 0.06 (K)
$T_{Chamber}$: 260.00 ± 0.03 (K)
20200324154217
Thermopower (μV/K)
Conductance ($G_0$)


40 traces
$T_{Hot}$: 263.62 ± 0.26 (K)
$T_{Cold}$ : 260.28 ± 0.06 (K)
$T_{Chamber}$: 260.00 ± 0.03 (K)
20200324154217
Thermopower (μV/K)
Conductance ($G_0$)


132 traces
$T_{Hot}$: 283.27 ± 0.21 (K)
$T_{Cold}$ : 280.29 ± 0.08 (K)
$T_{Chamber}$: 280.00 ± 0.06 (K)
20200325103057
Thermopower (μV/K)
Conductance ($G_0$)


66 traces
$T_{Hot}$: 283.27 ± 0.21 (K)
$T_{Cold}$ : 280.29 ± 0.08 (K)
$T_{Chamber}$: 280.00 ± 0.06 (K)
20200325103057
Thermopower (μV/K)
Conductance ($G_0$)


156 traces
T_Hot: 299.00 ± 0.26 (K)
T_Cold : 295.26 ± 0.07 (K)
T_Chamber: 295.00 ± 0.05 (K)
20200326093758
Thermopower (μV/K)
Conductance (G0)


78 traces
T_Hot: 299.00 ± 0.26 (K)
T_Cold : 295.26 ± 0.07 (K)
T_Chamber: 295.00 ± 0.05 (K)
20200326093758
Thermopower (μV/K)
Conductance (G0)

# B. Wiring

The setup is separated in different electronics groups. There are an adapter or even electronics inside the sample chamber which are connected to the IceCube via shielded pairs with individually sleeved wires. The eight measurement wires are connected to the back plane of the IceCube according to the schematic shown in figure B.1. The measurement pairs are numbered from 1 to 4.
The IceCube is connected to the NIM crate modules with different cables depending on which IceCube inserts are used. These connections are realized with LEMO connectors. The different keying possibilities for these plugs ensure that no mistakes can be made by plugging in the wrong wires.
For the pulsed measurements, the eight wires are connected to the sample in such a way, that four wire measurements can be performed on every connection. The necessary adapter to the sample connector with the for pins A to D is shown in figure B.2 and the schematic in figure B.3.

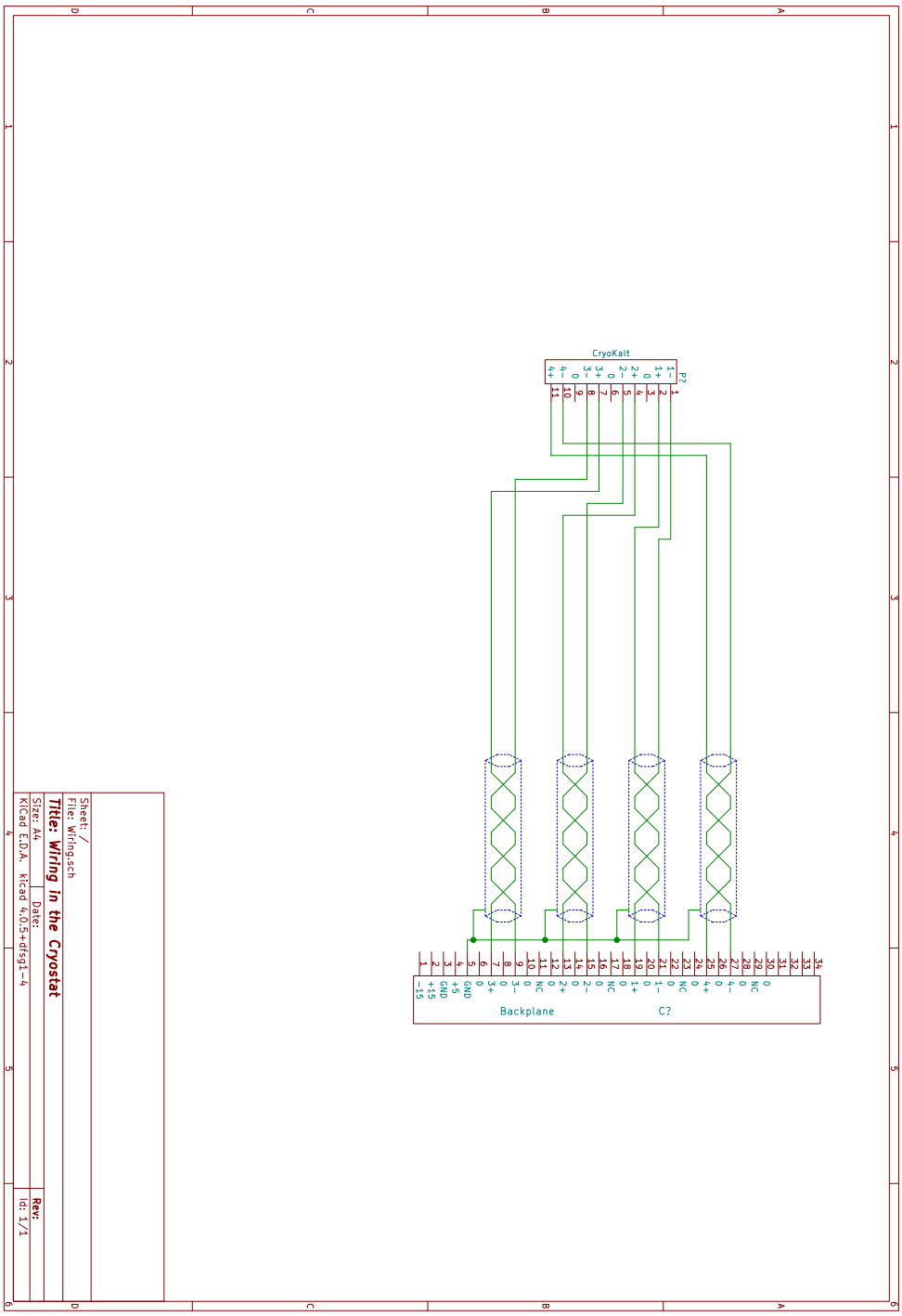



**Figure B.1.:** Schematic of the measurement wiring in the cryogenic insert

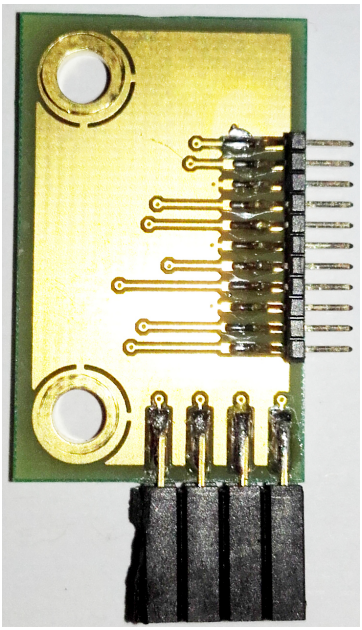


**Figure B.2.:** Photo of the sample chamber wiring adapter for the pulsed measurements

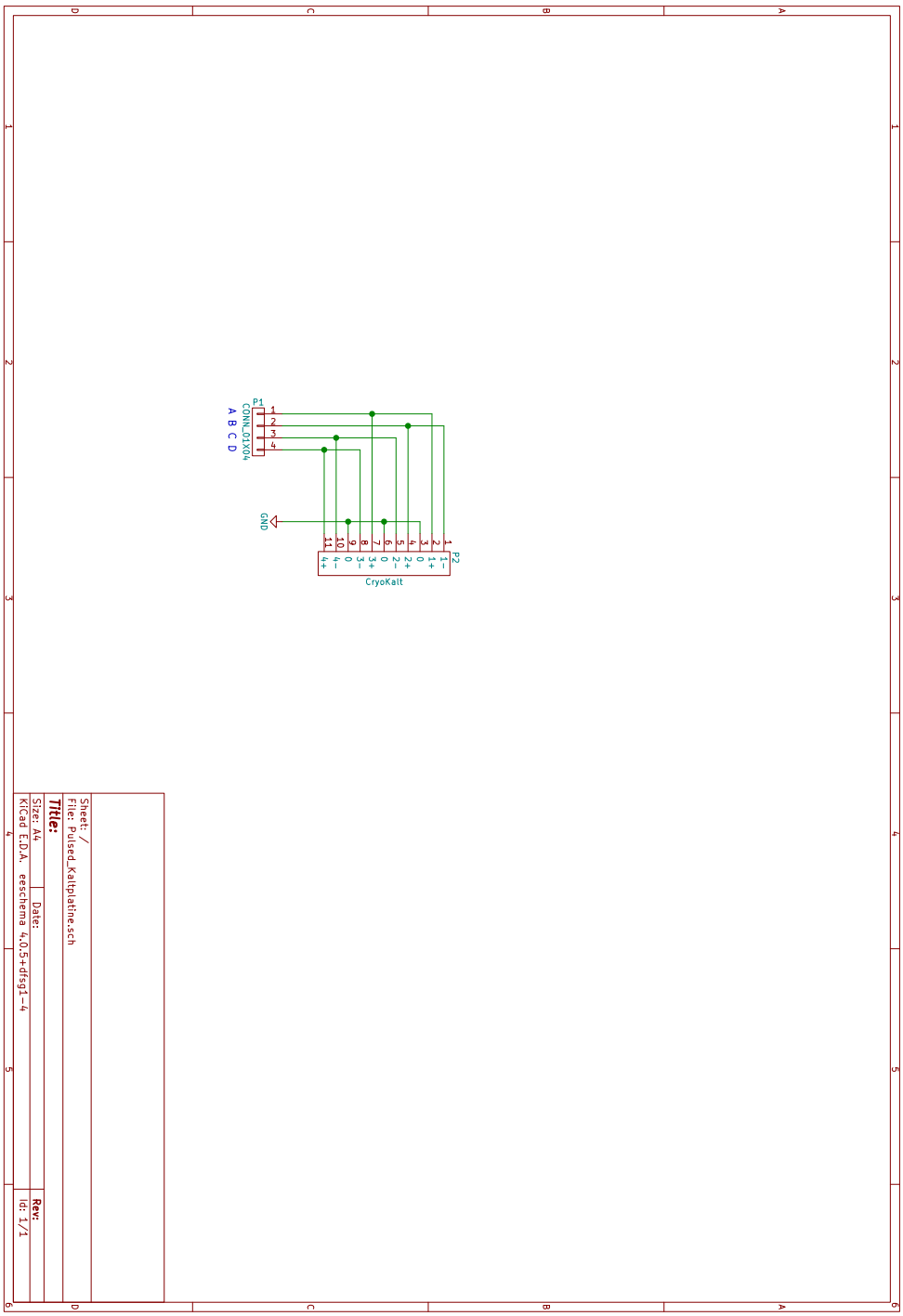



**Figure B.3.:** Schematic of the wiring adapter that converts the four sample connections A to D to the eight measurement wires

# C. IceCube inserts

## C.1. Power Supply

This insert is used to provide power to the other IceCube inserts via the backplane. The insert is powered by the symmetric 24 V supply of the NIM-crate. It creates symmetric 15 V rails and a single 5 V supply, using linear regulators. This voltage regulation is used to reduce any remaining noise in the power supply. An integrated thermometer can be used for monitoring, and the current provided on the 15 V rails is measured for debugging purposes as well. However, these features do not work currently.
A block diagram is shown in figure C.1 and the full schematic is shown in figure C.2. The circuit board layout is shown in figure C.3 and the pin configuration is shown in figure C.4. A photo of the complete insert is shown in figure C.5.

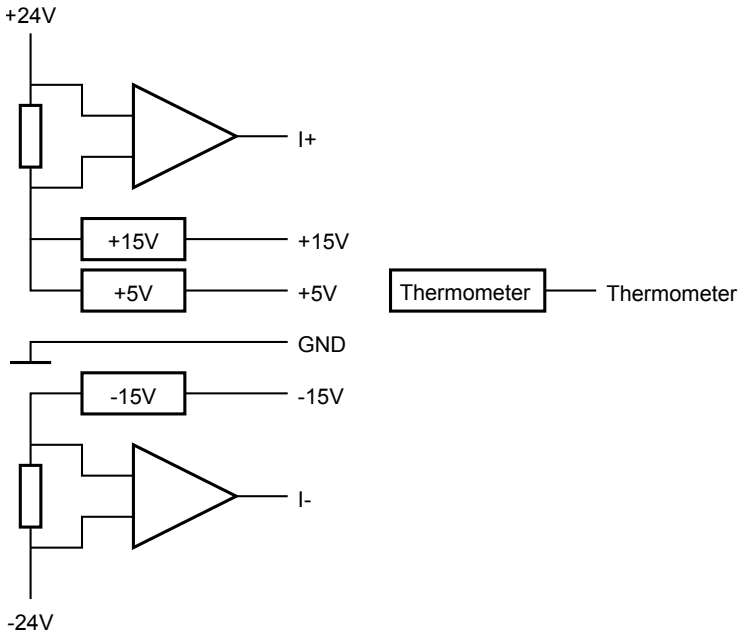



**Figure C.1.:** Block diagram of the power supply insert

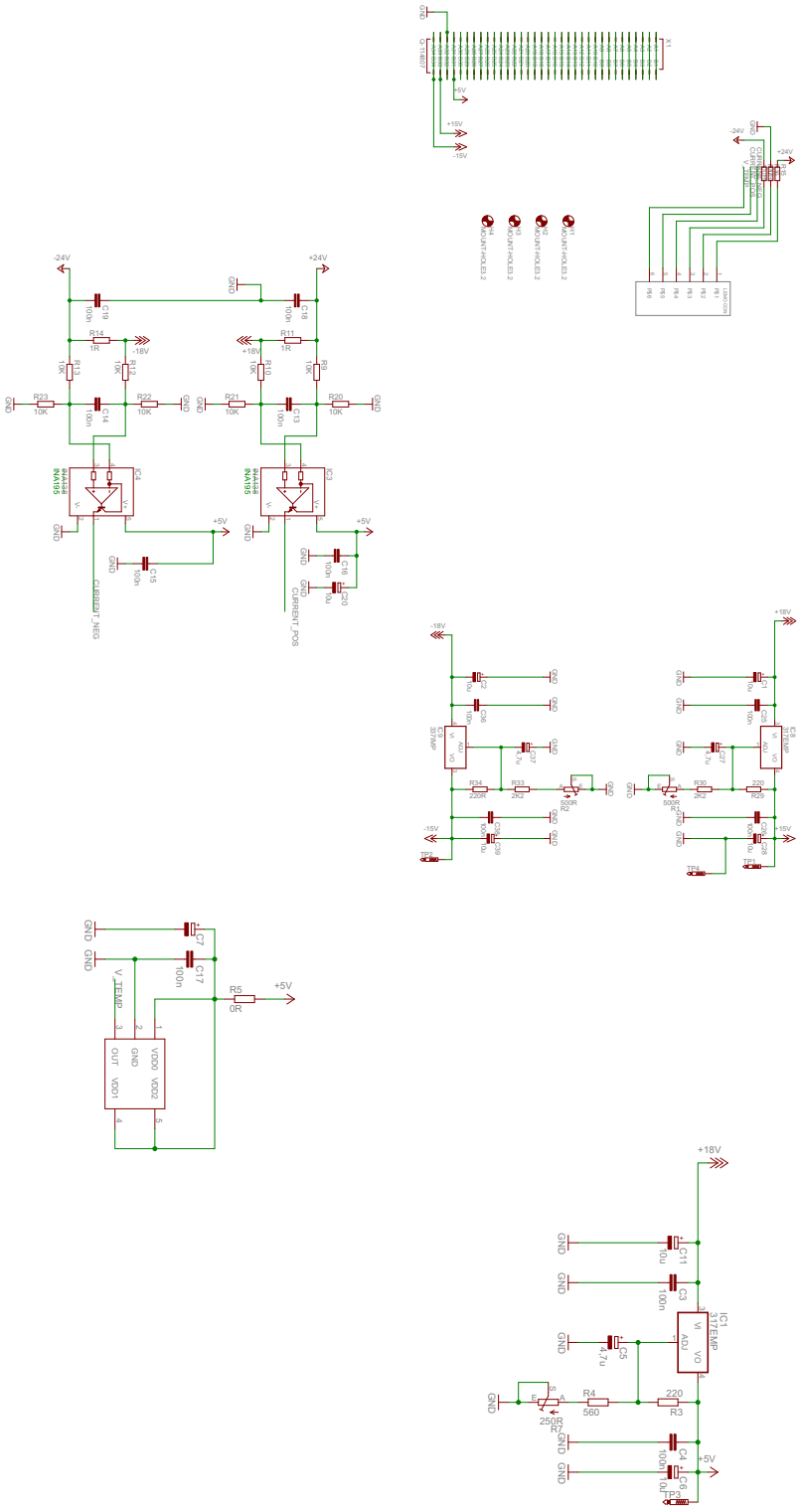
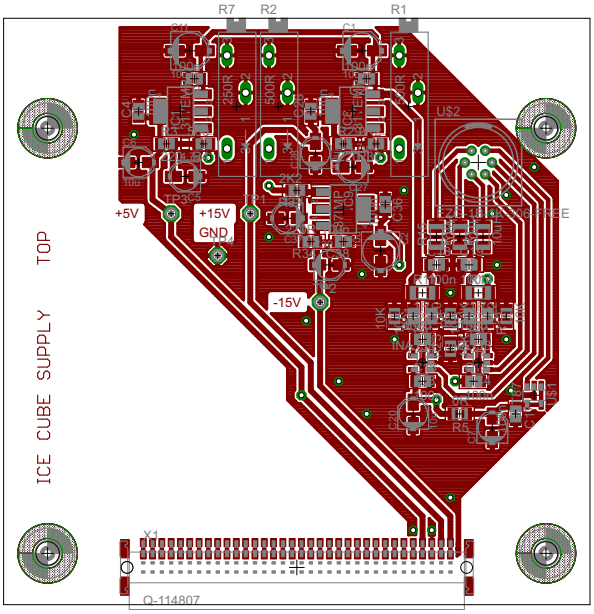
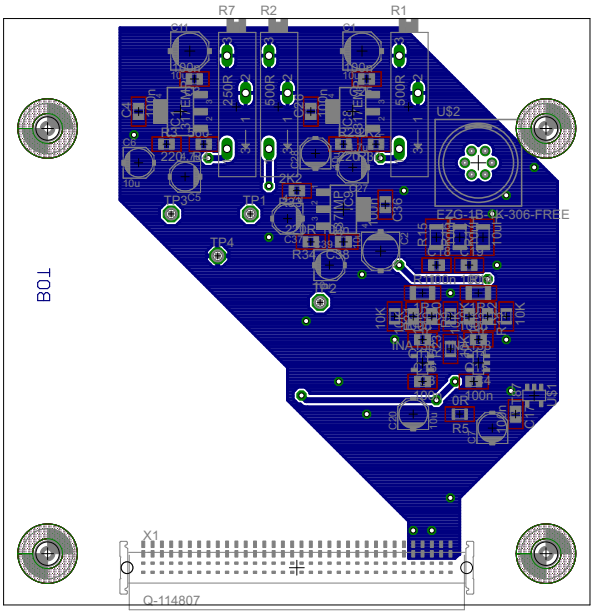


**Figure C.2.:** Complete schematic of the power supply IceCube insert

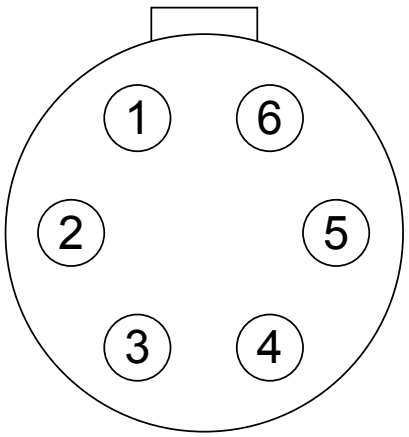



**(a)** Top side copper structure




**(b)** Bottom side copper structure

**Figure C.3.:** Circuit board layout of the power supply IceCube insert




| | | |
|---|---|---|
| 1 | I- | output |
| 2 | I+ | output |
| 3 | V thermometer | output |
| 4 | 24 V | input |
| 5 | GND | input |
| 6 | −24 V | input |

**Figure C.4.:** Pin configuration of the front side connector of the power supply IceCube insert as seen from the outside

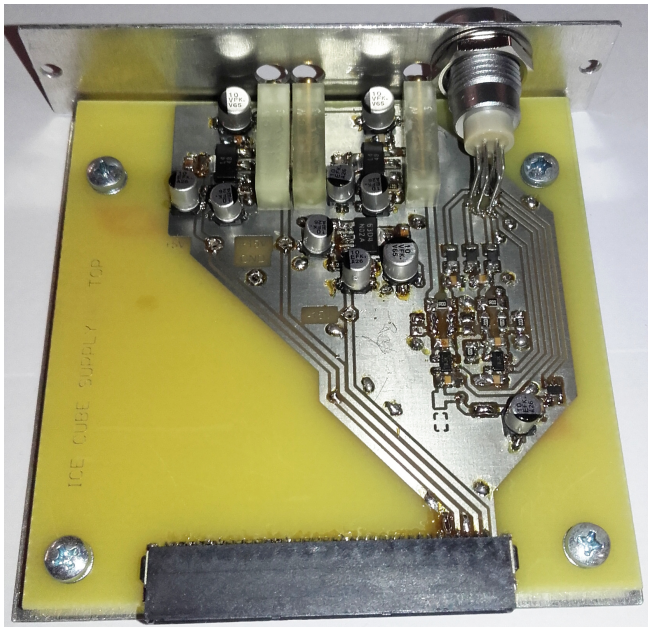


**Figure C.5.:** Photo of the power supply IceCube insert

## C.2. Sense

This insert is used to amplify the small measurements signals. The amplification is performed by an instrumentation amplifier to avoid any ground loops. A Sallen-Key filter is used to limit the bandwidth to 10 kHz in order to cut away any high frequency noise. Four equivalent circuits are used, one for each measurement pair.
The sense insert is also used during the molecule measurements, however it is rewired as seen in figure C.13. Here, the inputs are disconnected and only one stage is used to measure the bias voltage for the sample. Because of the existing wiring scheme, the voltage cannot be measured across a single pair. Thus the inputs are distributed manually via a jumper wire.
A block diagram is shown in figure C.6 and the full schematic is shown in figures C.7 to C.9. The circuit board layout is shown in figure C.10 and the pin configuration is shown in figure C.11. A photo of the complete insert is shown in figure C.12.

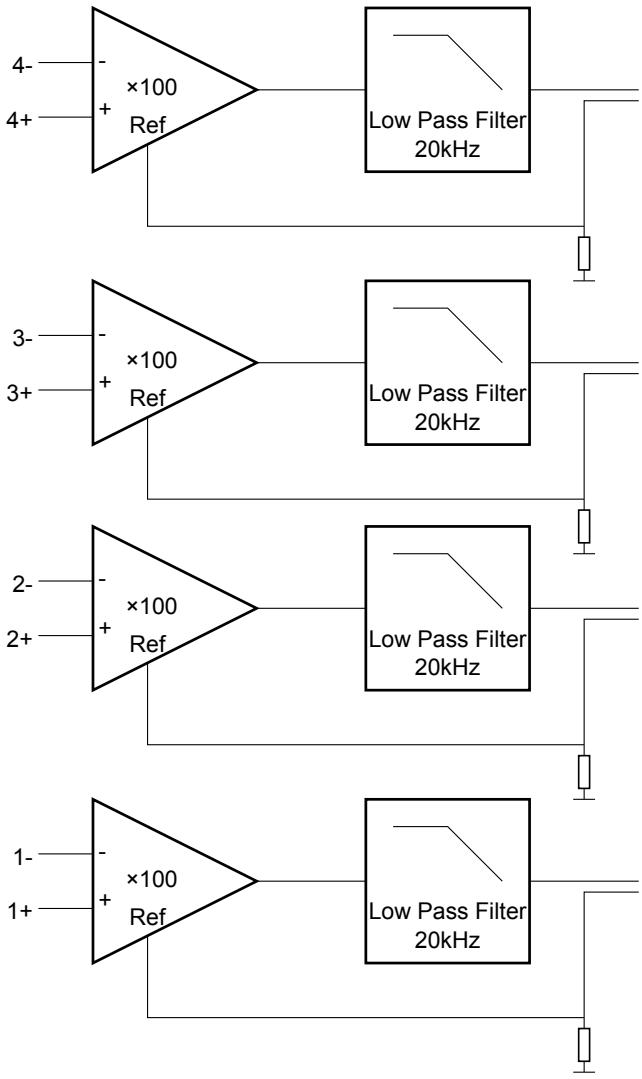



**Figure C.6.:** Block diagram of the sense insert

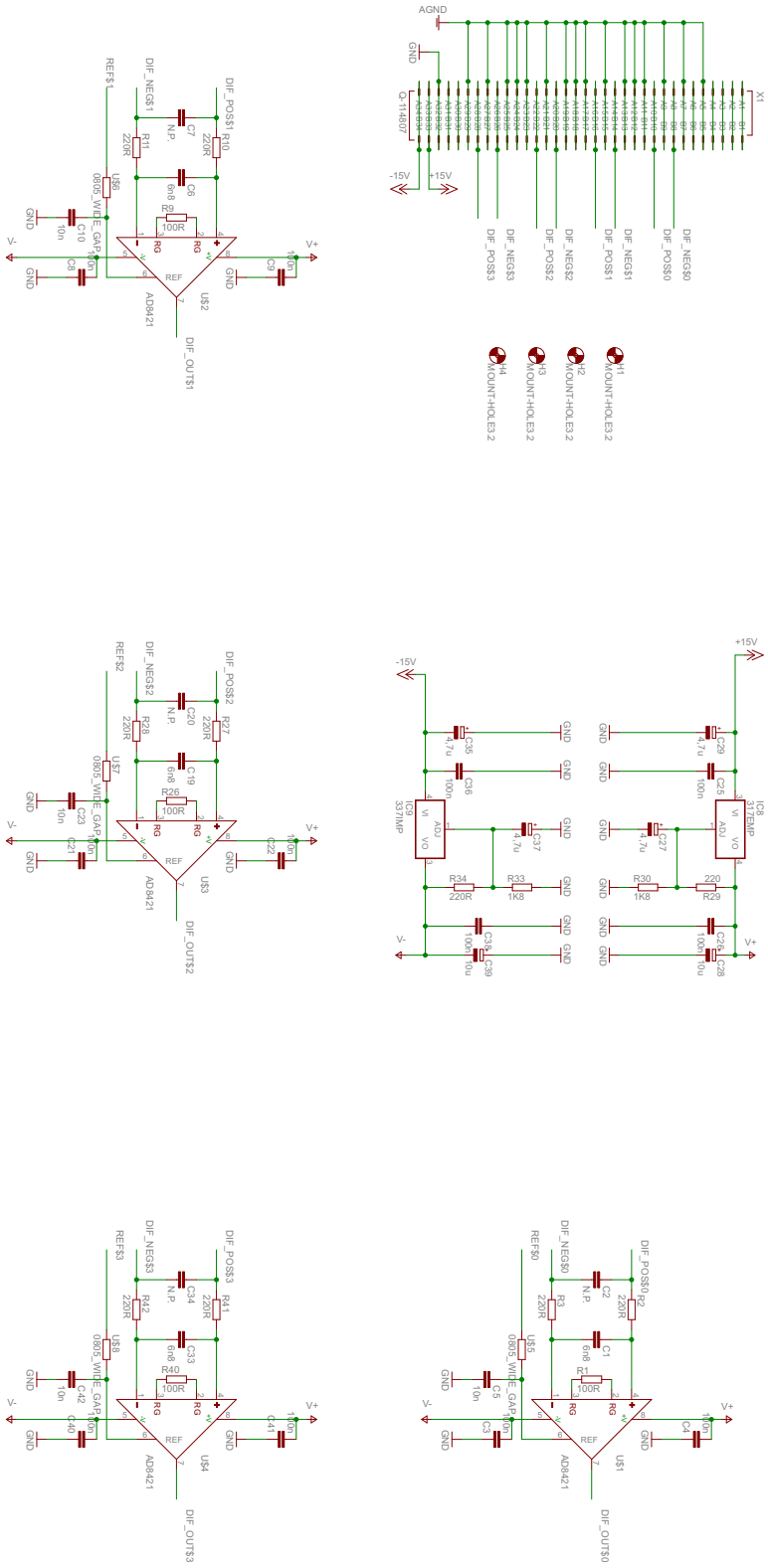



**Figure C.7.:** Complete schematic of the sense IceCube insert

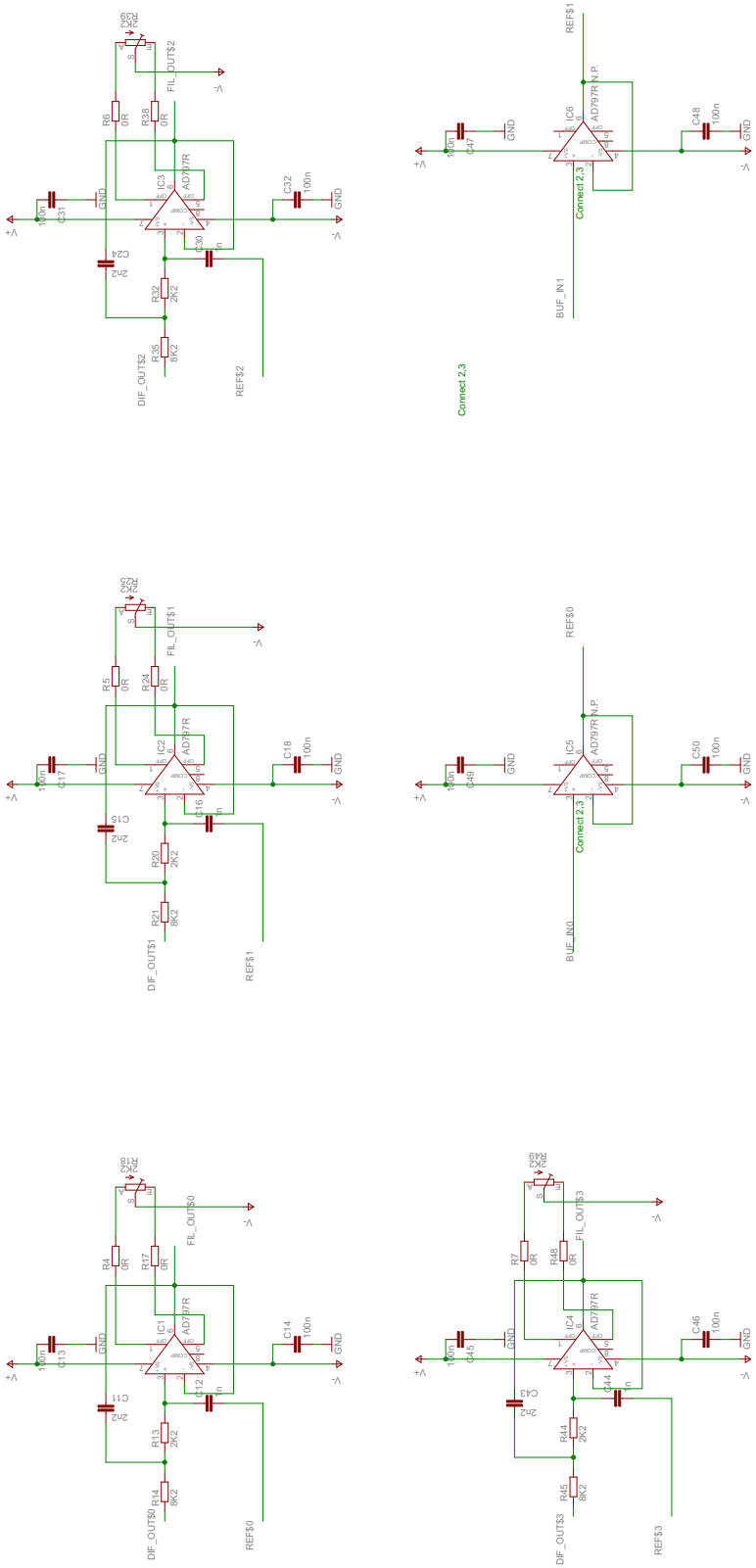



**Figure C.8.:** Complete schematic of the sense IceCube insert

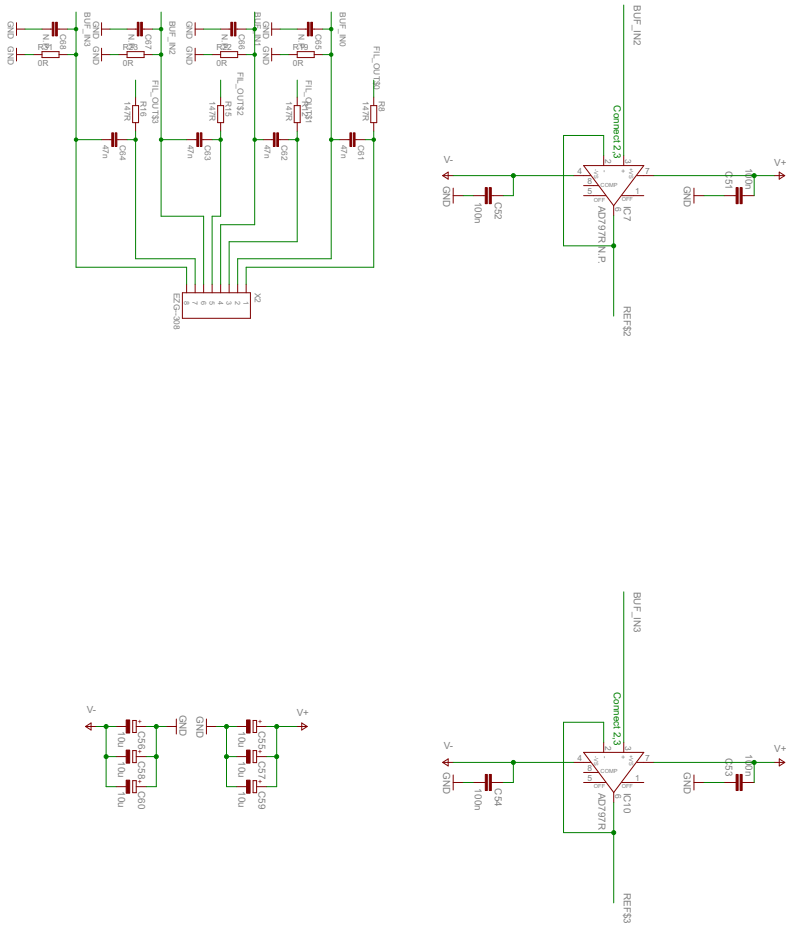
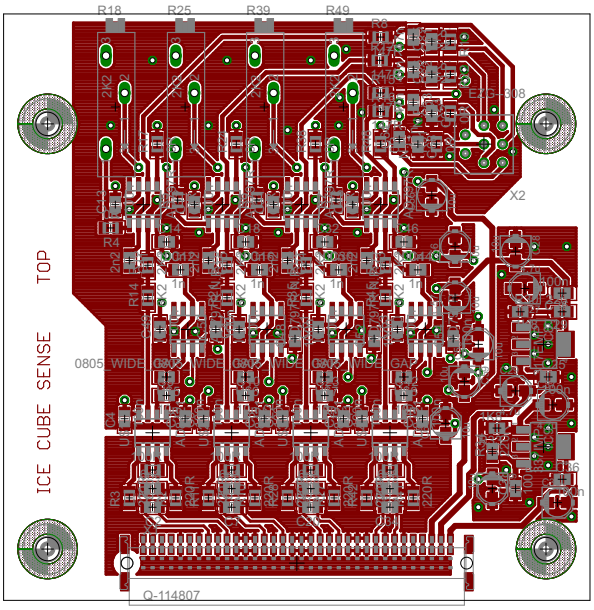
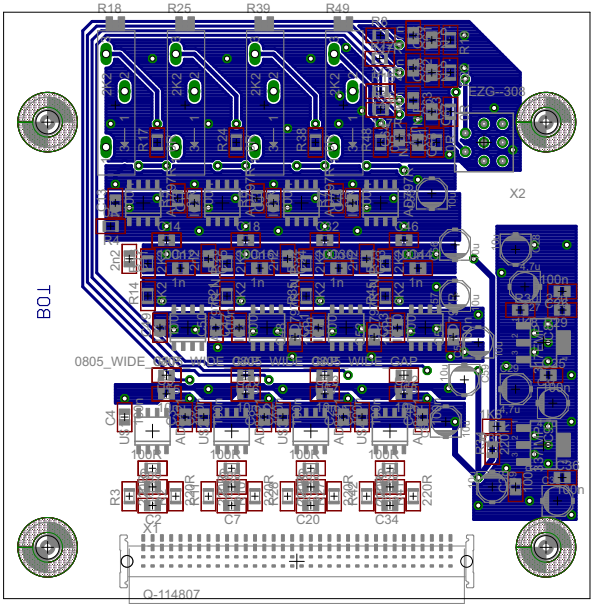



**Figure C.9.:** Complete schematic of the sense IceCube insert

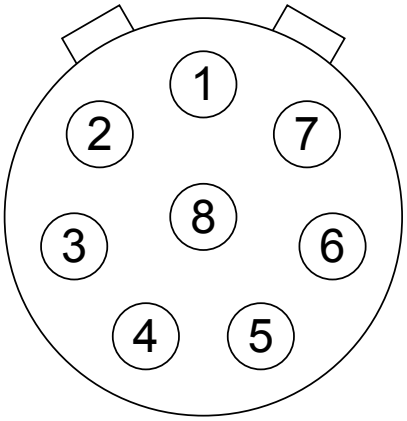



**(a)** Top side copper structure




**(b)** Bottom side copper structure

**Figure C.10.:** Circuit board layout of the sense IceCube insert




| | | |
|---|---|---|
| 1 | (2−) − (2+) reference | output |
| 2 | (3−) − (3+) reference | output |
| 3 | (4−) − (4+) signal | output |
| 4 | (4−) − (4+) reference | output |
| 5 | (1−) − (1+) signal | output |
| 6 | (1−) − (1+) reference | output |
| 7 | (2−) − (2+) signal | output |
| 8 | (3−) − (3+) signal | output |

**Figure C.11.:** Pin configuration of the front side connector of the sense IceCube insert as seen from the outside

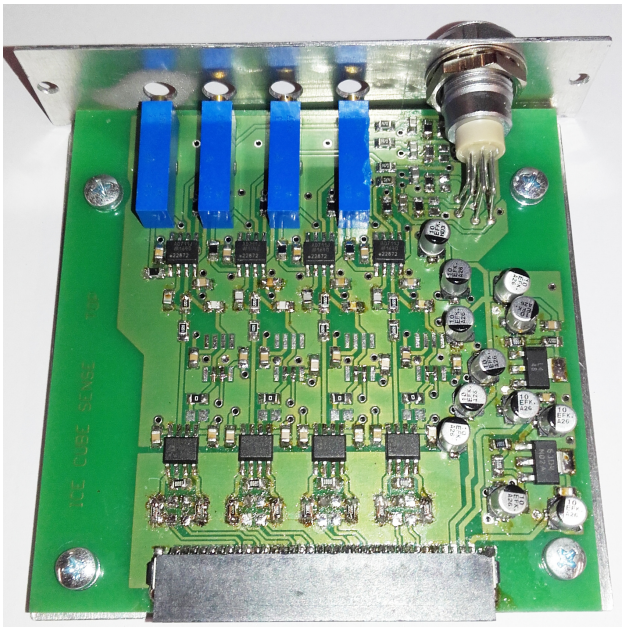


**Figure C.12.:** Photo of the sense IceCube insert

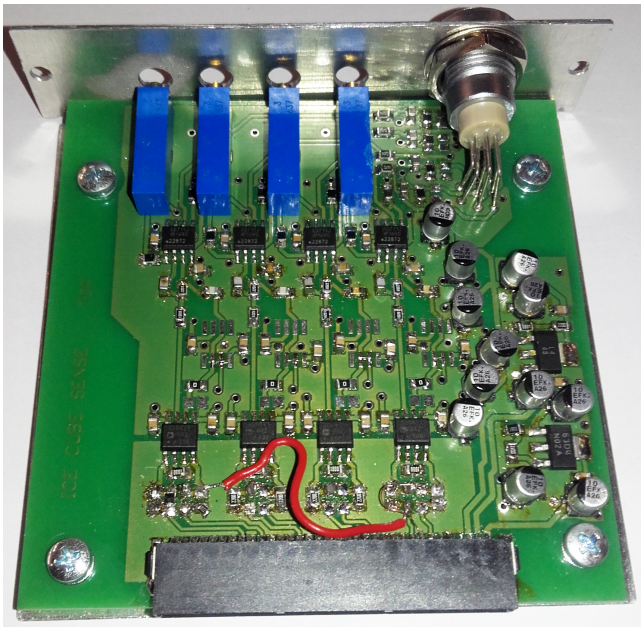


**Figure C.13.:** Photo of the sense IceCube insert in the rewired configuration for molecule measurements

## C.3. ΔR Source

This insert has two voltage sources with a 10 kΩ series resistor each. The voltage drop across the resistor is measured by a differential amplifier. The voltage sources can be configured to use the same or opposite polarities and their output can be referenced to any potential. This decouples the ground of the external source from the reference potential used for the outputs. The same ground decoupling is performed with the current signals. The differential amplifiers allow for an externally driven reference level.
In the insert, the two voltage sources are wired in a way that they produce the same polarity and they are referenced to the ground of the cryogenic insert.
A block diagram is shown in figure C.14 and the full schematic is shown in figures C.15 to C.17. The circuit board layout is shown in figure C.18 and the pin configuration is shown in figure C.19. A photo of the complete insert is shown in figure C.20.

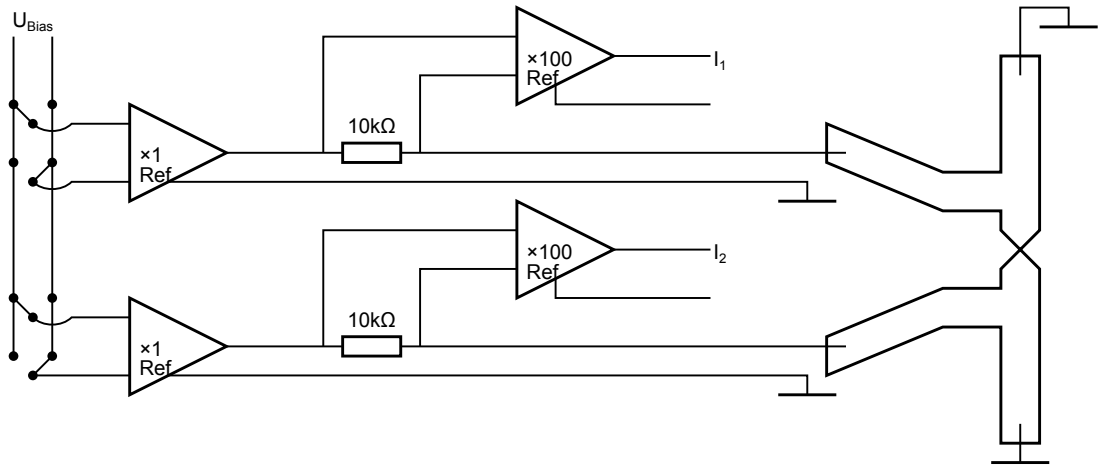



**Figure C.14.:** Block diagram of the ΔR source insert

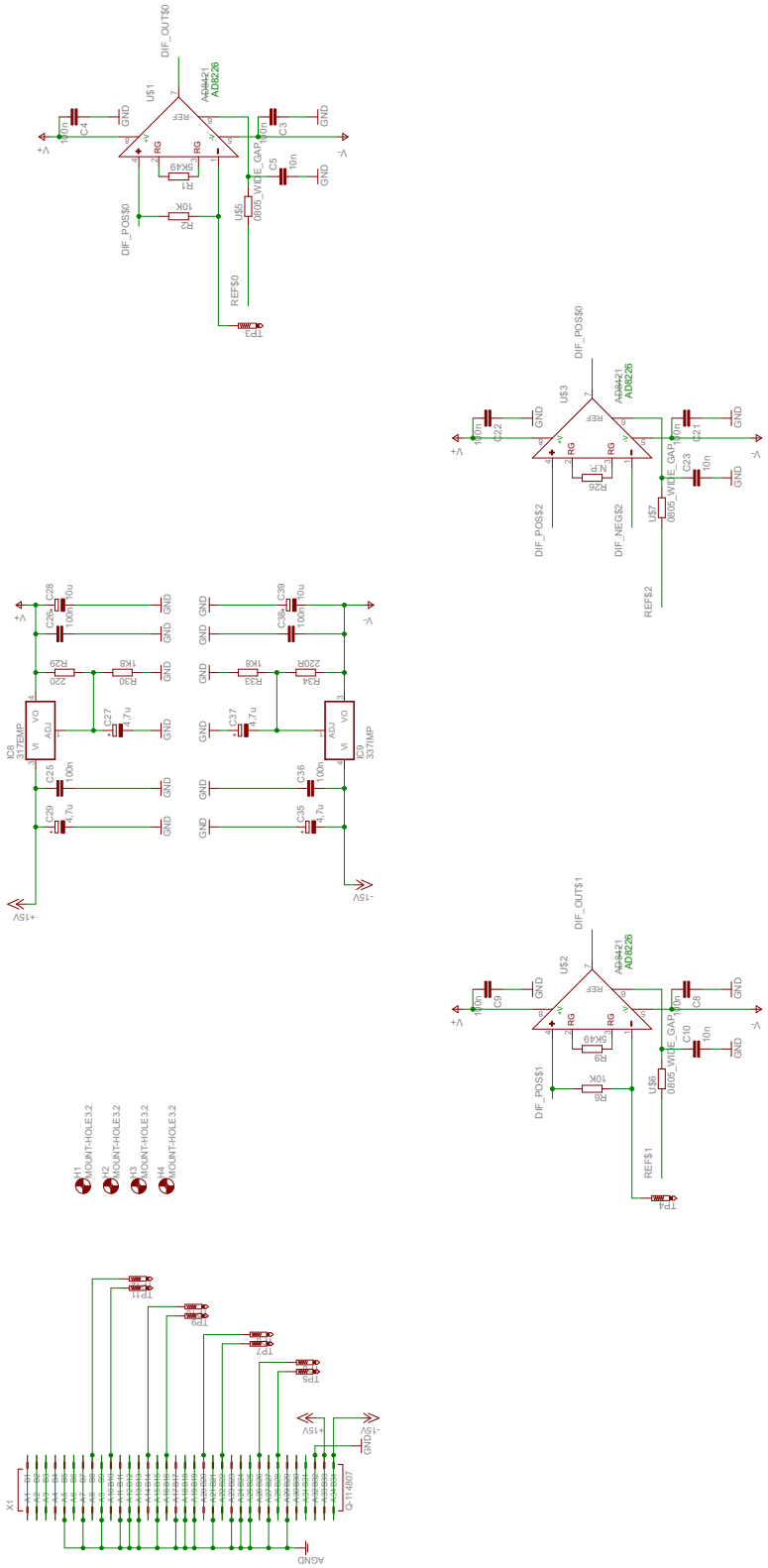



**Figure C.15.:** Complete schematic of the ΔR source IceCube insert

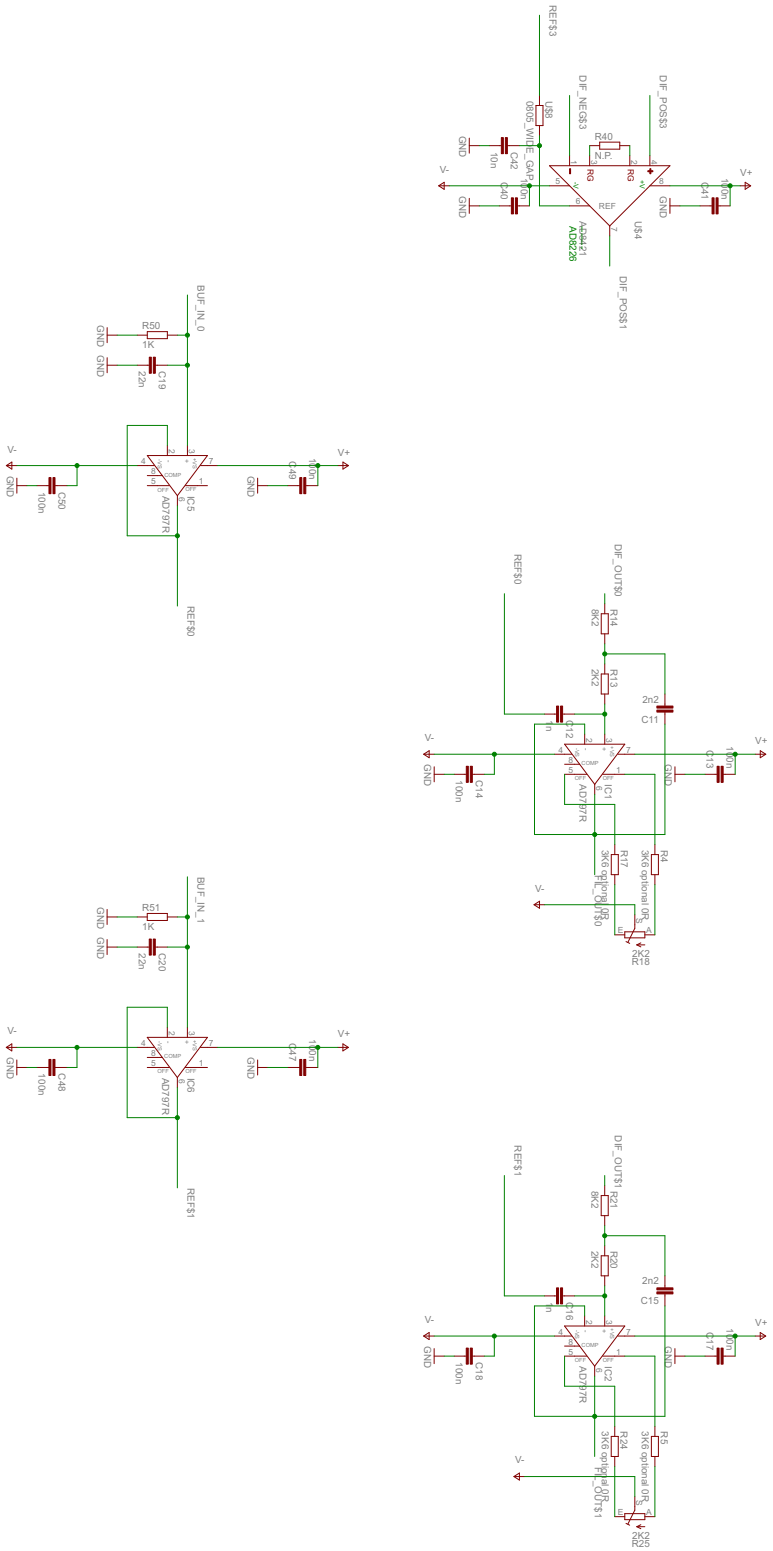



**Figure C.16.:** Complete schematic of the $\Delta$R source IceCube insert

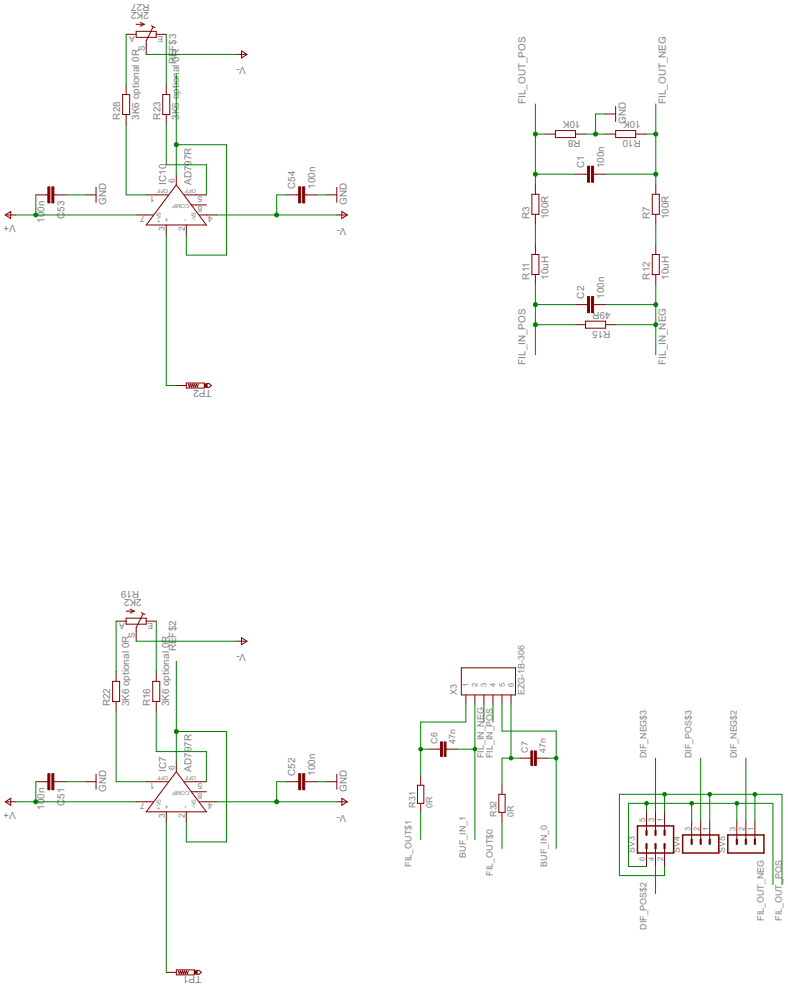
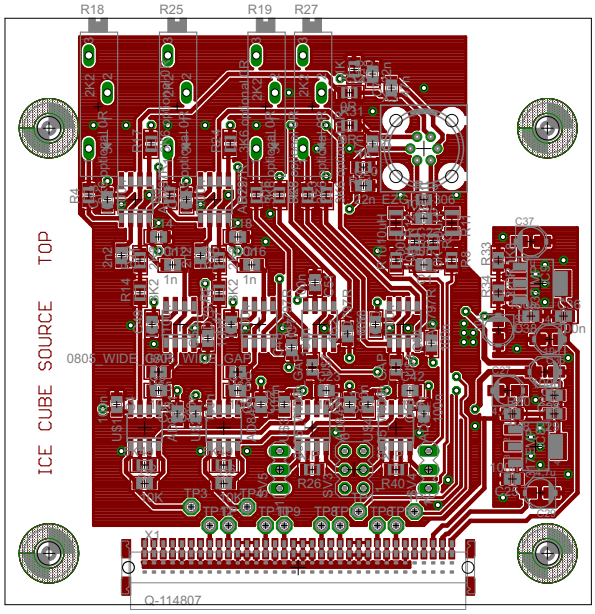
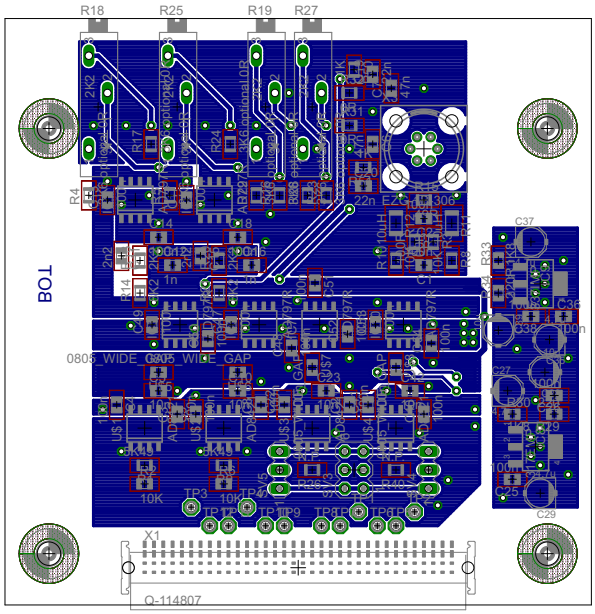



**Figure C.17.:** Complete schematic of the ΔR source IceCube insert

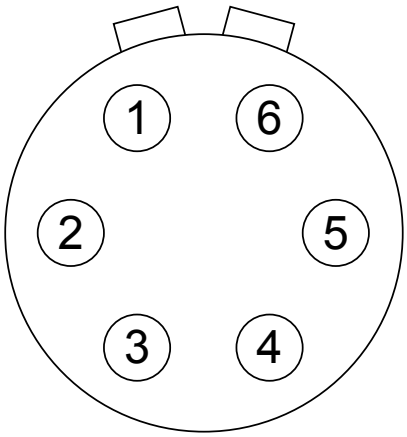



**(a)** Top side copper structure




**(b)** Bottom side copper structure

**Figure C.18.:** Circuit board layout of the ΔR source IceCube insert




| | | |
|---|---|---|
| 1 | V reference | input |
| 2 | $I_2$ reference | output |
| 3 | $I_2$ signal | output |
| 4 | $I_1$ signal | output |
| 5 | $I_1$ reference | output |
| 6 | V signal | input |

**Figure C.19.:** Pin configuration of the front side connector of the ΔR source IceCube insert as seen from the outside

**Figure C.20.:** Photo of the ΔR source IceCube insert

## C.4. ΔV Source

This insert uses identical circuitry to the ΔR source insert. However, the voltage sources are set up to provide opposite polarities for a symmetric bias of the sample.
A block diagram is shown in figure C.21. The full schematic, the pin configuration, and the circuit board layout are identical to those shown in figures C.15 to C.19. A photo of the complete insert is shown in figure C.22.

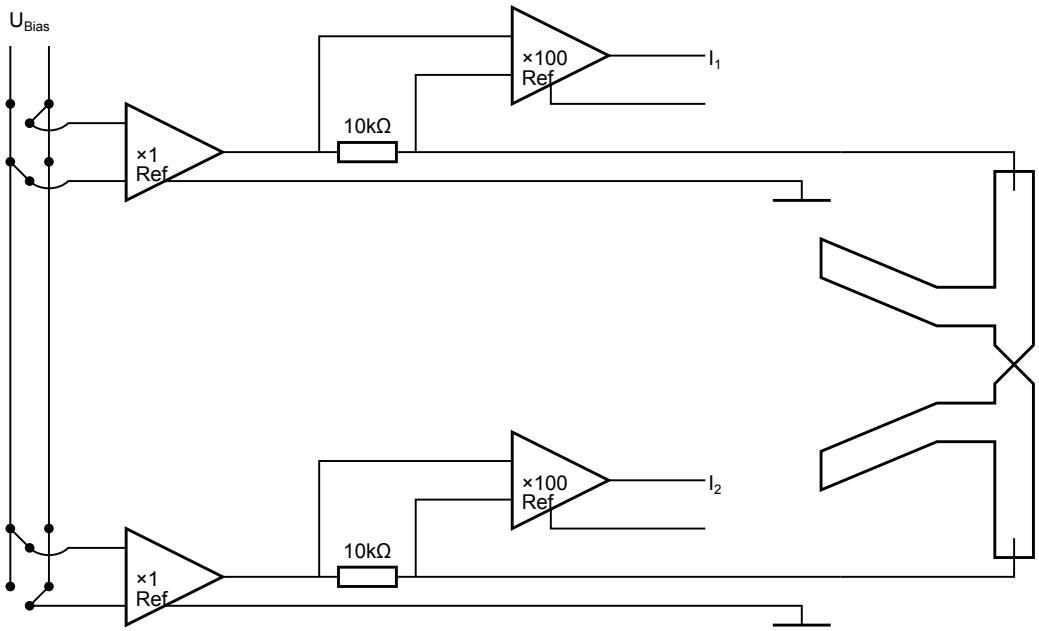



**Figure C.21.:** Block diagram of the ΔV source insert

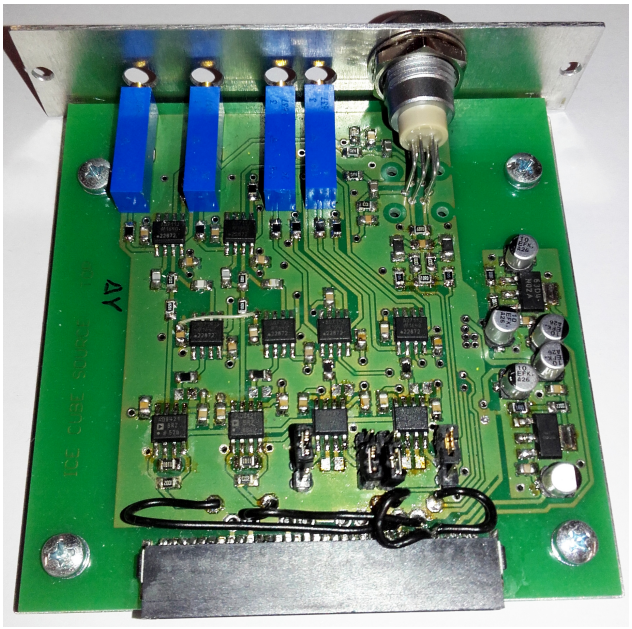



**Figure C.22.:** Photo of the $\Delta$V source IceCube insert

## C.5. IV-Converter for Metallic Contacts

For the static measurements on metallic contacts, an IV-converter is used to measure the current across the sample. A bias voltage is applied over a cold 10 kΩ resistor, and the voltage across the sample is measured with the "sense" insert. This allows to perform a four wire measurement. Additionally, the platinum resistance on the sample is measured in a four-wire configuration as well. Except for the voltage measurement, this insert provides everything which is needed:
The bias voltage is divided and goes through a differential amplifier to perform the level shifting to the ground potential of the cryogenic insert. The current is detected by the IV-converter with a transimpedance of $1 \cdot 10^6\,\mathrm{V\,A^{-1}}$. The output of the amplifier is turned into a symmetric signal for the output. The four wires to contact the resistance thermometer are presented with the identical plug and pinout used in the "breakout" insert. Thus the resistance can be measured with an external Keithley 2400 SourceMeter$^{\mathrm{TM}}$.
A block diagram is shown in figure C.23 and the full schematic for both the insert and the circuit in the sample chamber are shown in figures C.24 and C.25. The circuit board layouts are shown in figures C.26 and C.27 and the pin configuration is shown in figure C.28. Photos of the complete insert and the sample chamber electronics are shown in figure C.29.

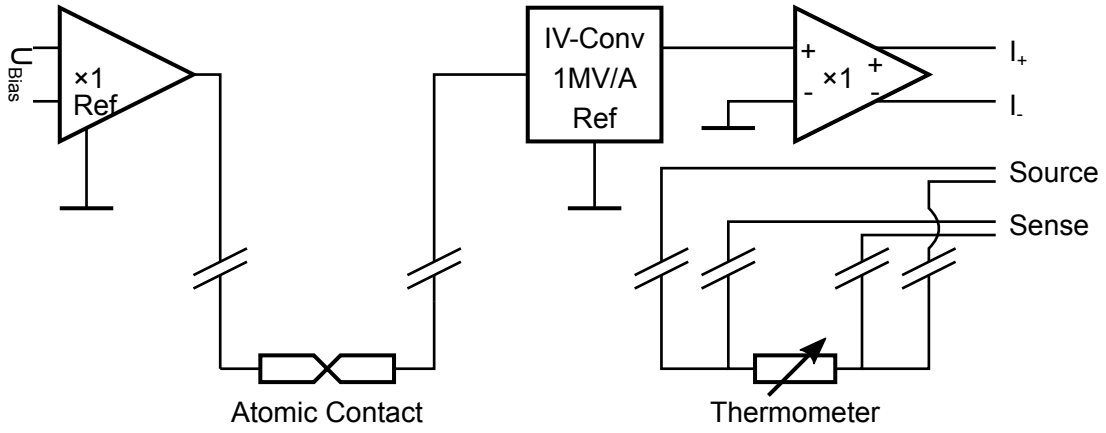



**Figure C.23.:** Block diagram of the cold IV converter

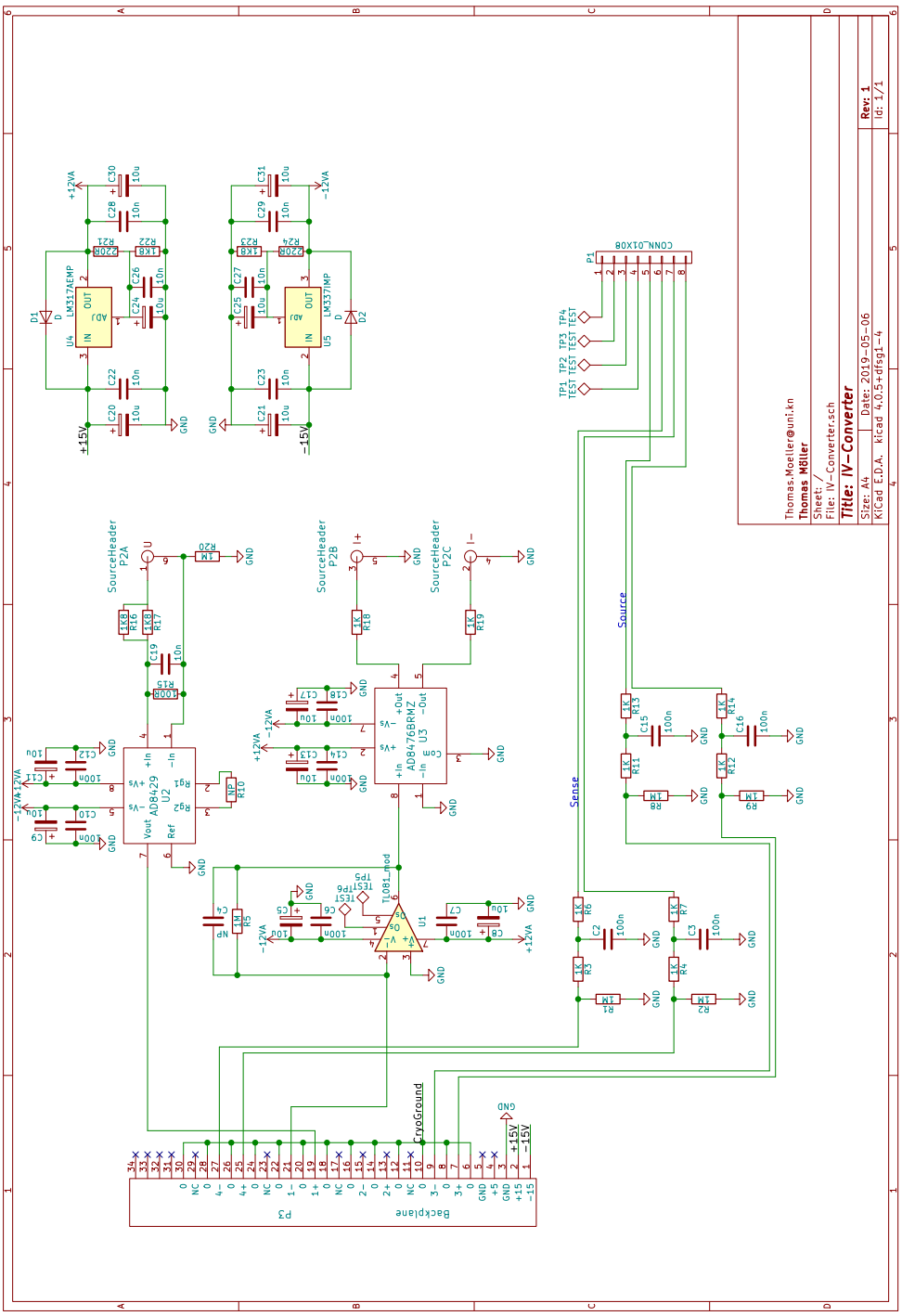



**Figure C.24.:** Complete schematic of the IV-converter IceCube insert for metallic contacts

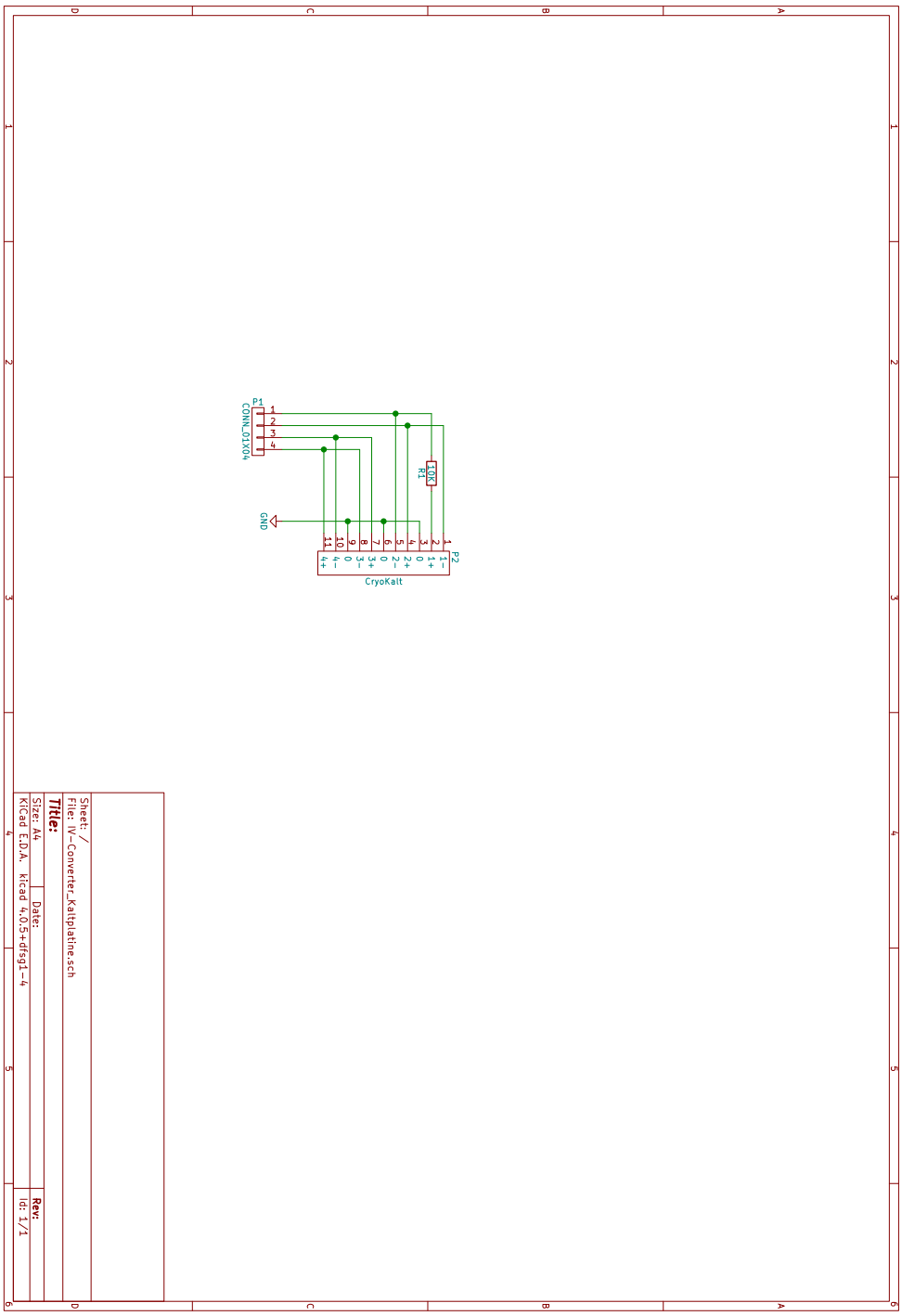
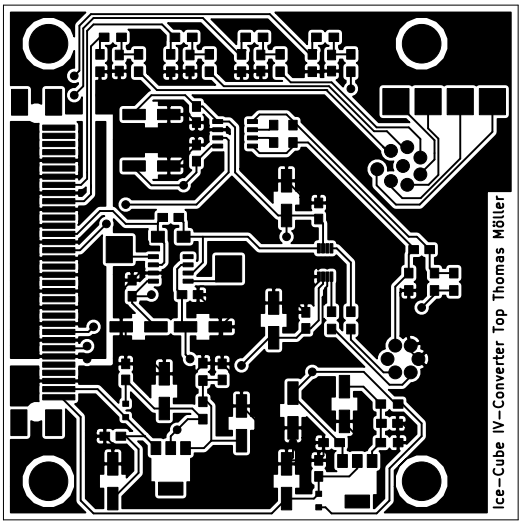
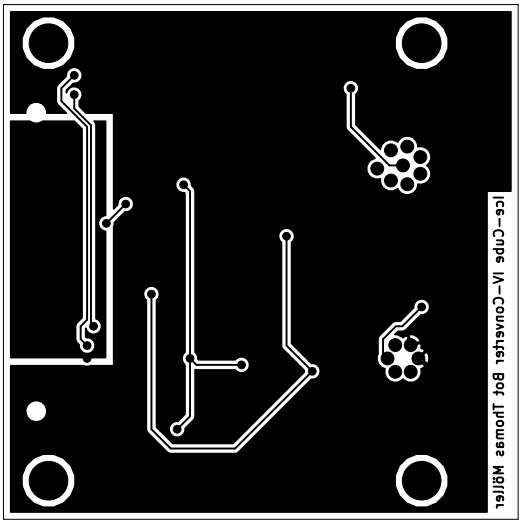


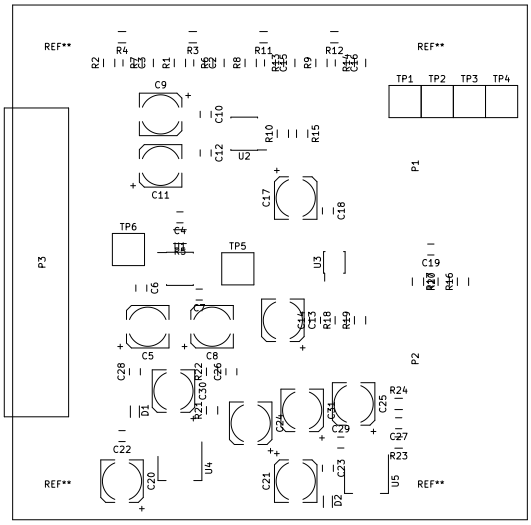

**Figure C.25.:** Complete schematic of the cold circuit for the IV-converter for metallic contacts




**(a)** Top side copper structure

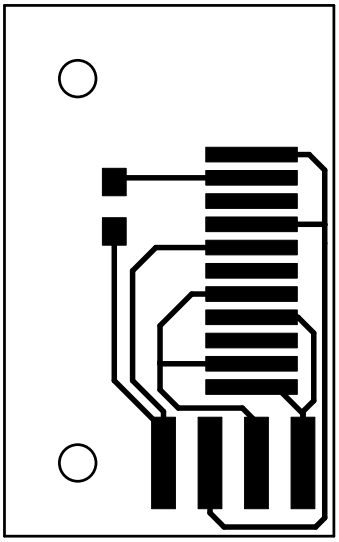



**(b)** Bottom side copper structure

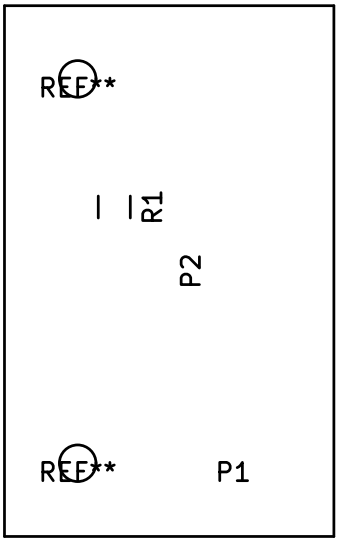



**(c)** Component references

**Figure C.26.:** Circuit board layout of the IV-converter insert for metallic contacts

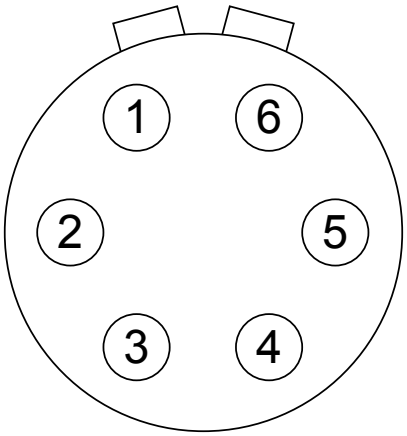


**(a)** Top side copper structure

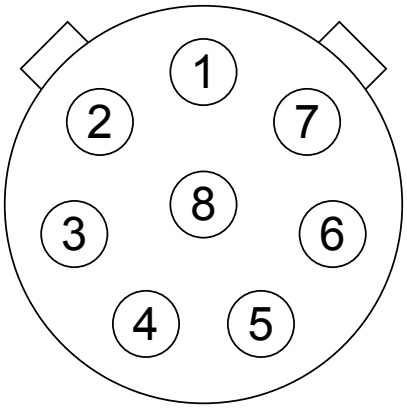



**(b)** Component references

**Figure C.27.:** Circuit board layout for the matching electronics in the sample chamber



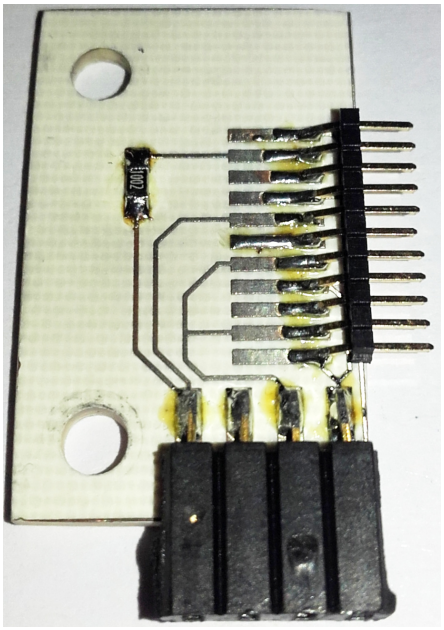

| | | |
|---|---|---|
| 1 | Bias voltage reference | input |
| 2 | not connected | |
| 3 | IV signal positive | output |
| 4 | IV signal negative | output |
| 5 | not connected | |
| 6 | Bias voltage signal | input |




| | | |
|---|---|---|
| 1 | not connected | |
| 2 | not connected | |
| 3 | not connected | |
| 4 | not connected | |
| 5 | 3− | input/output |
| 6 | 4− | input/output |
| 7 | 4+ | input/output |
| 8 | 3+ | input/output |

**Figure C.28.:** Pin configuration of the IV-converter insert as seen for the outside

**(a)** IceCube insert



**(b)** PCB inside the sample chamber

**Figure C.29.:** Photos of the IV-converter for metallic contacts and the corresponding electronics in the sample chamber

## C.6. IV-Converter with Cold Front-End

Since molecules have significantly higher resistances than the atomic contacts, the parasitic capacitances of the measurement wiring have to be avoided. This is achieved by having an IV-converter with a discrete differential voltage follower stage as the input. This stage and the necessary driving resistors, as well as the feedback resistor are mounted inside the sample chamber. Using the cold front-end reduces the input bias current and the noise generated in the feedback resistor. Additionally all long wires are driven by a low impedance source which reduces the EMI.
A block diagram is shown in figure C.30. The full schematic for the cold front-end is shown in figure C.31 and the insert is shown in figure C.32. The pin configuration is shown in figure C.33. Photos of the complete insert and the sample chamber electronics are shown in figure C.34.

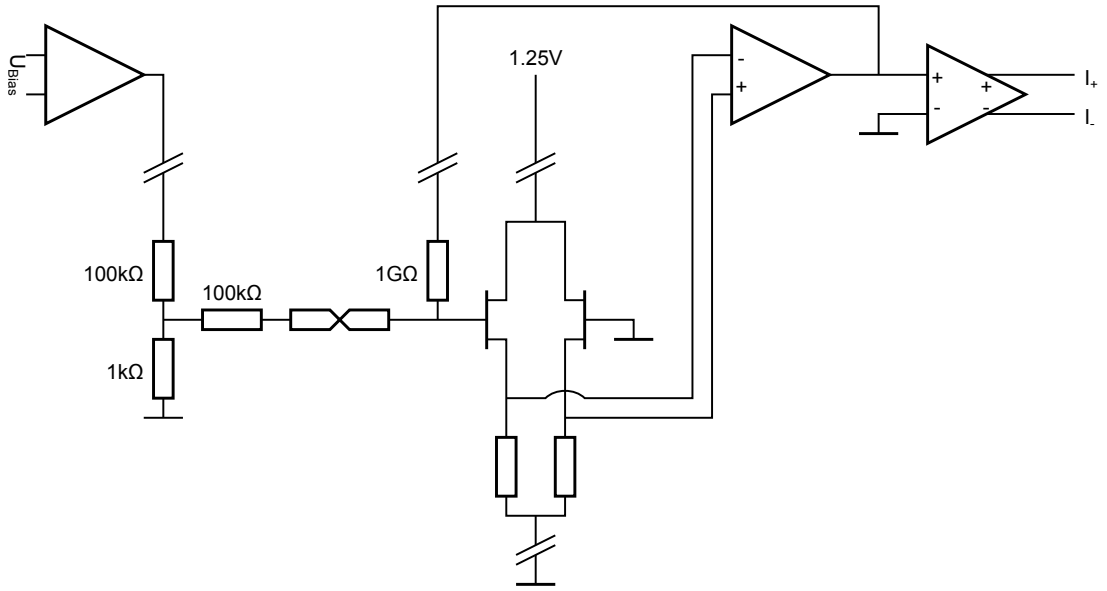



**Figure C.30.:** Block diagram of the cold IV converter

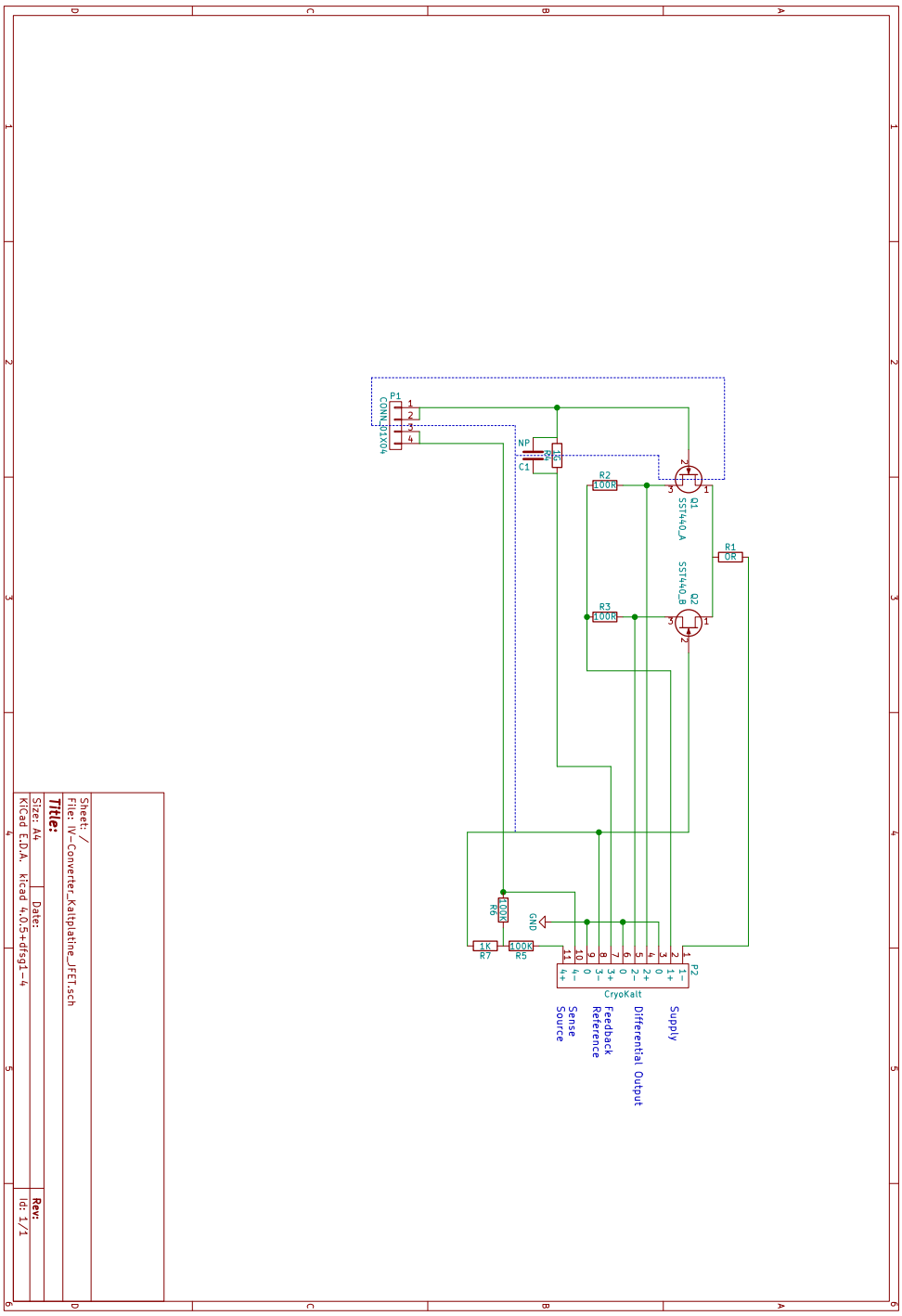



**Figure C.31.:** Complete schematic of the cold front-end

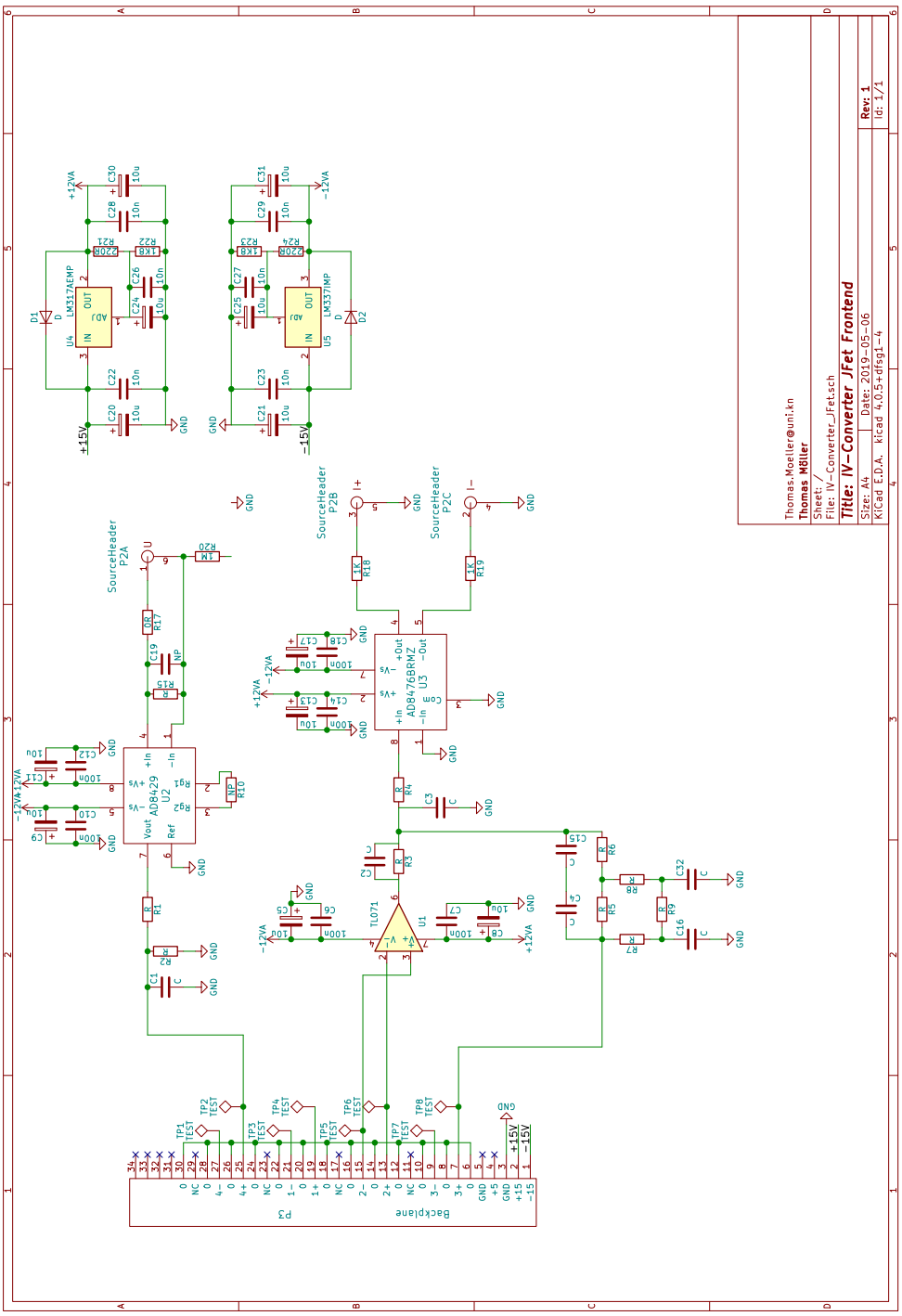
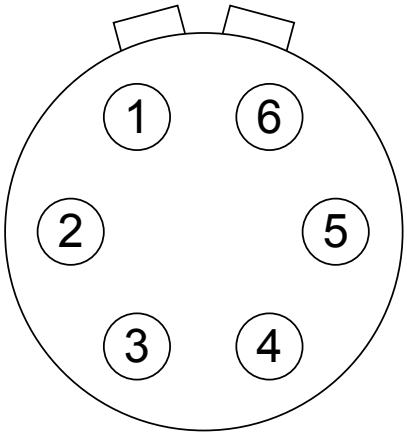
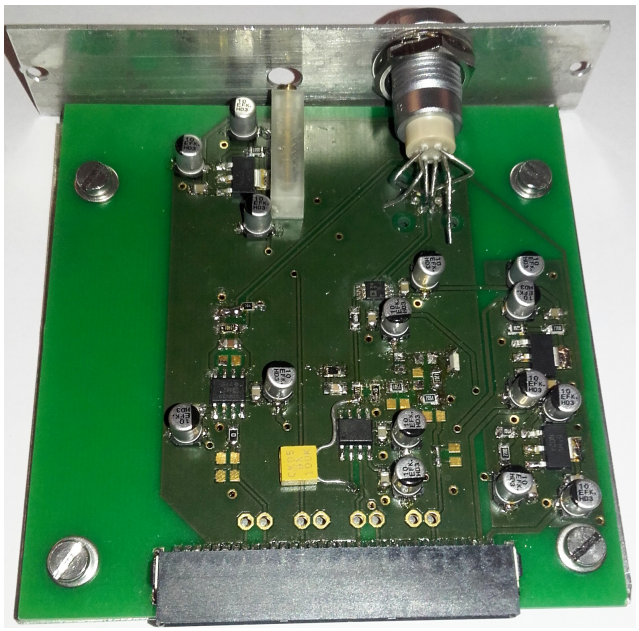



**Figure C.32.:** Complete schematic of the cold front-end IV-converter IceCube insert

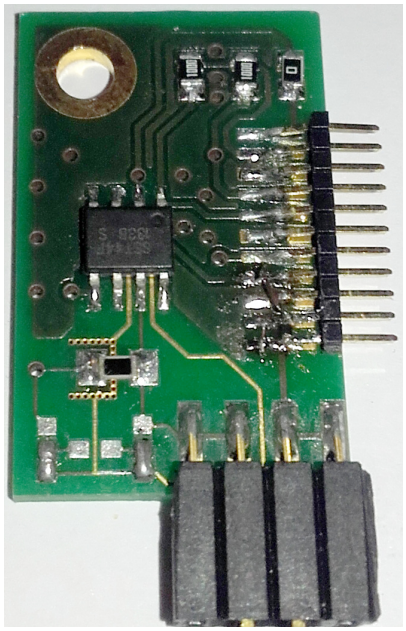



| | | |
|---|---|---|
| 1 | Bias voltage reference | input |
| 2 | not connected | |
| 3 | IV signal positive | output |
| 4 | IV signal negative | output |
| 5 | not connected | |
| 6 | Bias voltage signal | input |

**Figure C.33.:** Pin configuration of the cold front-end IV-converter insert as seen from the outside



**(a)** IceCube insert



**(b)** PCB inside the sample chamber

**Figure C.34.:** Photos of the IV-converter and the cold front-end

## C.7. Cold IV-Converter

The next step in measuring the molecules is to cool down the complete IV-converter. This reduces the stability problems which occurred with the multi stage design. A small yet flexible circuit is chosen for the cold end, and the insert in the IceCube provides the power and some supporting features like the line driver and the bias source.
A block diagram is shown in figure C.35. The full schematic for the insert is shown in figure C.36 and the front-end in figure C.37. The circuit board layouts are shown in figure C.38 and the pin configuration is shown in figure C.39. Photos of the complete insert and the sample chamber electronics are shown in figure C.40.

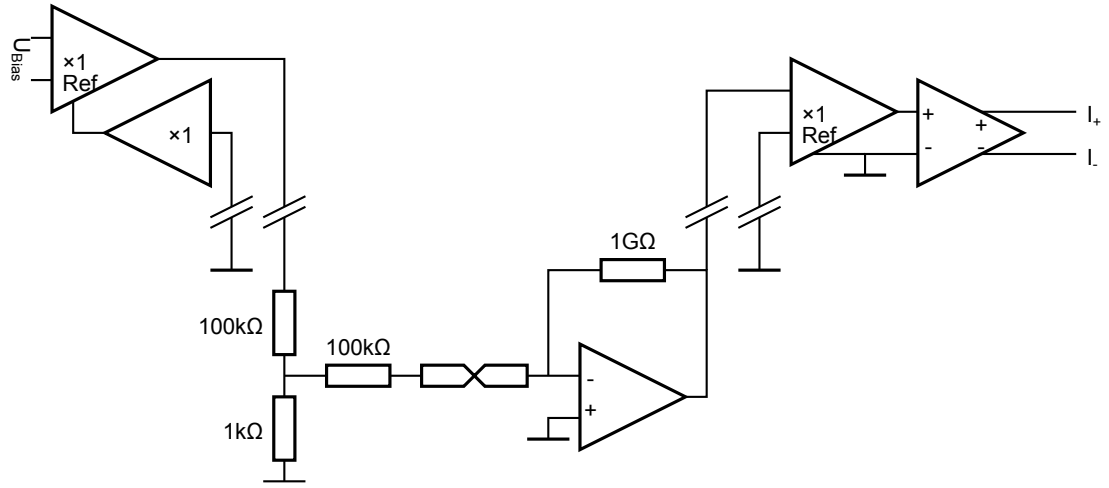



**Figure C.35.:** Block diagram of the cold IV-converter

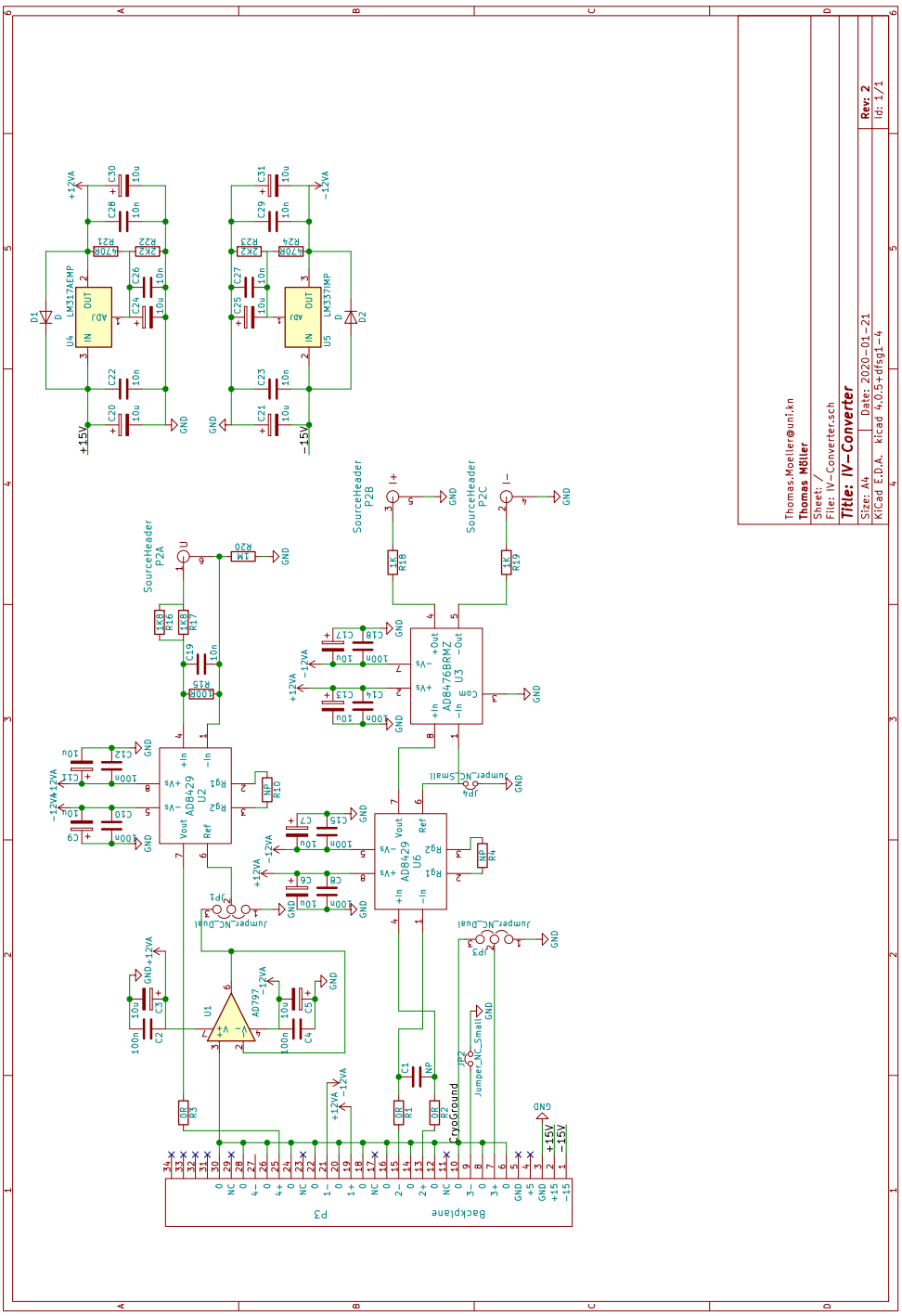



**Figure C.36.:** Complete schematic of the cold IV-converter IceCube insert

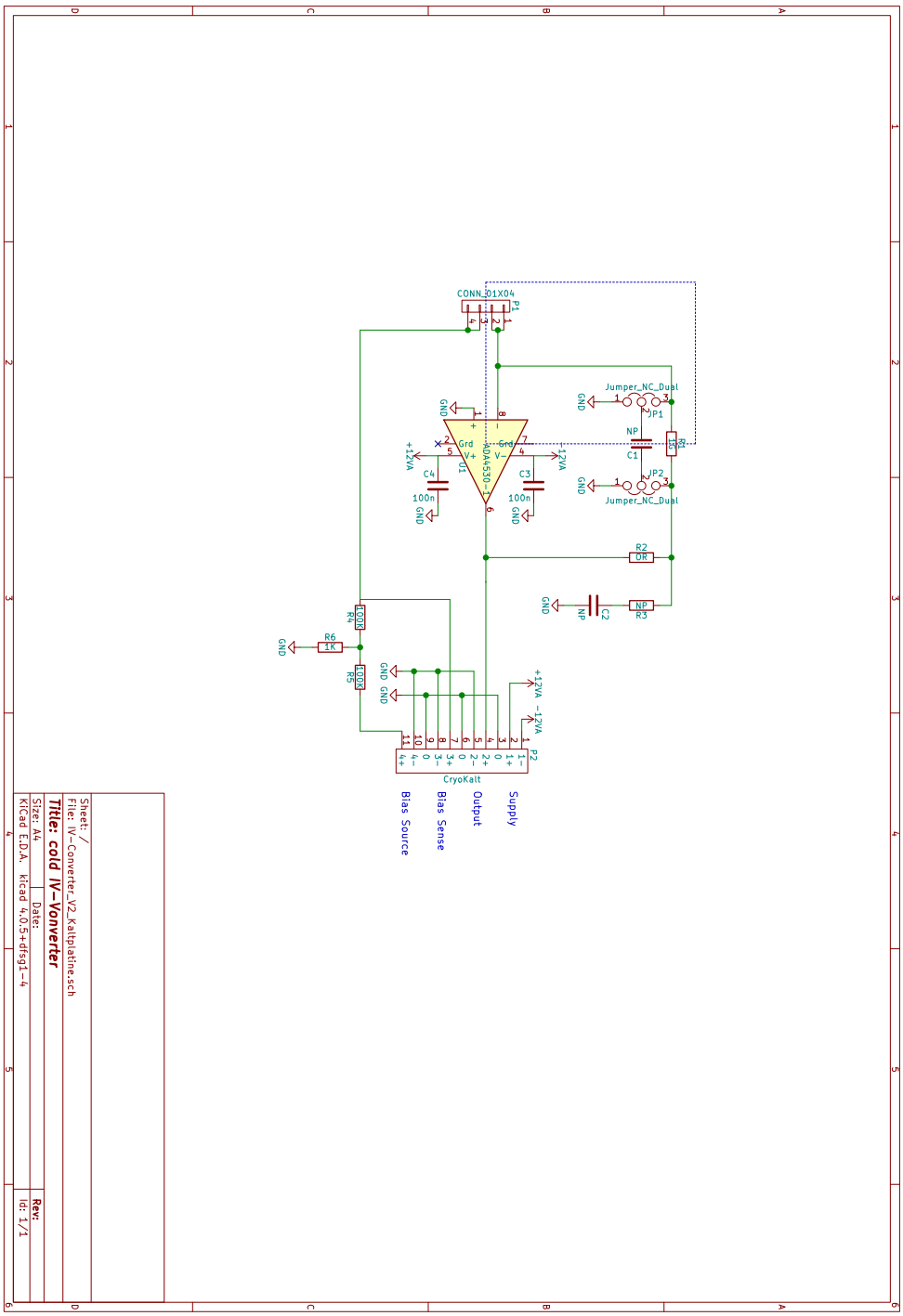
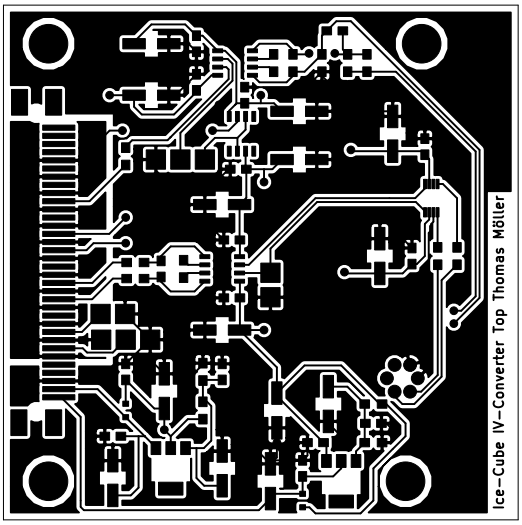
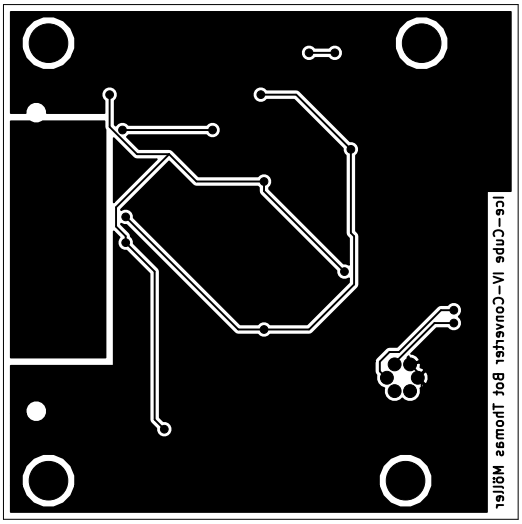


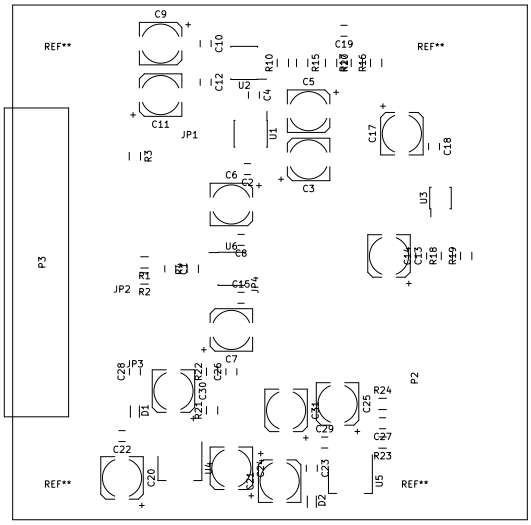

**Figure C.37.:** Complete schematic of the sample chamber electronics of the cold IV-converter




**(a)** Top side copper structure

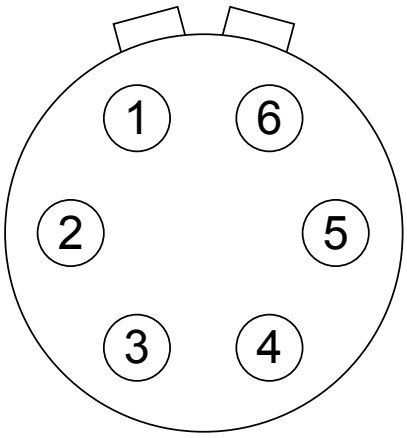



**(b)** Bottom side copper structure




**(c)** Component references

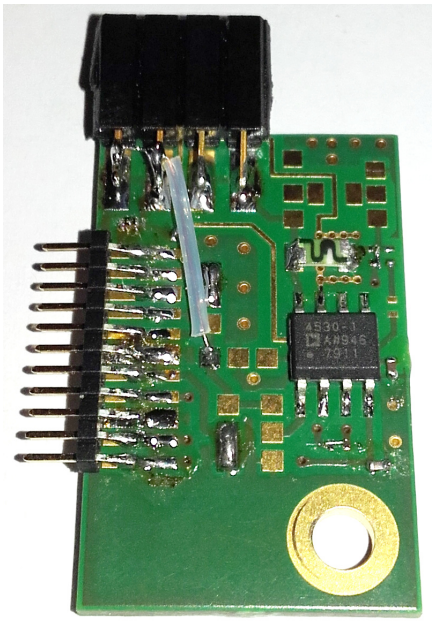
**Figure C.38.:** Circuit board layout of the cold IV-converter IceCube insert




| | | |
|---|---|---|
| 1 | Bias voltage reference | input |
| 2 | not connected | |
| 3 | IV signal positive | output |
| 4 | IV signal negative | output |
| 5 | not connected | |
| 6 | Bias voltage signal | input |

**Figure C.39.:** Pin configuration of the cold IV-converter insert as seen from the outside

**(a)** IceCube insert



**(b)** PCB inside the sample chamber

**Figure C.40.:** Photos of the cold IV-converter electronics

## C.8. Breakout

This insert provides direct access to each measurement line. It is used in combination with the "multiplexer" NIM crate module (see appendix D.2 on page 200) to allow automated and complete verification of the wiring. It also gives the complete flexibility for testing. To protect the sample and to reduce the noise influence of the external connections, all measurement lines are filtered.
The schematic is shown in figure C.41 and the circuit board layout is shown in figure C.42. The pin configuration is shown in figure C.43 and a photo of the complete insert is shown in figure C.44.

**Figure C.41.:** Complete schematic of the breakout IceCube insert

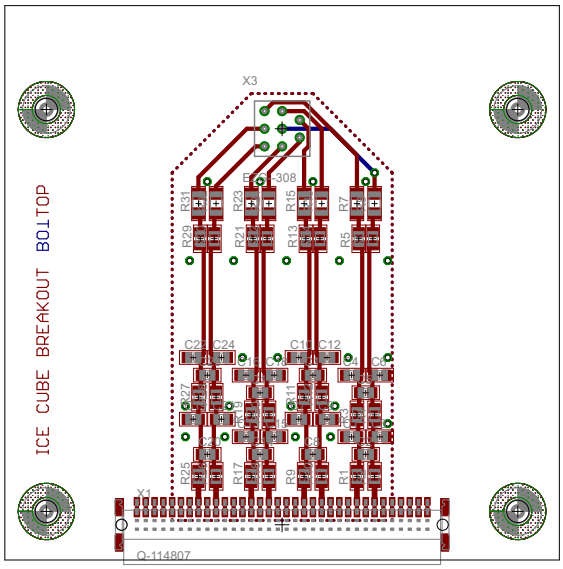
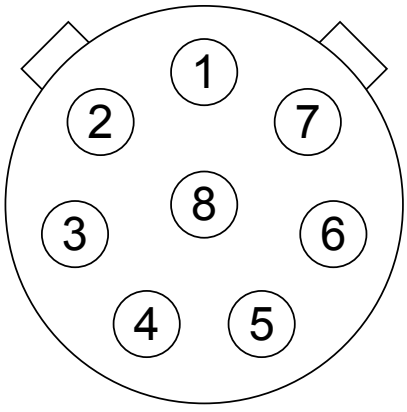



**Figure C.42.:** Circuit board layout of the breakout IceCube insert




| | | |
|---|---|---|
| 1 | 1- | input/output |
| 2 | 1+ | input/output |
| 3 | 2- | input/output |
| 4 | 2+ | input/output |
| 5 | 3- | input/output |
| 6 | 4- | input/output |
| 7 | 4+ | input/output |
| 8 | 3+ | input/output |

**Figure C.43.:** Pin configuration of the front side connector of the breakout IceCube insert as seen from the outside

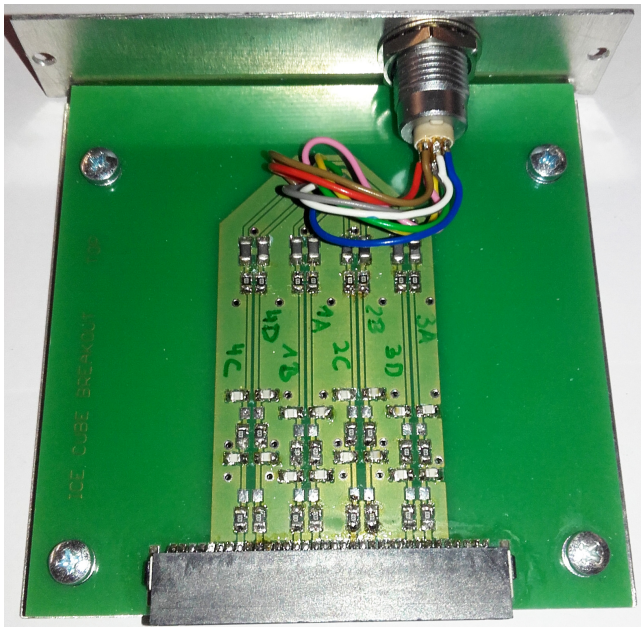



**Figure C.44.:** Photo of the breakout IceCube insert

## C.9. Grounding

When inserts are plugged in or the IceCube power supply is turned on, the electronics might produce voltage spikes. To prevent these spikes from damaging the sample, the grounding insert is used. All measurement lines are shorted to ground. Unfortunately the plugging in of such a short to ground may discharge some of the filtering capacitors through the sample. This is prevented by having the short to ground switched, so that, when the insert is plugged in, all measurement lines are pulled to ground via 1 MΩ resistors to slowly and safely discharge the capacitors. When everything is discharged, every wire pair can be individually switched to a low impedance ground connection.
A photo of the complete insert is shown in figure C.45.

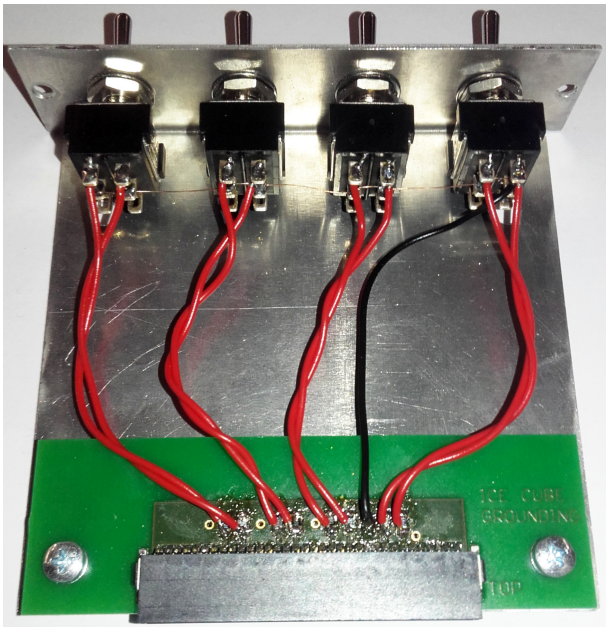



**Figure C.45.:** Photo of the grounding IceCube insert

## C.10. Debugging

This insert is used to perform debugging on the electronics of the setup. It has pins to externally provide the power supply normally generated by the supply insert and it simulates the wiring of the cryogenic insert. It has the same connector with the same pinout as can be found inside the sample chamber.
The schematic is shown in figure C.46 and the circuit board layout is shown in figure C.47. A photo of the complete insert is shown in figure C.48.

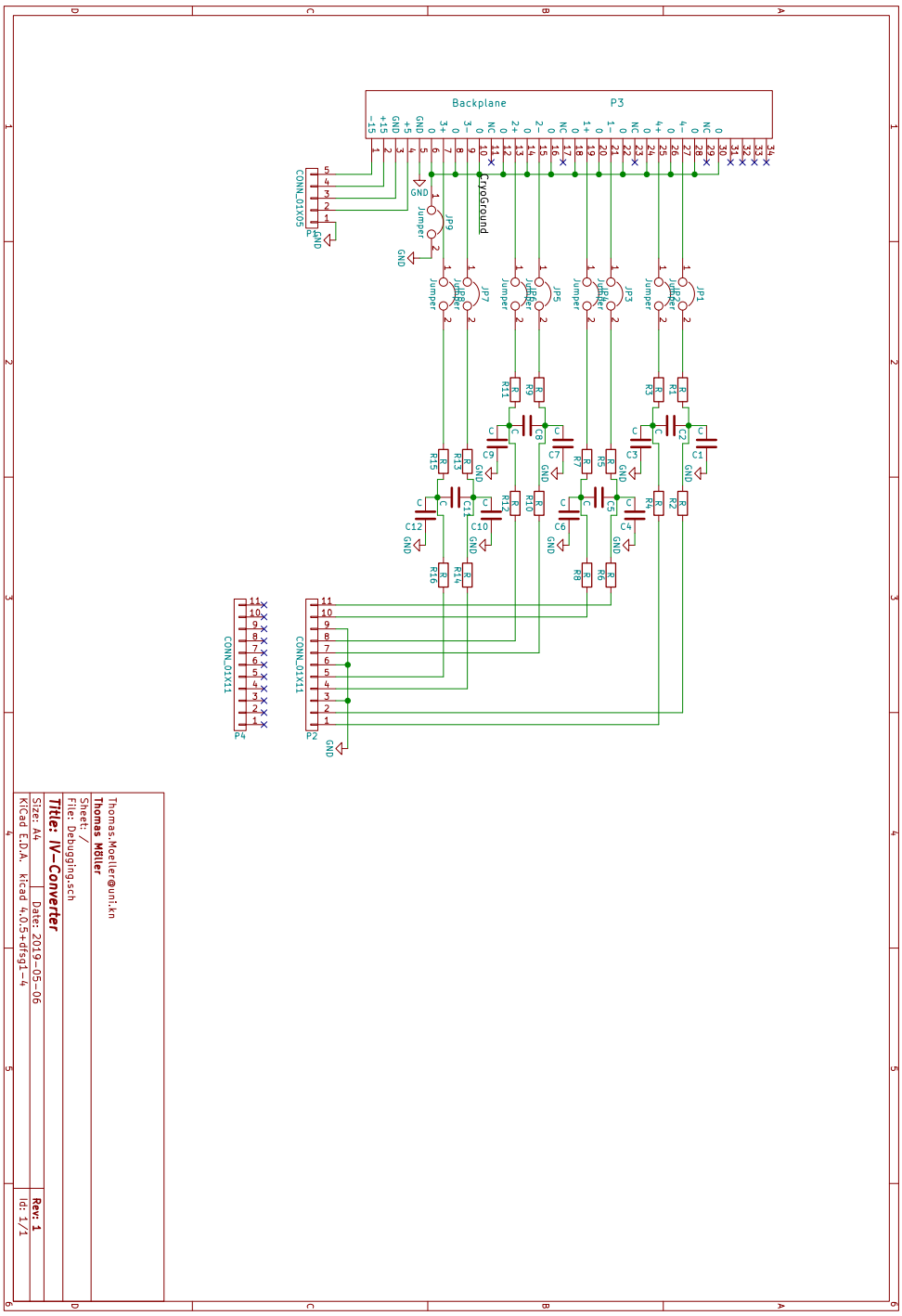
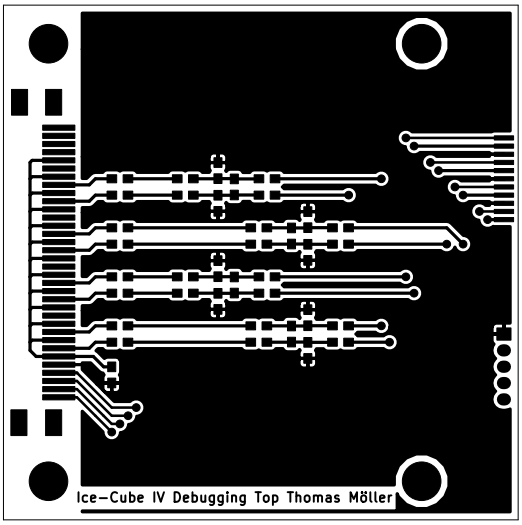
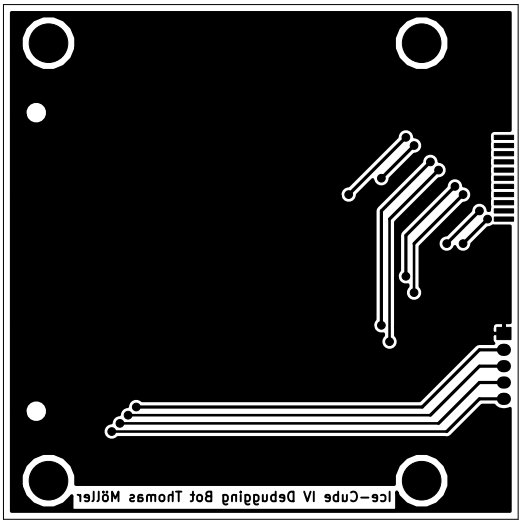


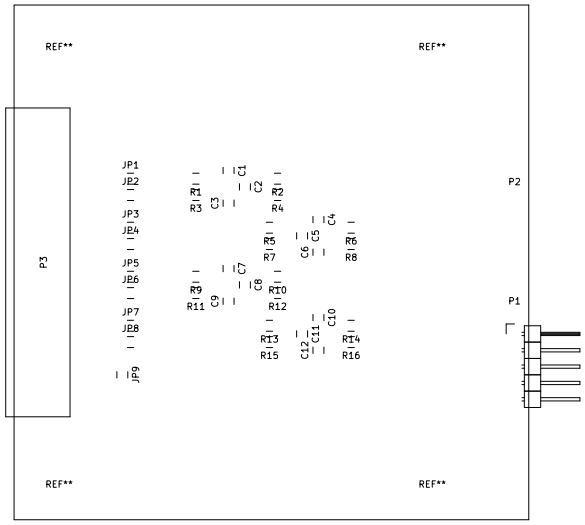

**Figure C.46.:** Complete schematic of the debugging IceCube insert




**(a)** Top side copper structure



**(b)** Bottom side copper structure




**(c)** Component references

**Figure C.47.:** Circuit board layout of the debugging IceCube insert

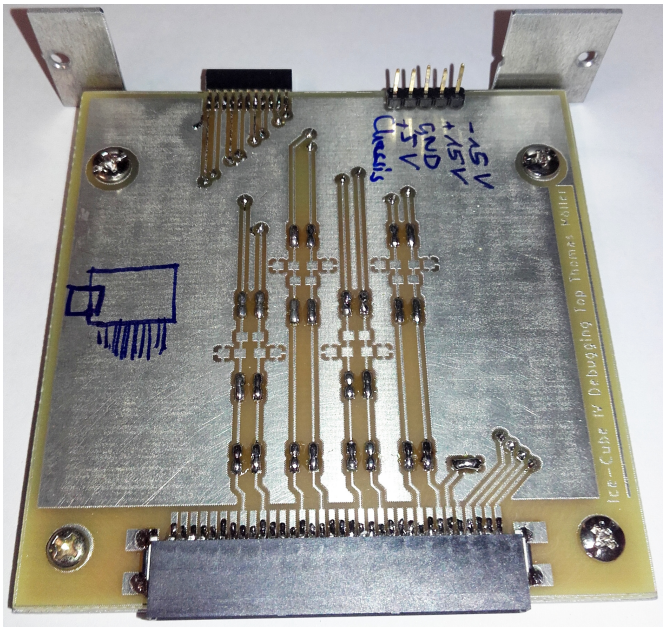



**Figure C.48.:** Photo of the debugging IceCube insert

# D. NIM crate modules

The important electronics for the signal conversion and amplification is in the aforementioned IceCube inserts. However this represents only a small part of the setup electronics. Most of the electronics is inside of custom built inserts for a NIM crate.
This commercial crate powers all of the setup electronics. It includes a symmetric linear power supply with 6, 12, and 24 V. The positive 6 V rail is used for the digital electronics and will therefore be rather "dirty". The symmetric 12 V rail is used to power the motor used to bend the sample. Although the supply rail is heavily filtered, the motor is driven by a pulse-width modulated signal, thus there will be switching noise in the 12 V rails as well. The 24 V rail is only used for static loads and is hence suitable as a power supply to analog parts of the setup electronics.
Most of the inserts contain a microcontroller connected to the computer by a USB interface which simulates a serial port. The communication is done at 9600 baud with 8 data bits, no parity bit, and at least one stop bit[1]. The line end can be any combination of `LF` and `CR` [2]. Unless specified otherwise, all commands will be acknowledged by the response `done` or, if the command could not be executed successfully, `failed`. The command syntax is loosely based on the usual GPIB command and the format of the SCPI language. The commands are not case sensitive and can be abbreviated to three characters. In the following documentation this is indicated by spelling the commands partially with capital letters (e.g. `CONnect`).
To prevent ground loops, the computer for controlling the setup needs to be isolated from the ground of the power supply. To realize this, custom built USB to serial adapters were integrated in each NIM insert with a microcontroller. They contain a FTDI FT232 USB to UART adapter chip with good driver support. The additional EEPROM can store a unique device identifier. This is used to identify and consistently name all the serial devices independent of the USB port, and the order in which they are plugged in. The chip and the access LEDs are powered from the USB port. The signals to the microcontroller are sent through an SI8422 chip which provides the galvanic isolation. The full schematic of this adapter is shown in figure D.1 and the circuit board layout is shown in figure D.2.

Most of the inserts have a microcontroller which interfaces with the rest of the hardware either directly or via an SPI bus. Since this large part of the schematic is identical for most inserts, it is shown in figure D.3 and omitted in the description of each individual insert.
Similar to the way the inserts share some common part in the schematic, they also share common parts in the firmware on the microcontroller. These files are printed in the following.

[1] The controller will respond using two stop bits.
[2] The controller will respond with `CRLF`.

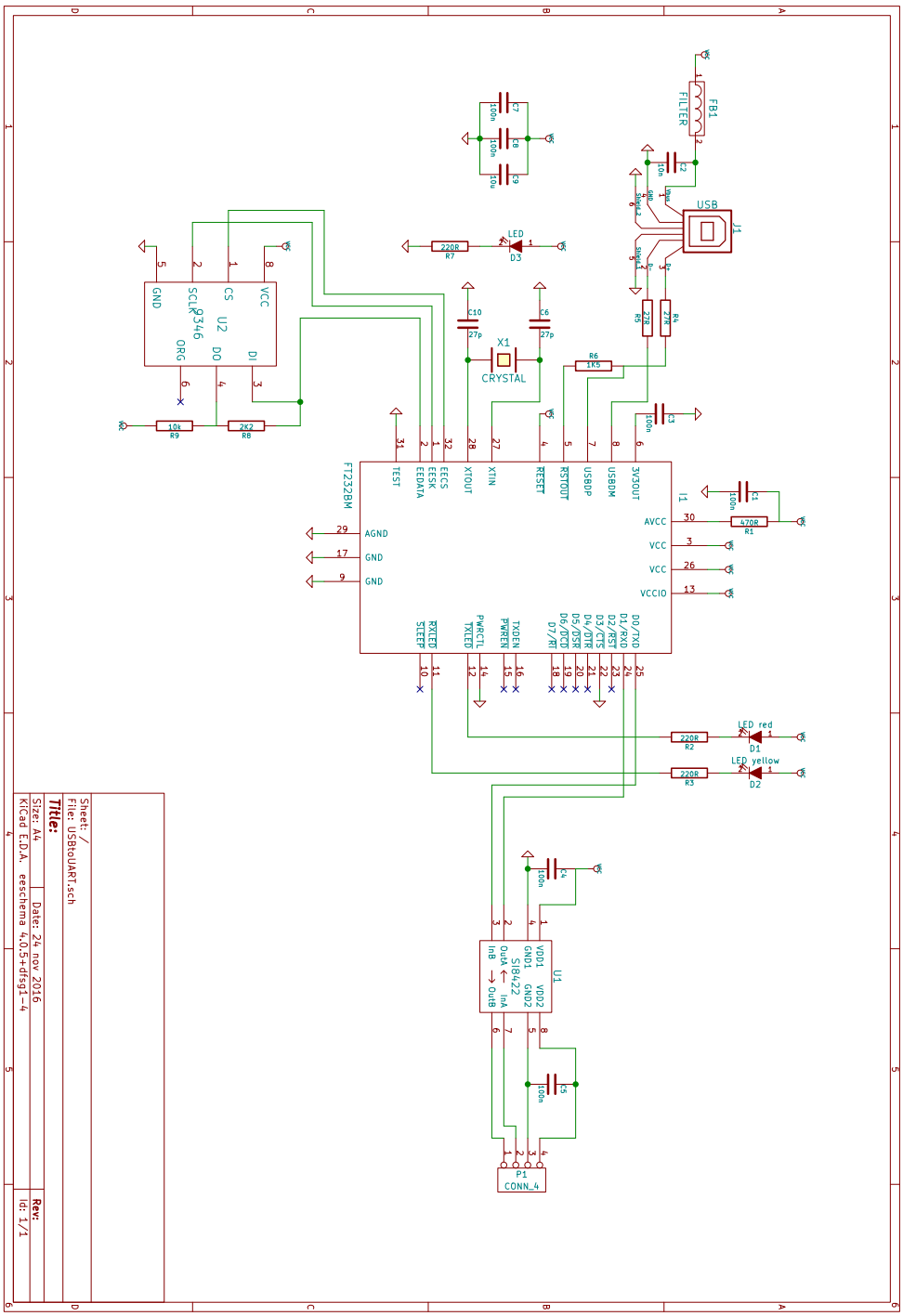
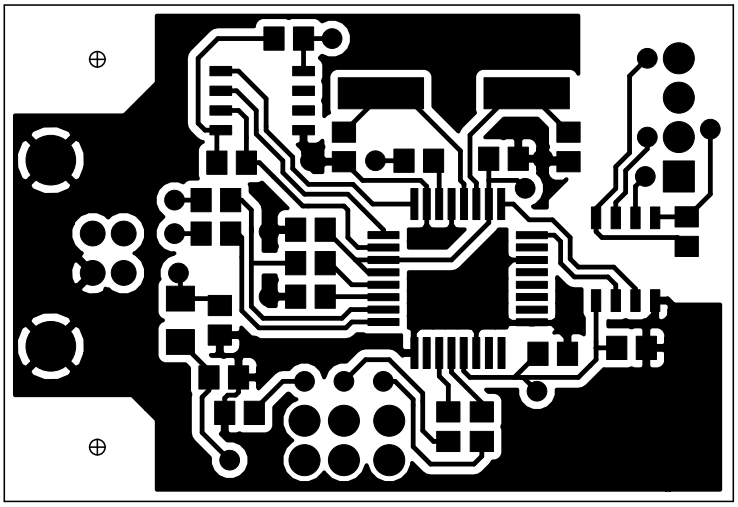
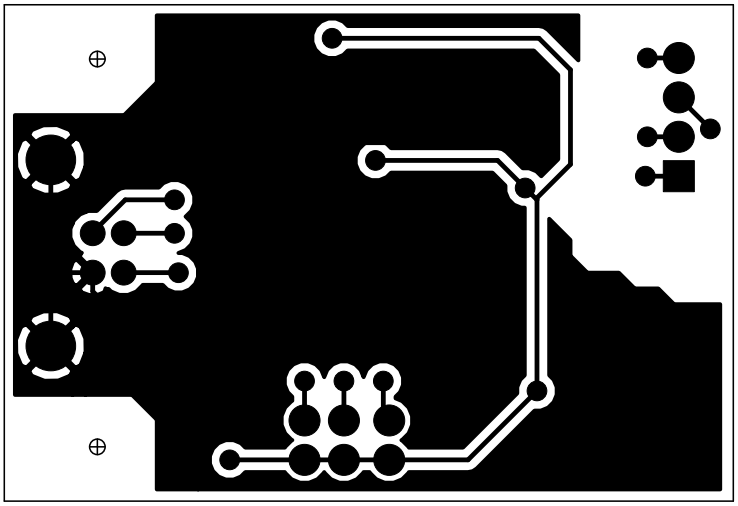


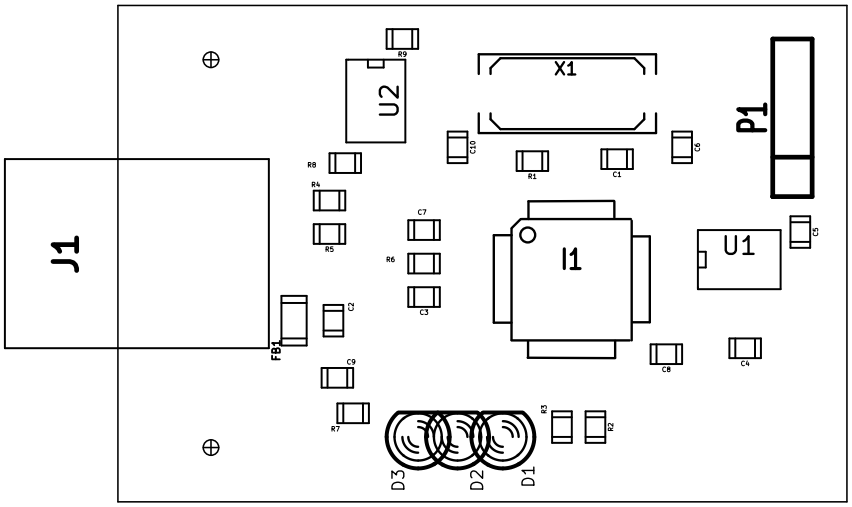

**Figure D.1.:** Complete schematic of the USB to serial adapter for the communication with the microcontroller



**(a)** Top side copper structure



**(b)** Bottom side copper structure




**(c)** Component references

**Figure D.2.:** Circuit board layout of the USB adapter board with galvanic isolation

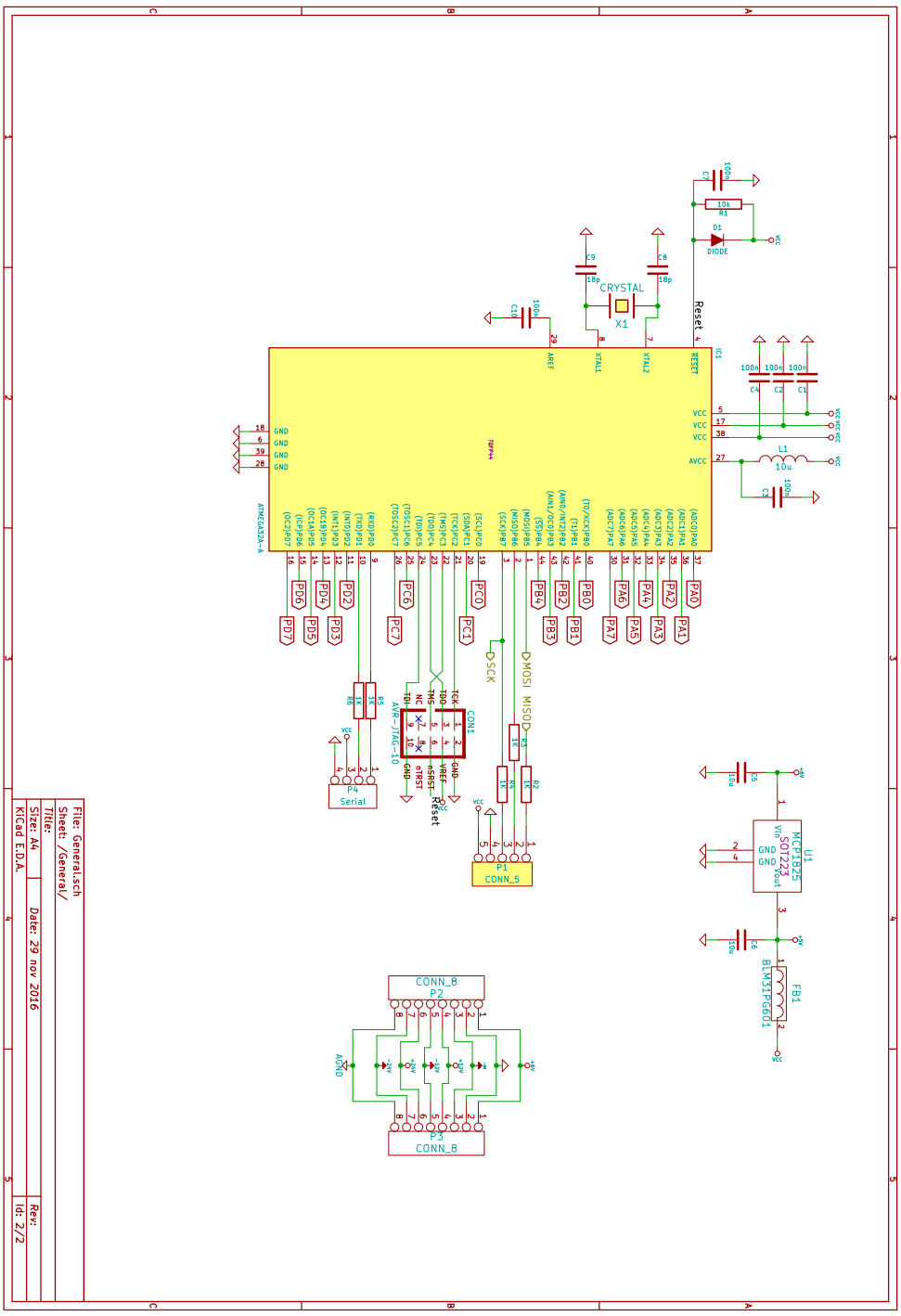



**Figure D.3.:** General schematic used in all the inserts with a microcontroller

**Source-Code D.1:** serial.h

```
#ifndef SERIAL_H_
#define SERIAL_H_

#include "types.h"

void initializeSerialPort();
void sendString(char input[]);

#endif /* SERIAL_H_ */
```

**Source-Code D.2:** serial.c

```
#include <avr/io.h>
#include "serial.h"
#include <avr/interrupt.h>
#include "parser.h"
#include "types.h"
#include <util/atomic.h>
#include "pinout.h"

volatile bool lastWasWhitespace = True; //To check for multiple whitespaces
volatile bool commandFinished = True; //To check for completed command
volatile uint8_t charsSinceLastColon = 0; //To automatically short commands to 3 chars
volatile uint8_t writePos=0; //Where to write the next char
volatile bool isData=False; //Keeps track whether it is in the data part of the command
    to prevent shortening and simplification

volatile char outputBuffer[100]; //Store data to bee sent
volatile uint8_t outputFilled = 0; //Elements in buffer
volatile uint8_t outputReadPos = 0; //Position where to read data from

void initializeSerialPort(){
  //Set Baud Rate to 9600
  UBRRL = 23;
  //Set TX Stop Bits to 2
  UCSRC |= (1<<URSEL) | (1<<USBS);
  //Set Parity to None
  //Set 8Bits per Character
  UCSRC |= (1<<URSEL) | (1<<UCSZ1) | (1<<UCSZ0);
  //Enable TX
  UCSRB |= (1<<TXEN);
  DDR_TX |= (1<<P_TX);
  //Enable RX
  PORT_RX |= (1<<P_RX);
  UCSRB |= (1<<RXEN);
  //Enable Receive Interrupt
  UCSRB |= (1<<RXCIE);
  //Enable Buffer Empty Interrupt
  UCSRB |= (1<<UDRIE);
}

bool isWhitespace(char buffer){
  if(buffer == ' ') return true;
  if(buffer == 9) return true;
  if(buffer == 11) return true;
  if(buffer == 12) return true;
  return false;
}

bool isEnd(char buffer){
  if(buffer == '\r') return true;
  if(buffer == '\n') return true;
  return false;
}

bool isColon(char buffer){
  if(buffer == ':'){
    return True;
```

```
  }else{
    return False;
  }
}

bool isUncounedChar(char buffer){
  if(buffer == '?') return true;
  return false;
}

bool isOldStart(char buffer){
  if(buffer == '*') return true;
  return false;
}

char simplifyCharacter(char input){
  if(input < 'a') return input;
  if(input <= 'z') return input-32;
  return input;
}

void newCharacter(char buffer){
  if(lastWasWhitespace && isWhitespace(buffer)) return; //remove multiple whitespaces
  if(isEnd(buffer)){ //End of command
    if(commandFinished) return; //Return if no multiple ends are recieved
    commandFinished = True; //Restore start-settings
    charsSinceLastColon = 0;
    lastWasWhitespace = True;
    isData = False;
    parseCommand(); //Parse input
    return;
  }
  if(commandFinished){ //At the beginning of an new command
    commandFinished = False;
    if(isColon(buffer)){ //When command starts with colon (absolute path), start at the
        front
      writePos = 0;
    }else if(isOldStart(buffer)){
      writePos = 0;
      isData = True; //Trick parser into accepting all remaining stuff
    }else{ //Leave old "tree" if no absolute path is given
      command[0]=':'; //make sure that there is one at the beginning
      writePos = 1; //so that the start is correct if the search remains fruitless
      for(uint8_t i=99;i>0;i--){ //Search for last colon
        if(command[i]==':'){
          writePos=i+1;
          break;
        }
      }
    }
    for(uint8_t i=writePos;i<100;i++) command[i] = 0; //Clear old stuff
  }
  if(isWhitespace(buffer)){ //In case of new Whitespace (multiples don't reach this point
      )
    command[writePos++] = '␣';
    lastWasWhitespace = True;
    isData = True; //Whitespace marks the transition from command to data
    return;
  }
  lastWasWhitespace = False; //This point is only reached with a normal character
  if(isData){
    command[writePos++] = simplifyCharacter(buffer); //Don't mess with the data, just
        make it uppercase
  }else{
    if(isColon(buffer)){
      command[writePos++] = ':';
      charsSinceLastColon = 0;
    }else if(isUncounedChar(buffer) || (charsSinceLastColon<3)){ //Don't put more than 3
        chars in the command except for special chars like '?'
      command[writePos++] = simplifyCharacter(buffer); //and make them simple
```

```
      charsSinceLastColon++;
    }
  }
}

ISR(USART_RXC_vect){
  char buffer;
  if(UCSRA & ((1<<FE) | (1<<DOR) | (1<<PE))){ //Check for FrameError, DataOverRun and
      ParityError
    //Error
    while (UCSRA & (1<<RXC)){ //Flush input buffer
      char _ = UDR;
    }
  }else{
    buffer = UDR;
    newCharacter(buffer);
  }
}

void transmitData(){
  if(outputFilled == 0) return;
  while(~UCSRA &(1<<UDRE)); //Wait for TX (should return immediately)
  UDR = outputBuffer[outputReadPos];
  outputFilled--;
  if(++outputReadPos>99) outputReadPos=0;
  UCSRB |= (1<<UDRIE); //Enable Interrupts to get informed when new data can be
      transmitted
}

void sendString(char input[]){
  uint8_t i = 0;
  if (outputFilled>=98) return;
  while(input[i] && outputFilled<98){
    ATOMIC_BLOCK(ATOMIC_RESTORESTATE){
      uint8_t pos = outputReadPos + outputFilled;
      if (pos >= 100) pos -= 100;
      outputBuffer[pos] = input[i];
      outputFilled++;
    }
    i++;
  }
  ATOMIC_BLOCK(ATOMIC_RESTORESTATE){
    uint8_t pos = outputReadPos + outputFilled;
    if (pos >= 100) pos -= 100;
    outputBuffer[pos] = '\r';
    outputFilled++;
  }
  ATOMIC_BLOCK(ATOMIC_RESTORESTATE){
    uint8_t pos = outputReadPos + outputFilled;
    if (pos >= 100) pos -= 100;
    outputBuffer[pos] = '\n';
    outputFilled++;
  }
  if(~UCSRB & (1<<UDRIE)) transmitData(); //Interrupt can only be off, when transmission
      was finished
}

ISR(USART_UDRE_vect){
  UCSRB &= ~(1<<UDRIE); //Turn off interrupt - Is re-enabled when new data is sent
  transmitData();
}
```

**Source-Code D.3:** types.h

```
#ifndef TYPES_H_
#define TYPES_H_

#include <stdint.h>
```

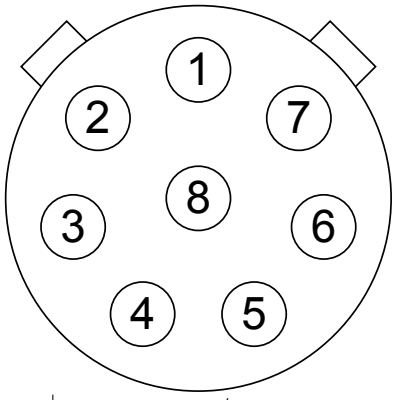
```
#define bool uint8_t
#define True 0xff
#define False 0x00
#define true 0xff
#define false 0x00

#endif /* TYPES_H_ */
```

# D.1. Breakout

This insert serves only as an adapter between the eight-pin connector to the IceCube breakout insert (see appendix C.8 on page 180)and eight normal SMC connectors at the front thus enabling easy and fully flexible access to all measurement lines. The pinout is shown in figure D.4, photos of the front and back are shown in figure D.5, and the inside in figure D.6.


| | | |
|---|---|---|
| 1 | 3+ | input/output |
| 2 | 3− | input/output |
| 3 | 2+ | input/output |
| 4 | 2− | input/output |
| 5 | 1+ | input/output |
| 6 | 1− | input/output |
| 7 | 4+ | input/output |
| 8 | 4− | input/output |

**(a)** front




| | | |
|---|---|---|
| 1 | 1− | input/output |
| 2 | 1+ | input/output |
| 3 | 2− | input/output |
| 4 | 2+ | input/output |
| 5 | 3− | input/output |
| 6 | 4− | input/output |
| 7 | 4+ | input/output |
| 8 | 3+ | input/output |

**(b)** back

**Figure D.4.:** Pin configuration of the breakout NIM insert


**(a)** front

**(b)** back

**Figure D.5.:** Photos of the outside of the breakout NIM insert

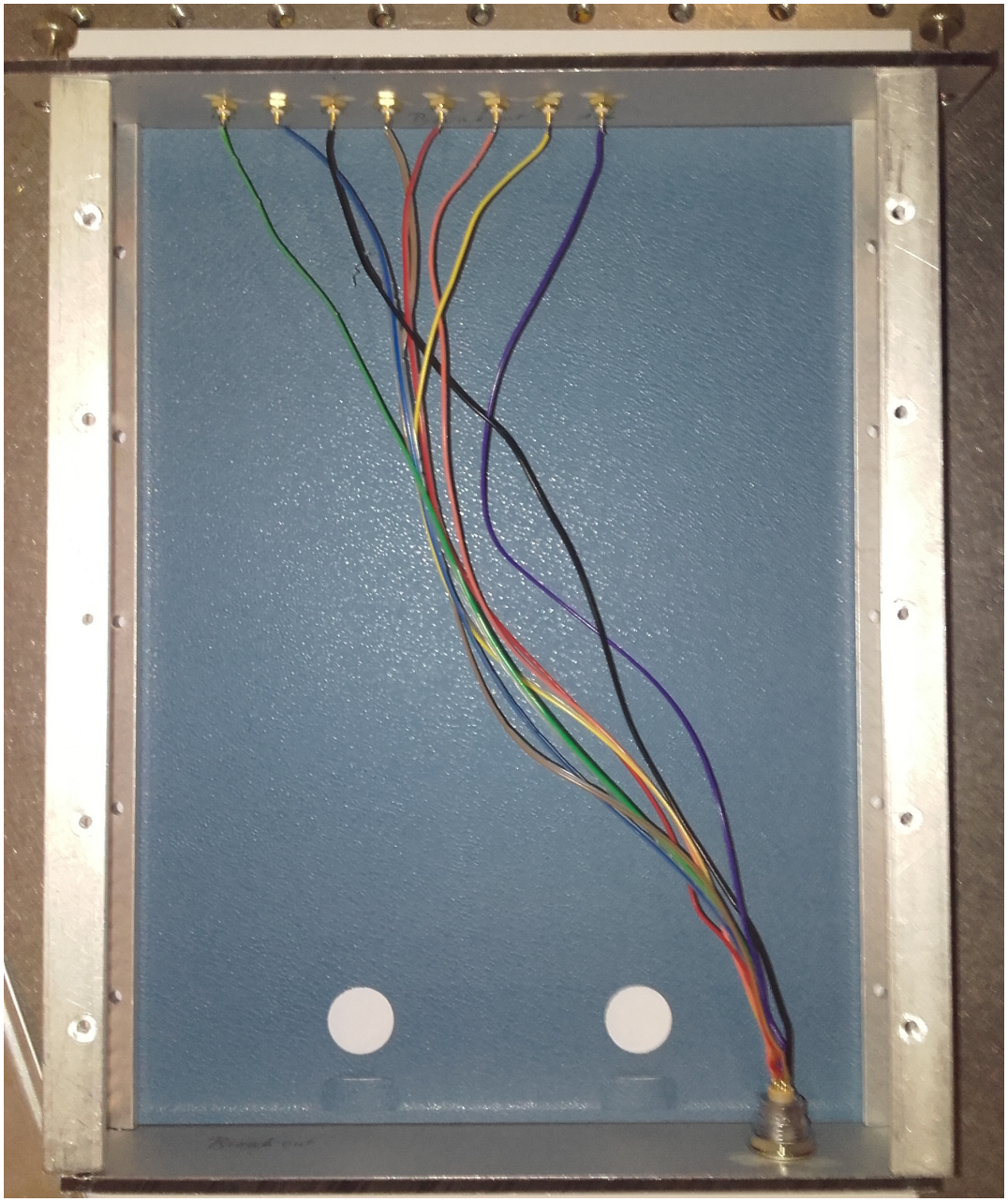


**Figure D.6.:** Photo of the inside of the breakout NIM insert

## D.2. Multiplexer

The multiplexer insert is used in conjunction with the breakout insert. It has four direct outputs and four outputs where the signals are buffered by an instrumentation amplifier. This means the output can be amplified and shifted to different reference levels.
Every output consists of two switching units: one for the inner conductor of the BNC connector and one for the shield. In the buffered outputs the switches are connected to the positive and negative inputs of the instrumentation amplifier. There, the BNC central conductor is connected to the output of the instrumentation amplifier, and the shield is connected to the reference input.
The inputs to these switching units are the twelve inputs, a low impedance ground, and a 1 MΩ resistor to ground. When the insert is in the "multiple" mode, one of the twelve inputs can be connected to several outputs at the same time.
The switching possibilities can be seen in the simplified block diagram in figure D.7.
All connections can be configured by the inputs in the front and via the computer interface using the following commands.

- `*IDN?`
  This command returns the name of the insert, the author, the version, and date of the firmware. The output is three lines long.
- `:CLEar:ALL` This command clears all connections during the next update of the switches. To find out if and when this update has occurred use the `:OPE?` command.
- `:OPErationcomplete?` This command will return `done` if all switches are set to the positions required by previous commands.
- `:CLEar:INPut␣<Number>` This command clears all connections to the specified input during the next update of the switches. To find out if and when this update has occurred use the `:OPE?` command.
- `:CLEar:OUTput:[SIGnal|SHIeld]␣<Number>` This command clears all connections to the specified output during the next update of the switches. To find out if and when this update has occurred use the `:OPE?` command.
- `:SINgle` This command sets the insert to the "single connection" mode. It includes the clearing of all connections.
- `:CON:<OutputNumber>:[SIGnal|SHIeld]␣<InputNumber>` This command connects the specified output to the specified input during the next update of the switches. To find out if and when this update has occurred use the `:OPE?` command.

Any number between 1 and 16 is a valid input. The numbers 1 to 12 correspond to the inputs on the front, 13 is the low impedance ground connection, and 16 is the high impedance ground connection. Numbers 14 and 15 were intended to be connections to the positive and negative power rail. However, this resulted in several design problems and was not implemented. Any requests to connect to input 14 and 15 will be ignored by the controller.
The outputs are specified by a combination of the number 1 to 8 and either `SIGnal` to

specify the inner conductor, or SHIeld for the shield connection on the plug. Outputs 1 to 4 are direct connections and 5 to 8 are buffered and amplified.
To configure the multiplexer with the controls in the front, the unit has two displays, three LEDs and four two directional momentary switches.
The left display shows the currently selected input numbered from 0 1 to 0 2, A 0 (hard ground) for the low impedance ground connection, or S 0 (soft ground) for the high impedance ground connection. The settings 0 S and A S for the positive and negative supply rails cannot be used. The input selection can be changed by pushing the left switch up to increase the number or down to decrease the number.
The right display shows the output: The left digit shows the number and the right digit shows 0 for the signal, and 0 for the shield. The right switch changes the selected output channel. Again, pushing up increases the number and down decreases it.
If the selected input and output are connected the green LED between the displays lights up. Pushing the top switch to the right connects the input and output and pushing it to the left disconnects them.
The two LEDs on the sides of the bottom switch show the mode of operation. In single connection mode the left green LED is switched on and in multiple connection mode the right yellow LED is switched on. Pushing the bottom switch to the left changes the mode to single connection, pushing it to the right changes to mode to multiple connection.
The full schematics of the circuit boards are shown in figures D.8 to D.14 and the corresponding board layouts in figures D.15 to D.17. The pinout of the connectors is shown in figure D.18 and photos of the front, back, and inside are shown in figures D.19 and D.20.
The firmware consists of the following files.

**Source-Code D.4:** display.h

```
#ifndef DISPLAY_H_
#define DISPLAY_H_

#include "types.h"

void displayInput(uint8_t number);
void displayOutput(uint8_t number);

#endif /* DISPLAY_H_ */
```

**Source-Code D.5:** display.c

```
#include "display.h"
#include "SPI.h"

/*
 *   2222
 *  7    3
 *  7    3
 *   8888
 *  6    4
 *  6    4
 *   5555  1
 */

char numbers[] = {
    0b01111110, //0
    0b00001100, //1
    0b10110110, //2
    0b10011110, //3
    0b11001100, //4
    0b11011010, //5
```

```
    0b11111010, //6
    0b00001110, //7
    0b11111110, //8
    0b11011110  //9
};
char positive = 0b11100110;
char negative = 0b10101000;
char ground = 0b01111010;
char supply = 0b11011010;
char signal = 0b10000000;
char shield = 0b00011110;
char empty = 0b00000000;
char hard = 0b11101100;
char soft = 0b11011010;

void displayInput(uint8_t number){ //Set the input display correctly. 1 <= number <= 16
  if (number < 10){
    SPIsetDisplay(0,empty);
    SPIsetDisplay(1,numbers[number]);
  }else if(number <= 12){
    SPIsetDisplay(0,numbers[1]);
    SPIsetDisplay(1,numbers[number-10]);
  }else if(number == 13){
    SPIsetDisplay(0,hard);
    SPIsetDisplay(1,ground);
  }else if (number == 14){
    SPIsetDisplay(0,positive);
    SPIsetDisplay(1,supply);
  }else if(number == 15){
    SPIsetDisplay(0,negative);
    SPIsetDisplay(1,supply);
  }else if(number == 16){
    SPIsetDisplay(0,soft);
    SPIsetDisplay(1,ground);
  }
  SPItriggerTransmission();
}

void displayOutput(uint8_t number){ //set the output display correcty. 2 <= number <= 17
    (even numbers are signal, odd shield)
  SPIsetDisplay(2,numbers[number>>1]);
  if(number & 0b00000001){
    SPIsetDisplay(3,shield);
  }else{
    SPIsetDisplay(3,signal);
  }
  SPItriggerTransmission();
}
```

**Source-Code D.6:** hardware.h

```
#ifndef HARDWARE_H_
#define HARDWARE_H_

#include "types.h"

void hardwareInitialize();
void hardwareScan();
void hardwareConnectedLED(bool state);
void hardwareSingleLED();
void hardwareMultipleLED();

extern bool flagInputUp;
extern bool flagInputDown;
extern bool flagOutputUp ;
extern bool flagOutputDown;
extern bool flagConnect;
extern bool flagDisconnect;
extern bool flagSingle;
```

```
extern bool flagMultiple;

#endif /* HARDWARE_H_ */
```

**Source-Code D.7:** hardware.c

```
#include "hardware.h"
#include "pinout.h"
#include <util/delay.h>

bool stateInputUp = False;
bool stateInputDown = False;
bool stateOutputUp = False;
bool stateOutputDown = False;
bool stateConnect = False;
bool stateDisconnect = False;
bool stateSingle = False;
bool stateMultiple = False;

bool flagInputUp = False;
bool flagInputDown = False;
bool flagOutputUp = False;
bool flagOutputDown = False;
bool flagConnect = False;
bool flagDisconnect = False;
bool flagSingle = False;
bool flagMultiple = False;

void hardwareInitialize(){
  DDRLEDConnected |= (1<<PLEDConnected);
  DDRLEDSingle |= (1<<PLEDSingle);
  DDRLEDMultiple |= (1<<PLEDMultiple);
  PORTConnect |= (1<<PConnect);
  PORTDisconnect |= (1<<PDisconnect);
  PORTInputUp |= (1<<PInputUp);
  PORTInputDown |= (1<<PInputDown);
  PORTOutputUp |= (1<<POutputUp);
  PORTOutputDown |= (1<<POutputDown);
  PORTSingle |= (1<<PSingle);
  PORTMultiple |= (1<<PMultiple);
}

void hardwareConnectedLED(bool state){
  if(state){
    PORTLEDConnected |= (1<<PLEDConnected);
  }else{
    PORTLEDConnected &= ~(1<<PLEDConnected);
  }
}

void hardwareSingleLED(){
  PORTLEDSingle |= (1<<PLEDSingle);
  PORTLEDMultiple &= ~(1<<PLEDMultiple);
}

void hardwareMultipleLED(){
  PORTLEDSingle &= ~(1<<PLEDSingle);
  PORTLEDMultiple |= (1<<PLEDMultiple);
}

void hardwareScan(){
  if(~PINInputUp & (1<<PInputUp)){
    if(!stateInputUp){
      flagInputUp = True;
    }
    stateInputUp = True;
  }else{
    stateInputUp = False;
  }
```

```
  if(~PINInputDown & (1<<PInputDown)){
    if(!stateInputDown){
      flagInputDown = True;
    }
    stateInputDown = True;
  }else{
    stateInputDown = False;
  }
  if(~PINOutputUp & (1<<POutputUp)){
    if(!stateOutputUp){
      flagOutputUp = True;
    }
    stateOutputUp = True;
  }else{
    stateOutputUp = False;
  }
  if(~PINOutputDown & (1<<POutputDown)){
    if(!stateOutputDown){
      flagOutputDown = True;
    }
    stateOutputDown = True;
  }else{
    stateOutputDown = False;
  }
  if(~PINConnect & (1<<PConnect)){
    if(!stateConnect){
      flagConnect = True;
    }
    stateConnect = True;
  }else{
    stateConnect = False;
  }
  if(~PINDisconnect & (1<<PDisconnect)){
    if(!stateDisconnect){
      flagDisconnect = True;
    }
    stateDisconnect = True;
  }else{
    stateDisconnect = False;
  }
  if(~PINSingle & (1<<PSingle)){
    if(!stateSingle){
      flagSingle = True;
    }
    stateSingle = True;
  }else{
    stateSingle = False;
  }
  if(~PINMultiple & (1<<PMultiple)){
    if(!stateMultiple){
      flagMultiple = True;
    }
    stateMultiple = True;
  }else{
    stateMultiple = False;
  }
}
```

**Source-Code D.8:** main.c

```
#include "serial.h"
#include <avr/interrupt.h>

#include "hardware.h"
#include "userInterface.h"
#include "SPI.h"


int main(){
```

```
  SPIInitialize();
  hardwareInitialize();
  initializeSerialPort();
  sei();
  userInterfaceInitialize();
  while(True){
    hardwareScan();
    handleFlags();
  }
}
```

**Source-Code D.9:** management.h

```
/*
 * management.h
 *
 *  Created on: 31.10.2017
 *      Author: thomas
 */

#ifndef MANAGEMENT_H_
#define MANAGEMENT_H_

#include "types.h"

bool operationComplete();
bool isConnected(uint8_t input, uint8_t output);
void addConnection(uint8_t input, uint8_t output);
void removeConnection(uint8_t input, uint8_t output);
void clearOutput(uint8_t output);
void clearInput(uint8_t input);
void clearAll();
void singleConnection(uint8_t input, uint8_t output);

extern volatile bool singleMode;

//TODO: Single or Multiple Mode

#endif /* MANAGEMENT_H_ */
```

**Source-Code D.10:** management.c

```
#include "management.h"
#include "SPI.h"
#include "userInterface.h"

volatile uint8_t config[18];
volatile bool singleMode = True;

void update(){
  SPItriggerTransmission();
  userInterfaceUpdate();
}

bool operationComplete(){
  return transmissionComplete;
}

bool isConnected(uint8_t input, uint8_t output){
  if(input > 8){
    return (config[output] & 0xf0) == ((input-1)<<4);
  }else{
    return (config[output] & 0x0f) == ((1<<3) | (input-1));
  }
}

void addConnectionRaw(uint8_t input, uint8_t output){
  if(input == 14) return;   //Positive Supply leads to problems
  if(input == 15) return;   //Negative Supply leads to problems
```

```
  /*if((input > 12) && (input < 16)){   //If an input is forced to a hard powersupply
    for(uint8_t i = 13; i<16; i++){
      if(input != i) clearInput(i); //Make sure that no other power supply is used in the
           multiplexer to prevent shorts.
    }
  }*/
  if(input > 8){
    config[output] |= ((input-1)<<4);
  }else{
    config[output] |= (1<<3) | (input-1);
  }
  SPIsetOutput(output, config[output]);
}

void addConnection(uint8_t input, uint8_t output){
  singleMode = False;
  addConnectionRaw(input, output);
  update();
}

void removeConnectionRaw(uint8_t input, uint8_t output){
  if(!isConnected(input,output)) return;
  if(input > 8){
    config[output] &= ~(0xf0);
  }else{
    config[output] &= ~(0x0f);
  }
  SPIsetOutput(output, config[output]);
}

void removeConnection(uint8_t input, uint8_t output){
  removeConnectionRaw(input, output);
  update();
}

void clearOutputRaw(uint8_t output){
  config[output] = 0;
  SPIsetOutput(output, config[output]);
}

void clearOutput(uint8_t output){
  clearOutputRaw(output);
  update();
}

void clearInputRaw(uint8_t input){
  for(uint8_t i=2; i<18; i++){
    removeConnectionRaw(input,i);
  }
}

void clearInput(uint8_t input){
  clearInputRaw(input);
  update();
}

void clearAll(){
  for(uint8_t i=2; i<18; i++){
    clearOutputRaw(i);
  }
  singleMode = True;
  update();
}

void singleConnection(uint8_t input, uint8_t output){
  clearInputRaw(input);
  clearOutput(output);
  addConnectionRaw(input, output);
  update();
}
```

**Source-Code D.11:** parser.h

```
#ifndef PARSER_H_
#define PARSER_H_

#include "types.h"

extern volatile char command[100];

void parseCommand();

#endif /* PARSER_H_ */
```

**Source-Code D.12:** parser.c

```
#include "parser.h"
#include "types.h"
#include "serial.h"
#include <stdio.h>
#include "management.h"

volatile char command[100];

bool compareCommandOffset(char reference[], uint8_t offset){
  uint8_t i = 0;
  while(reference[i]){
    if(command[i+offset] != reference[i]) return false;
    i++;
  }
  return true;
}

bool compareCommand(char reference[]){
  return compareCommandOffset(reference,0);
}

void respond(bool result){
  if(result){
    sendString("done");
  }else{
    sendString("failed");
  }
}

void respondBool(bool input){
  if(input){
    sendString("true");
  }else{
    sendString("false");
  }
}

uint8_t readOutput(){
  uint8_t outnum = command[5]-'0';
  if ((outnum<1) || (outnum>8)) return 255;
  if(compareCommandOffset(":SIG",6)){
    return 2*outnum;
  }else{
    return 2*outnum+1;
  }
}

uint8_t readNumber(uint8_t offset, uint8_t min, uint8_t max){
  uint8_t temp;
  int8_t res = sscanf(&command[offset],"%u",&temp);
  if(res>0){
    if(temp>=min && temp<=max){
      return temp;
    }
  }
```

```
  return 255;
}

void parseCommand(){
  if(compareCommand("*IDN?")){
    sendString("Multiplexer␣unit\r\nby␣Thomas.Moeller@uni.kn\r\nFirmware␣3␣(19.01.2018)")
        ;
    return;
  }
  if(compareCommand(":CLE:ALL")){
    clearAll();
    respond(True);
    return;
  }
  if(compareCommand(":OPE?")){
    respondBool(operationComplete());
    return;
  }
  if(compareCommand(":CLE:INP")){
    uint8_t temp = readNumber(9,1,16);
    if (temp != 255){
      clearInput(temp);
      respond(true);
      return;
    }
    respond(false);
    return;
  }
  if(compareCommand(":CLE:OUT:SIG")){
    uint8_t temp = readNumber(13,1,8);
    if(temp != 255){
      clearOutput(2*temp);
      respond(true);
      return;
    }
    respond(false);
    return;
  }
  if(compareCommand(":CLE:OUT:SHI")){
    uint8_t temp = readNumber(13,1,8);
    if(temp != 255){
      clearOutput(2*temp+1);
      respond(true);
      return;
    }
    respond(false);
    return;
  }
  uint8_t mode = 0;
  if(compareCommand(":SIN:")) mode = 1;
  if(compareCommand(":CON:")){
    if(command[10]=='?'){
      mode = 3;
    }else{
      mode = 2;
    }
  }
  if(mode){
    uint8_t out = readOutput();
    if(out == 255){
      respond(false);
      return;
    }
    uint8_t in = readNumber(11,1,16);
    if(in == 255){
      respond(false);
      return;
    }
    if(mode == 3){
      respondBool(isConnected(in,out));
```

```
      return;
    }
    else if(mode == 2){
      addConnection(in,out);
    }
    else{
      singleConnection(in,out);
    }
    respond(true);
    return;
  }
  sendString("command␣unknown");
}
```

**Source-Code D.13:** pinout.h

```
#ifndef PINOUT_H_
#define PINOUT_H_

#include <avr/io.h>

#define PORT_RX PORTD
#define DDR_RX  DDRD
#define PIN_RX  PIND
#define P_RX  PD0

#define PORT_TX PORTD
#define DDR_TX  DDRD
#define PIN_TX  PIND
#define P_TX  PD1

#define PLEDConnected   PB0
#define PORTLEDConnected  PORTB
#define DDRLEDConnected   DDRB
#define PINLEDConnected   PINB

#define PInputUp      PA4
#define PORTInputUp     PORTA
#define DDRInputUp      DDRA
#define PINInputUp      PINA

#define PInputDown      PA5
#define PORTInputDown   PORTA
#define DDRInputDown    DDRA
#define PINInputDown    PINA

#define POutputUp     PA1
#define PORTOutputUp    PORTA
#define DDROutputUp     DDRA
#define PINOutputUp     PINA

#define POutputDown     PA0
#define PORTOutputDown    PORTA
#define DDROutputDown   DDRA
#define PINOutputDown   PINA

#define PConnect      PA7
#define PORTConnect     PORTA
#define DDRConnect      DDRA
#define PINConnect      PINA

#define PDisconnect     PA6
#define PORTDisconnect    PORTA
#define DDRDisconnect   DDRA
#define PINDisconnect   PINA

#define PSingle       PA2
#define PORTSingle      PORTA
#define DDRSingle     DDRA
```

```
#define PINSingle     PINA

#define PMultiple     PA3
#define PORTMultiple    PORTA
#define DDRMultiple     DDRA
#define PINMultiple     PINA

#define PLEDSingle      PB2
#define PORTLEDSingle   PORTB
#define DDRLEDSingle    DDRB
#define PINLEDSingle    PINB

#define PLEDMultiple    PB1
#define PORTLEDMultiple   PORTB
#define DDRLEDMultiple    DDRB
#define PINLEDMultiple    PINB

#define PSS     PB4
#define PORTSS    PORTB
#define DDRSS   DDRB
#define PINSS   PINB

#define PMOSI   PB5
#define PORTMOSI  PORTB
#define DDRMOSI   DDRB
#define PINMOSI   PINB

#define PMISO   PB6
#define PORTMISO  PORTB
#define DDRMISO   DDRB
#define PINMISO   PINB

#define PSCK    PB7
#define PORTSCK   PORTB
#define DDRSCK    DDRB
#define PINSCK    PINB

#define PRCK    PB3
#define PORTRCK   PORTB
#define DDRRCK    DDRB
#define PINRCK    PINB

#endif /* PINOUT_H_ */
```

**Source-Code D.14:** SPI.h

```
/*
 * SPI.h
 *
 *  Created on: 26.10.2017
 *      Author: thomas
 */

#ifndef SPI_H_
#define SPI_H_

#include "types.h"

extern volatile bool transmissionComplete;  //Will be set to true if the transmission is
    completed (Do not mess with the value, just read it.)

void SPItriggerTransmission();            //Triggers a new transmission of the complete
    data set.
void SPIsetDisplay(uint8_t number, uint8_t data); //Insert data for the dispay. This will
     NOT trigger a transmission.
void SPIsetOutput(uint8_t number, uint8_t data);  //Insert data for the output. This will
     NOT trigger a transmission.
void SPIInitialize();             //Intitialize hardware. Has to be called once after
    power-on
```

```
#endif /* SPI_H_ */
```

**Source-Code D.15:** SPI.c

```
#include "SPI.h"
#include "pinout.h"
#include <util/delay.h>
#include <avr/interrupt.h>

volatile uint8_t shiftRegisters[20];    //4 for the displays and 1 for each of the 16 (8
    inner and 8 shield) outputs
volatile uint8_t transmissionNumber = 0;  //Which byte is to be transmitted next
volatile bool transmissionComplete = True;  //Will be set to true if the transmission is
    completed

void transmitByte(){
  SPDR = shiftRegisters[19-transmissionNumber];
  transmissionNumber++;
}

void SPItriggerTransmission(){
  transmissionNumber = 0;
  if(!transmissionComplete) return; //If these is an ongoing transmission, restarting the
       chain at 0 is sufficient
  transmissionComplete = False;
  transmitByte();
  SPCR |= (1<<SPIE);          //Enable Interrupt
}

ISR(SPI_STC_vect){
  if(transmissionNumber<20){
    transmitByte();
  }else{
    PORTRCK |= (1<<PRCK);     //Make rising edge to transfer data to output register
    _delay_us(2);
    PORTRCK &= ~(1<<PRCK);
    SPCR &= ~(1<<SPIE);       //Disable Interrupt
    transmissionComplete = True;
  }
}

void SPIsetDisplay(uint8_t number, uint8_t data){
  shiftRegisters[number] = data;
}

void SPIsetOutput(uint8_t number, uint8_t data){
  shiftRegisters[2+number] = data;  //not 4+number since the first output is number 2
}

void SPIInitialize(){
  DDRSS |= (1<<PSS);        //Force SS Line up to not interrupt the transmission
  DDRMOSI |= (1<<PMOSI);
  DDRSCK |= (1<<PSCK);
  DDRRCK |= (1<<PRCK);
  SPCR |= (1<<SPR0);        //Frequency 3686400 / 16 = 230400Hz
  SPCR |= (1<<MSTR);        //make me "master of transmission"
  //SPCR |= (1<<CPHA);        //Data is valid on falling clock
  SPCR |= (1<<SPE);       //enable SPI
  _delay_ms(1000);
  //TODO vorbereitungen für interrupt?
}
```

**Source-Code D.16:** userInterface.h

```
/*
 * userInterface.h
 *
 *  Created on: 31.10.2017
```

```
 *      Author: thomas
 */

#ifndef USERINTERFACE_H_
#define USERINTERFACE_H_

void userInterfaceInitialize();
void userInterfaceUpdate();
void handleFlags();

#endif /* USERINTERFACE_H_ */
```

**Source-Code D.17:** userInterface.c

```
#include "userInterface.h"
#include "display.h"
#include "types.h"
#include "management.h"
#include "hardware.h"

uint8_t input = 1;
uint8_t output = 2;

void userInterfaceInitialize(){
  userInterfaceUpdate();
}

void userInterfaceUpdate(){
  displayInput(input);
  displayOutput(output);
  hardwareConnectedLED(isConnected(input,output));
  if(singleMode){
    hardwareSingleLED();
  }else{
    hardwareMultipleLED();
  }
}

void handleFlags(){
  if(flagInputUp){
    if(input < 16) input++;
    flagInputUp = false;
    userInterfaceUpdate();
  }
  if(flagInputDown){
    if(input > 1) input--;
    flagInputDown = false;
    userInterfaceUpdate();
  }
  if(flagOutputUp){
    if(output<17) output++;
    flagOutputUp = false;
    userInterfaceUpdate();
  }
  if(flagOutputDown){
    if(output > 2) output--;
    flagOutputDown = false;
    userInterfaceUpdate();
  }
  if(flagConnect){
    if(singleMode){
      singleConnection(input,output);
    }else{
      addConnection(input,output);
    }
    flagConnect = false;
    userInterfaceUpdate();
  }
  if(flagDisconnect){
```

```
    removeConnection(input,output);
    flagDisconnect = false;
    userInterfaceUpdate();
  }
  if(flagMultiple){
    singleMode = False;
    flagMultiple = False;
    userInterfaceUpdate();
  }
  if(flagSingle){
    clearAll();
    flagSingle = False;
    userInterfaceUpdate();
  }
}
```

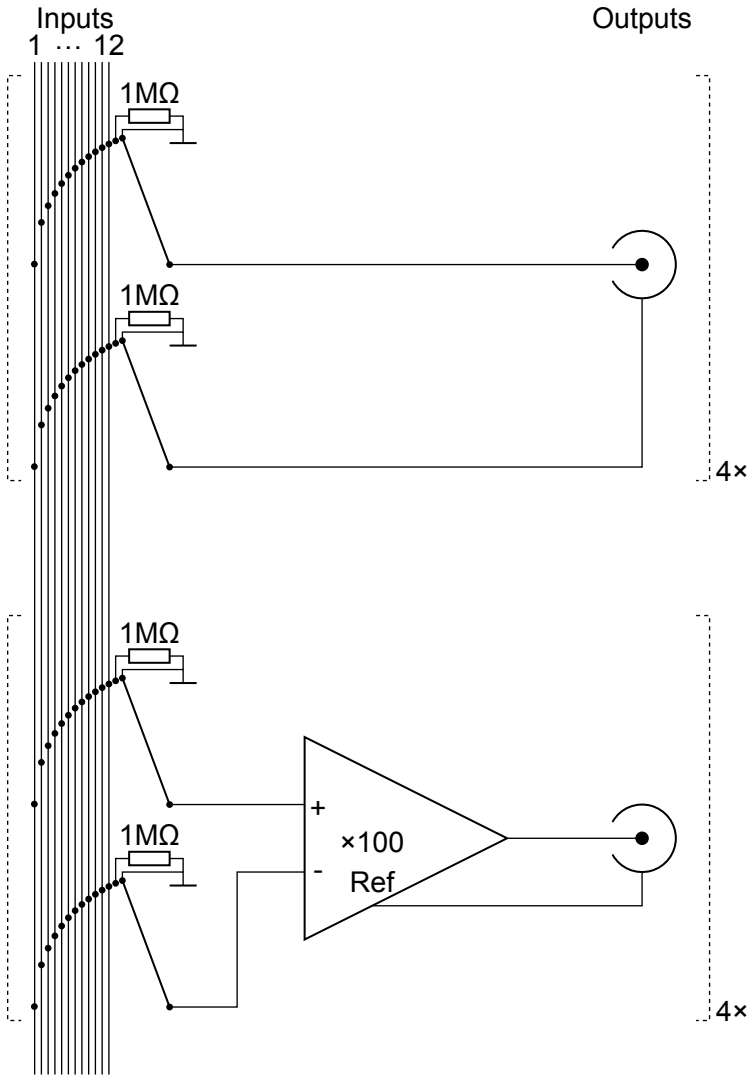



**Figure D.7.:** Simplified block diagram of the multiplexer NIM insert

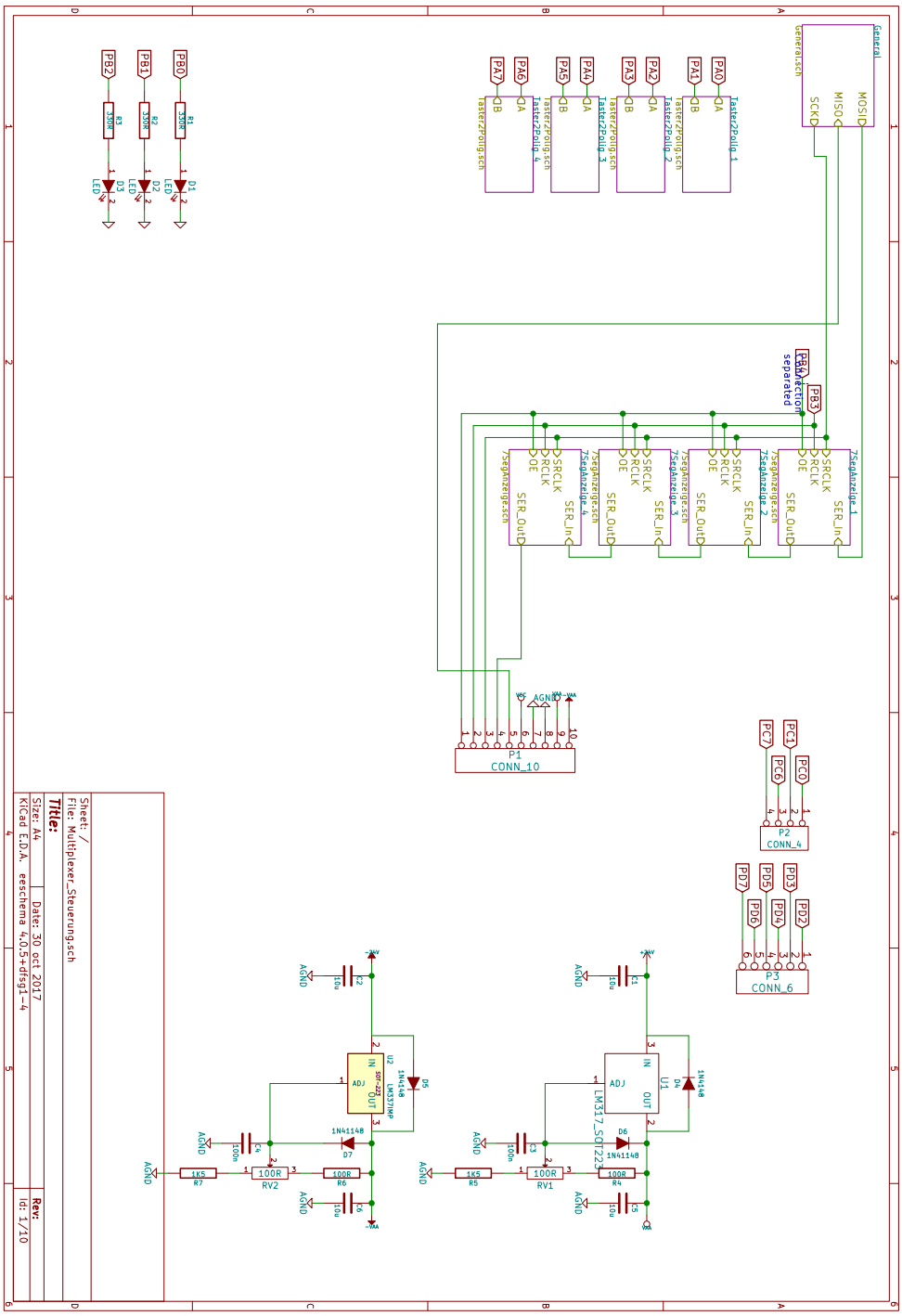



**Figure D.8.:** Schematic of the control board in the multiplexer NIM insert

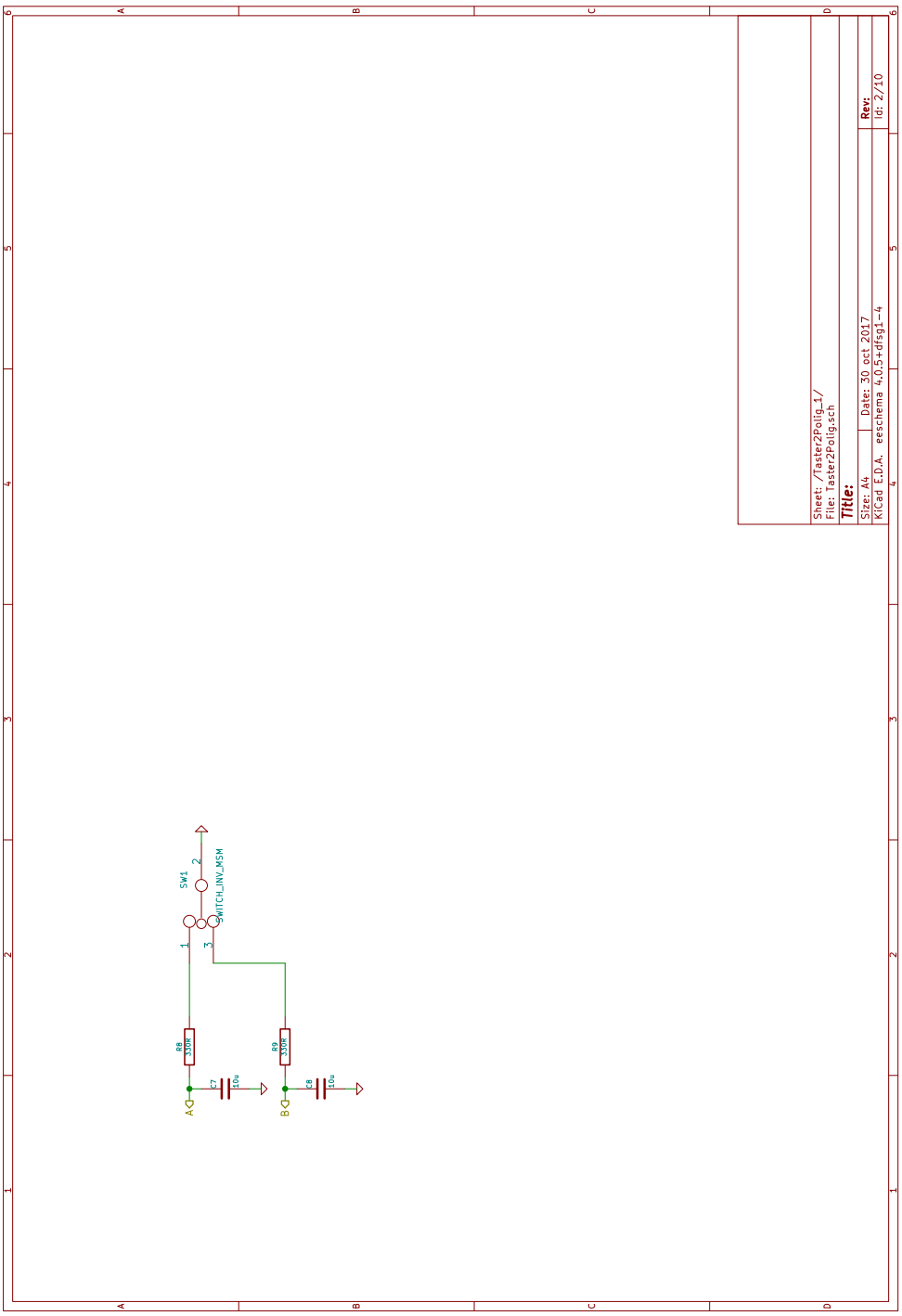



**Figure D.9.:** Schematic of the control board in the multiplexer NIM insert

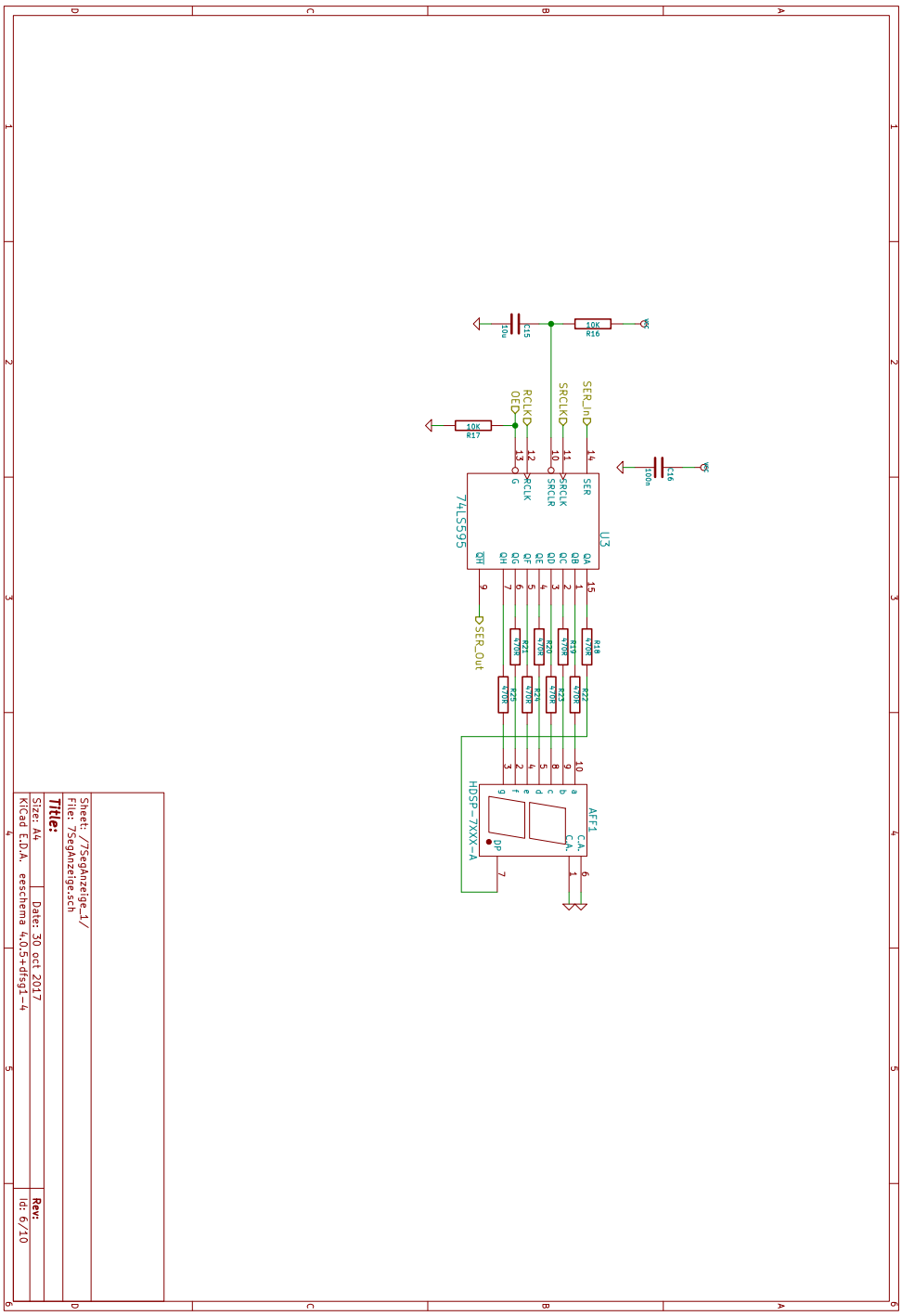



**Figure D.10.:** Schematic of the control board in the multiplexer NIM insert

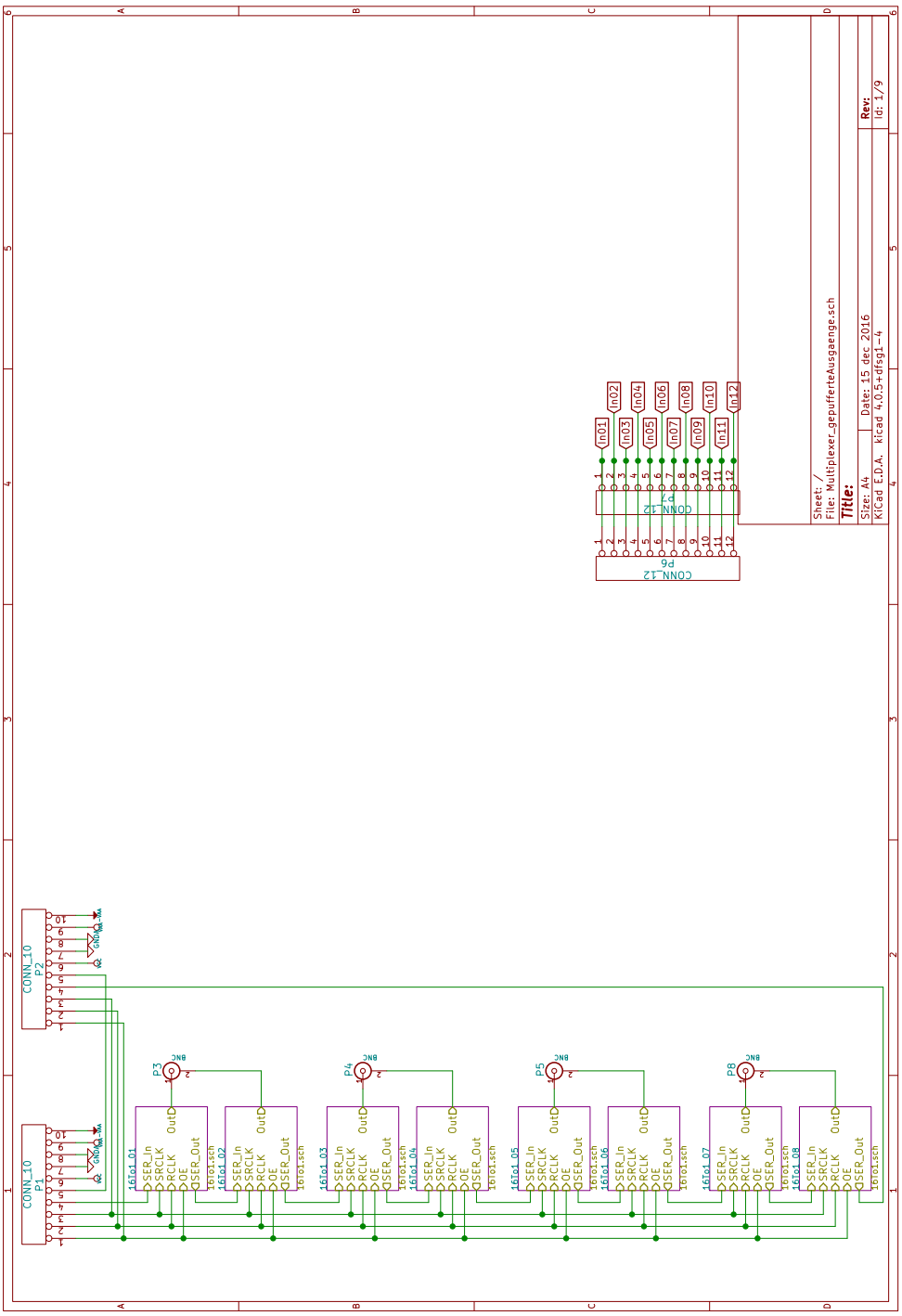



**Figure D.11.:** Schematic of the switching board for the normal outputs in the multiplexer NIM insert

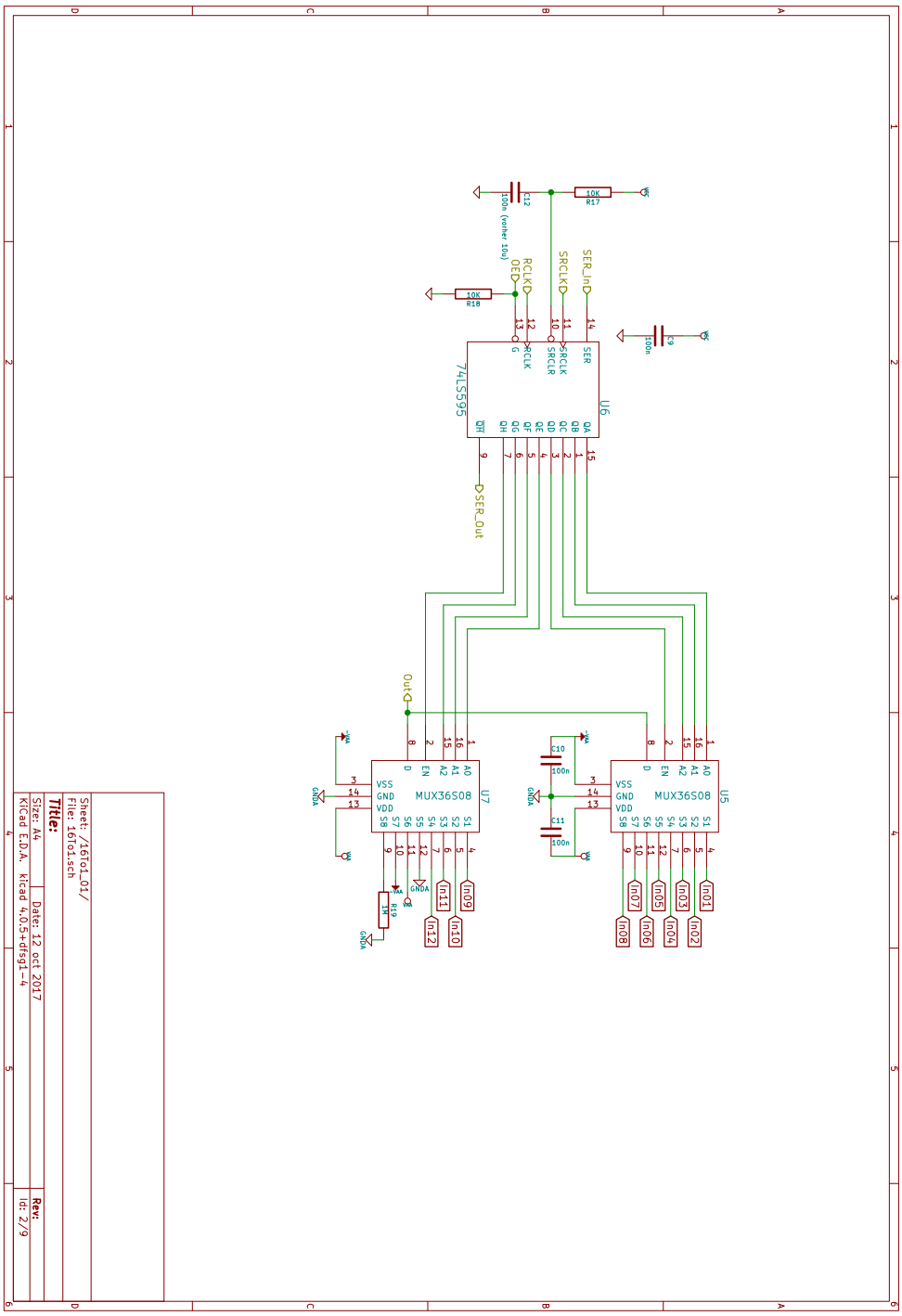



**Figure D.12.:** Schematic of the switching board for the normal outputs in the multiplexer NIM insert

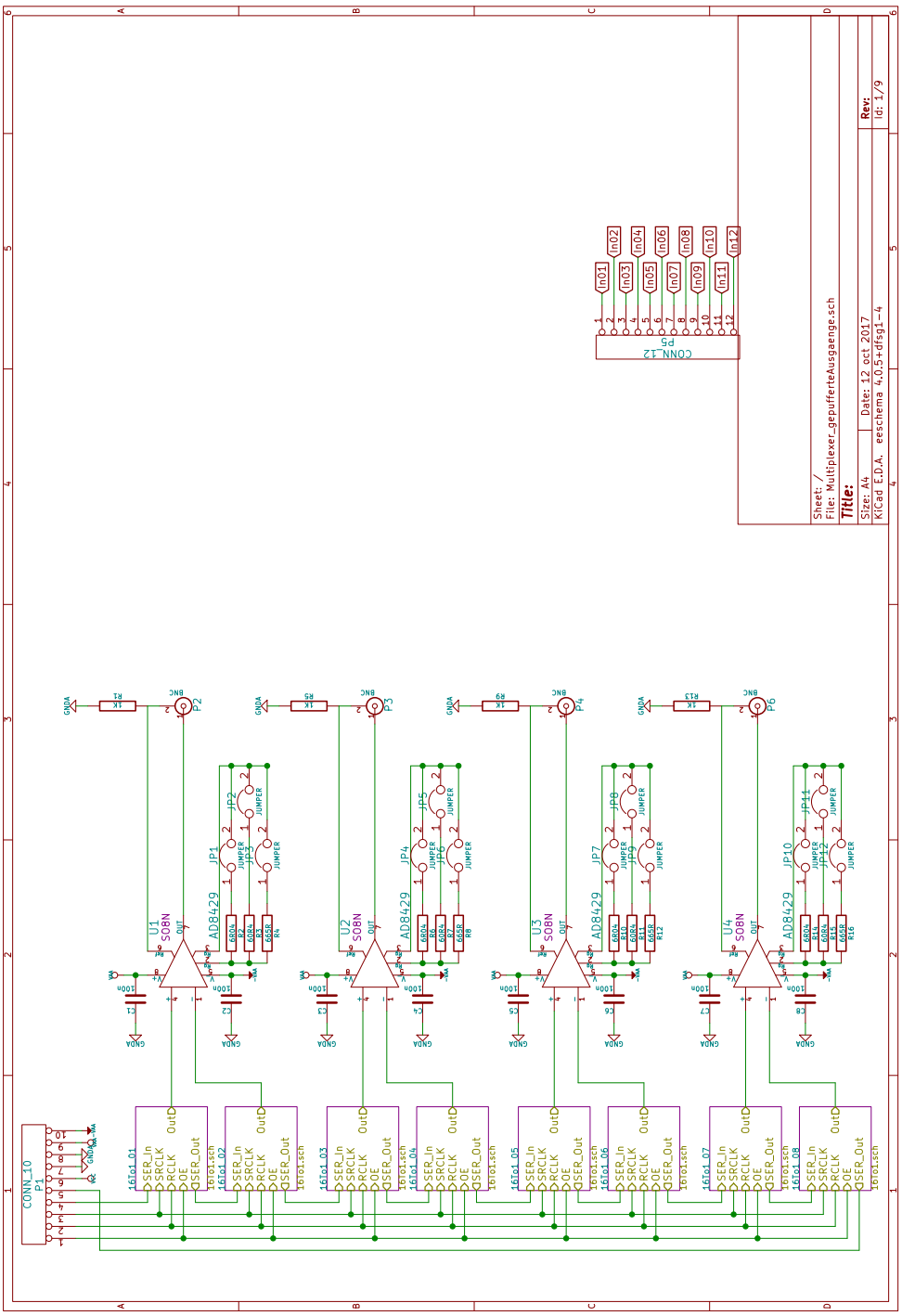



**Figure D.13.:** Schematic of the switching board for the buffered and amplified outputs in the multiplexer NIM insert

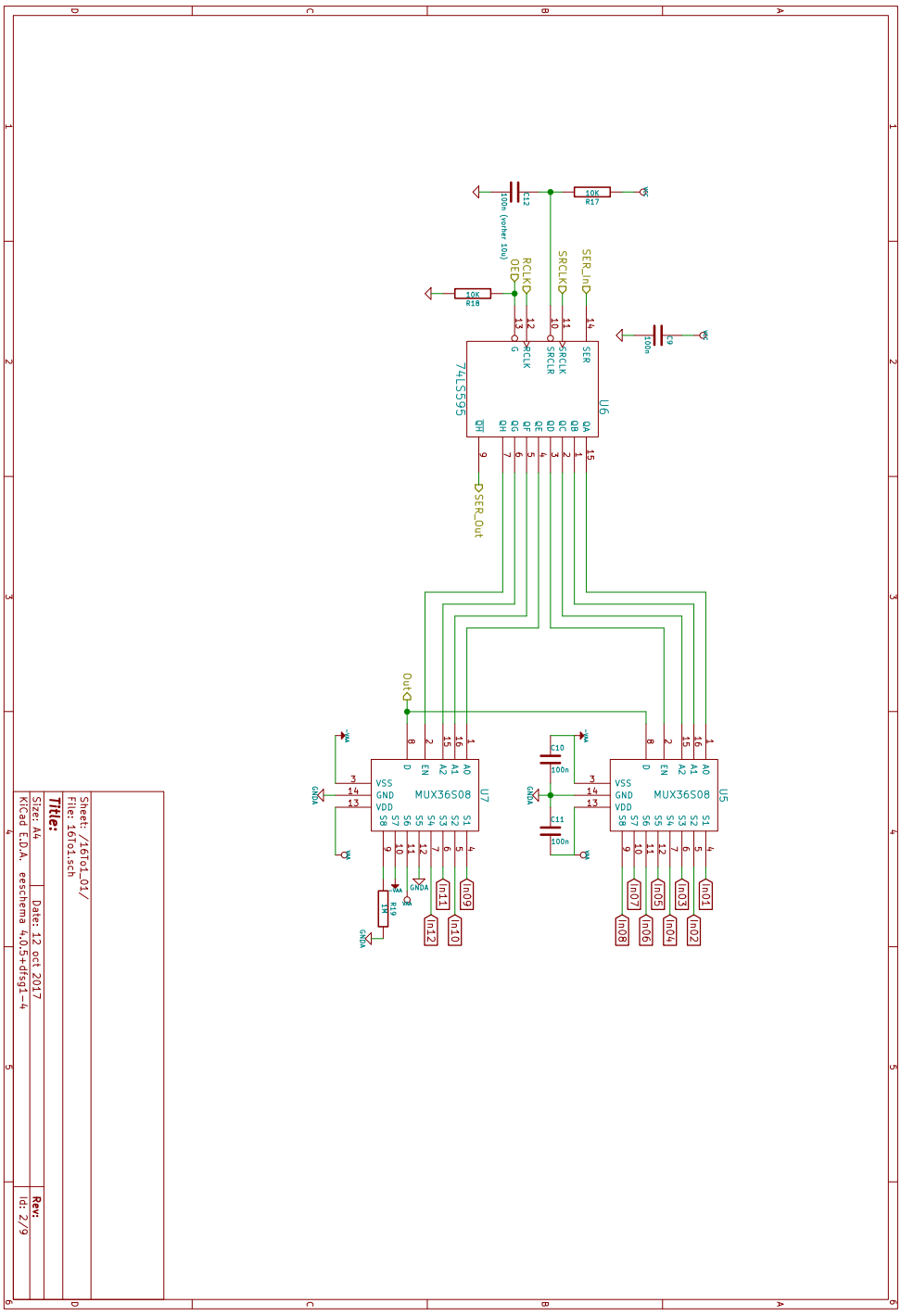
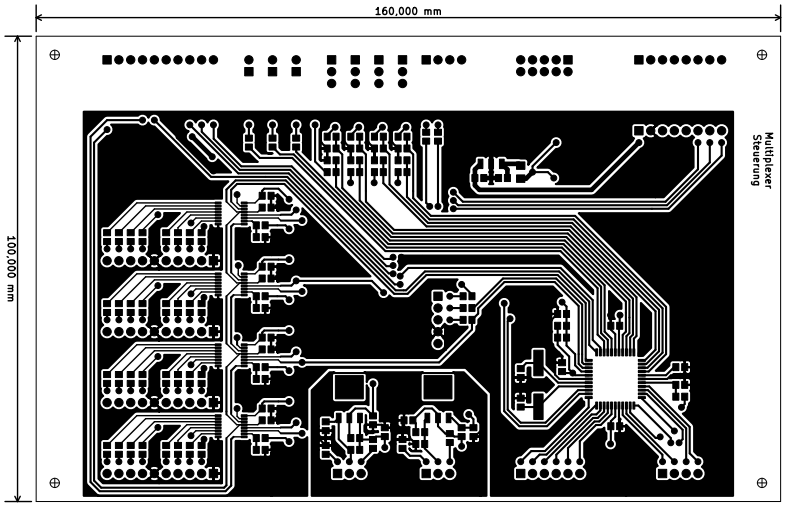
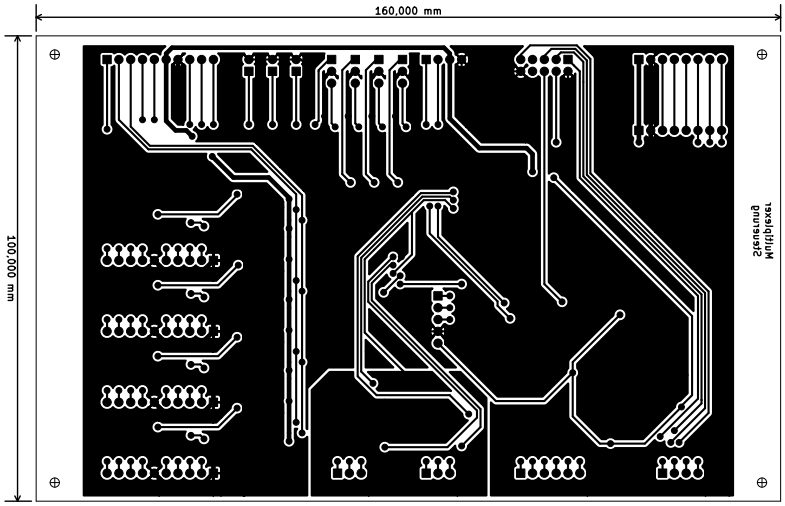


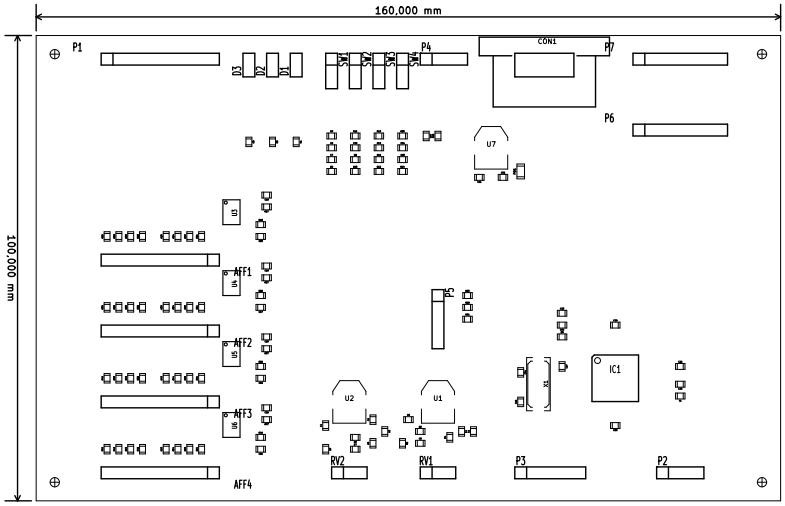

**Figure D.14.:** Schematic of the switching board for the buffered and amplified outputs in the multiplexer NIM insert




**(a)** Top side copper structure

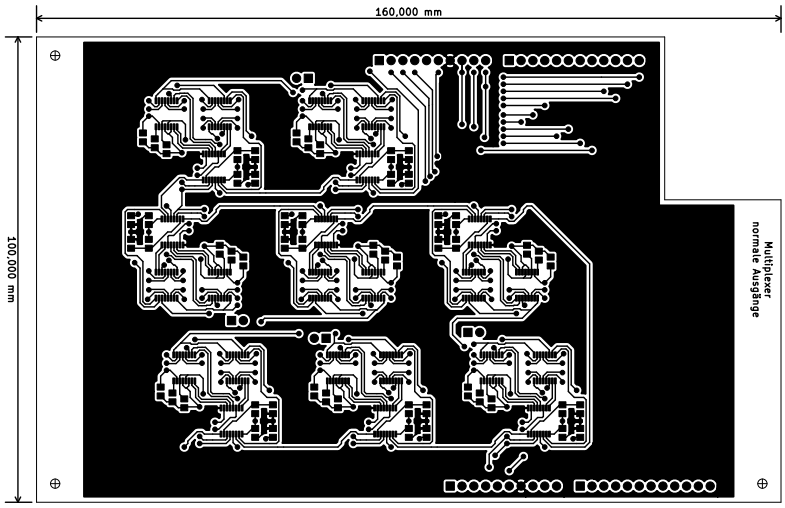



**(b)** Bottom side copper structure

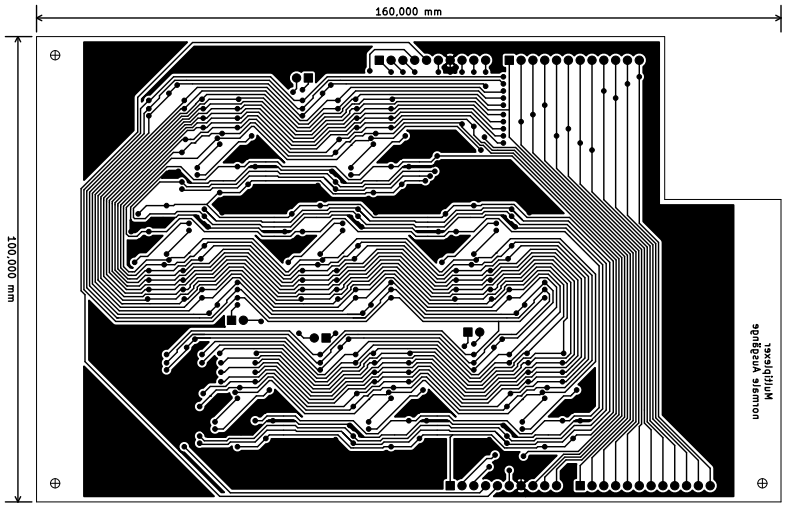



**(c)** Component references

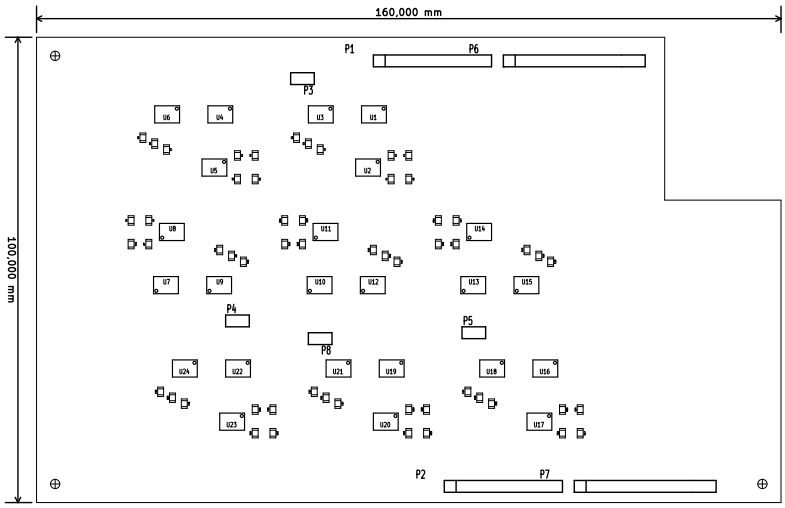
**Figure D.15.:** Circuit board layout of the control board in the multiplexer NIM insert




**(a)** Top side copper structure

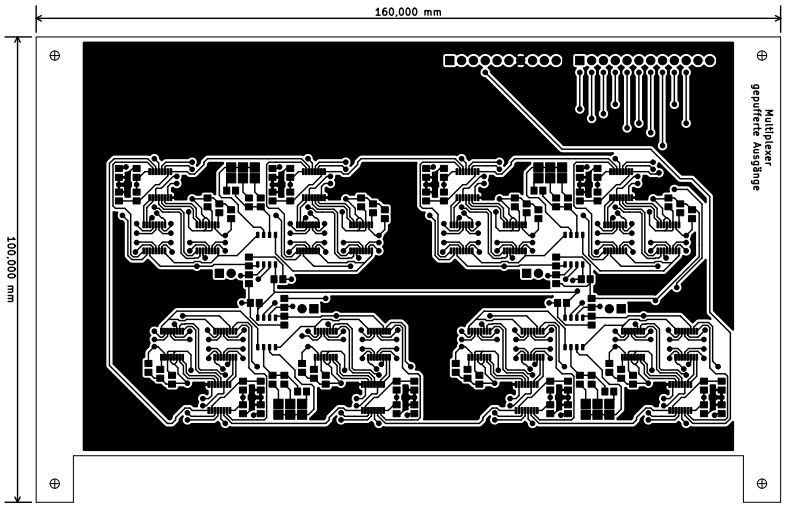



**(b)** Bottom side copper structure

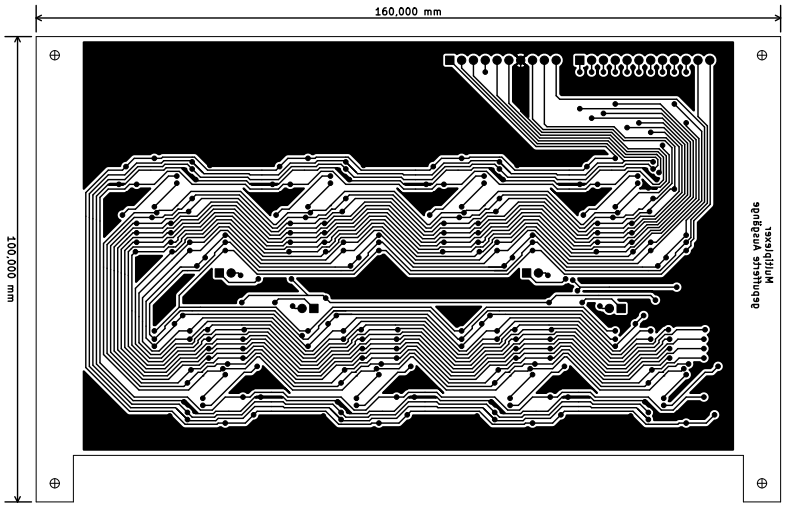



**(c)** Component references

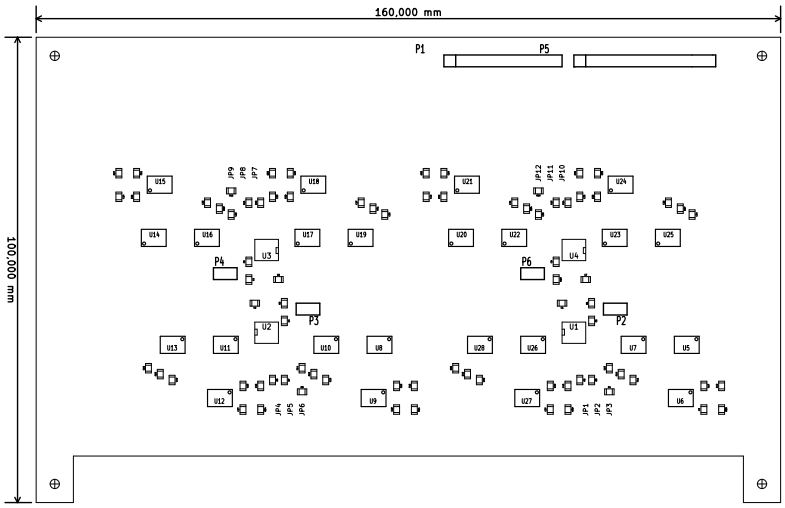
**Figure D.16.:** Circuit board layout of the switching board for the normal outputs in the multiplexer NIM insert




**(a)** Top side copper structure

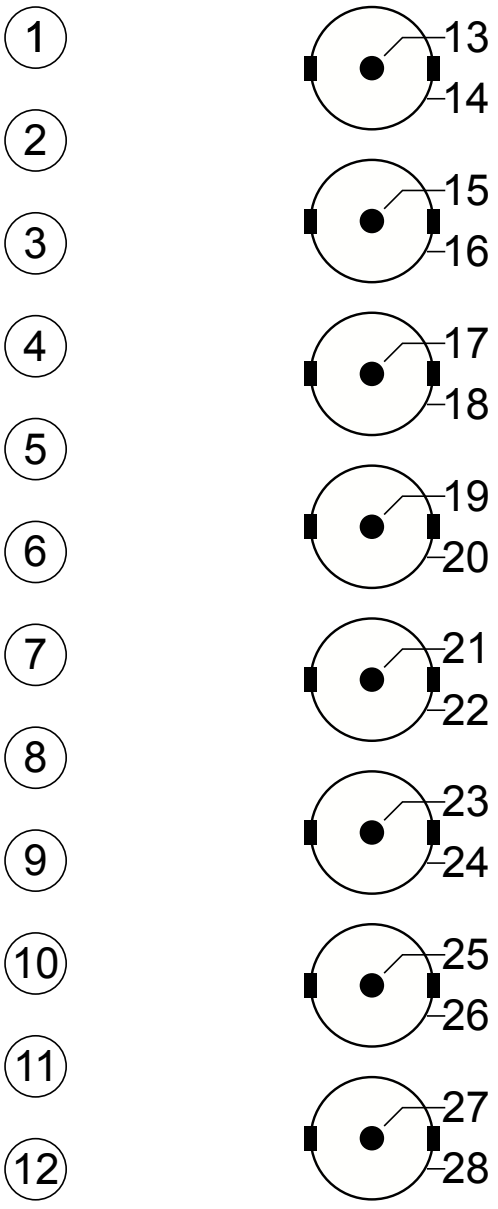



**(b)** Bottom side copper structure




**(c)** Component references

**Figure D.17.:** Circuit board layout of the switching board for the buffered and amplified outputs in the multiplexer NIM insert




| | | |
|---|---|---|
| 1 | Input 1 | input |
| 2 | Input 2 | input |
| 3 | Input 3 | input |
| 4 | Input 4 | input |
| 5 | Input 5 | input |
| 6 | Input 6 | input |
| 7 | Input 7 | input |
| 8 | Input 8 | input |
| 9 | Input 9 | input |
| 10 | Input 10 | input |
| 11 | Input 11 | input |
| 12 | Input 12 | input |
| 13 | Output 1 signal | output |
| 14 | Output 1 shield | output |
| 15 | Output 2 signal | output |
| 16 | Output 2 shield | output |
| 17 | Output 3 signal | output |
| 18 | Output 3 shield | output |
| 19 | Output 4 signal | output |
| 20 | Output 4 shield | output |
| 21 | Output 5 signal | output |
| 22 | Output 5 shield | input/output |
| 23 | Output 6 signal | output |
| 24 | Output 6 shield | input/output |
| 25 | Output 7 signal | output |
| 26 | Output 7 shield | input/output |
| 27 | Output 8 signal | output |
| 28 | Output 8 shield | input/output |

**Figure D.18.:** Pin configuration of the multiplexer NIM insert


**(a)** front **(b)** back

**Figure D.19.:** Photos of the outside of the multiplexer NIM insert

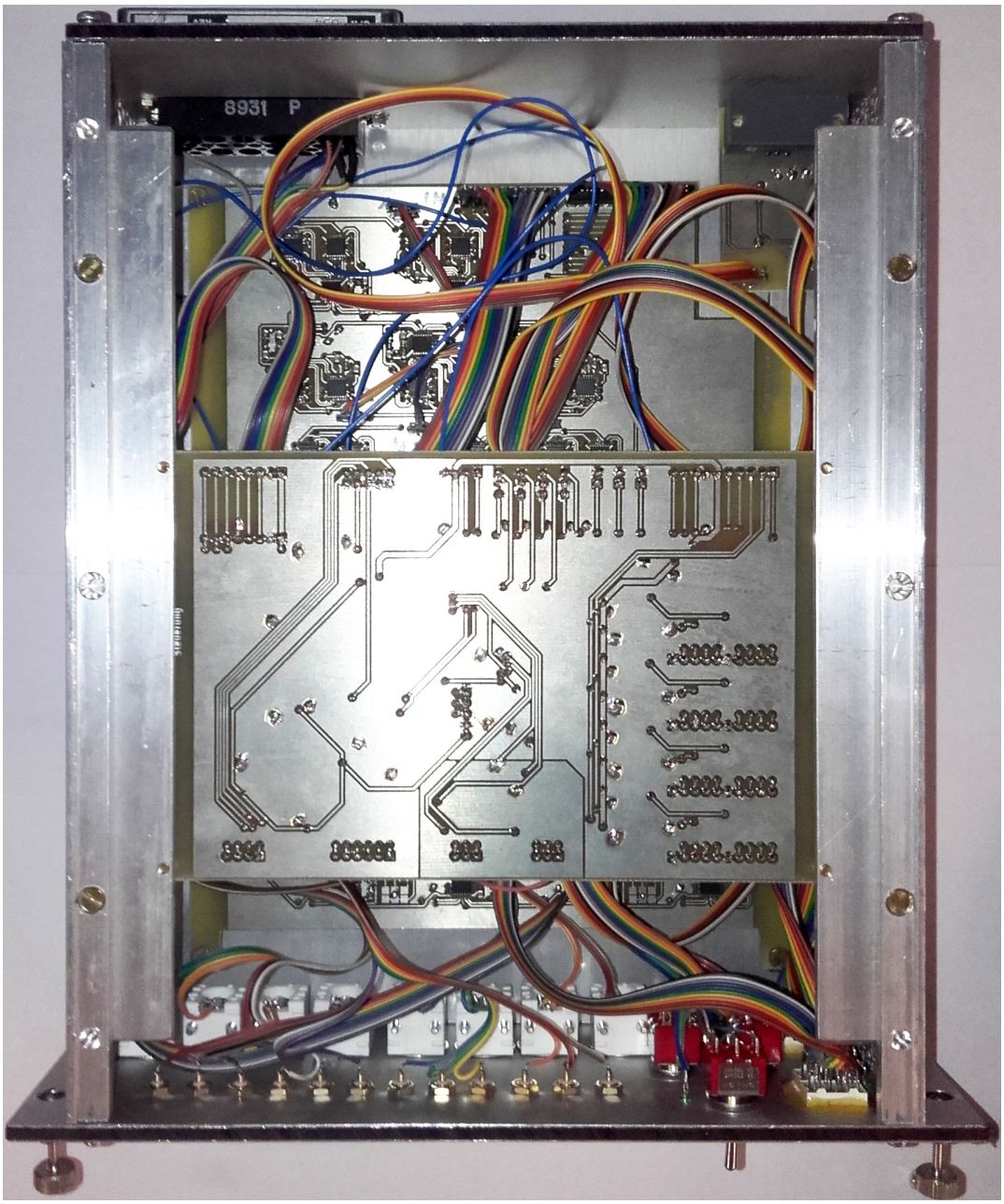



**Figure D.20.:** Photo of the inside of the multiplexer NIM insert

## D.3. Source

The source insert serves two purposes. It serves as an adapter between the 6-pin LEMO plug for the sense IceCube insert and three isolated BNC plugs at the front. These plugs provide an input for the bias voltage and current measurement outputs.
The other purpose is that it contains a combination of differential amplifiers and resistive dividers. These give three inputs which are divided by 1, 10, and 100. The output is the sum of the divided inputs. The benefit of using this board is that large input signals can be used, which reduces the noise of the external devices. Additionally all inputs are differential to prevent ground loops.
In normal operating conditions, the output of the adder is looped back directly into the input of the bias voltage.
A block diagram of the signal processing part of the insert is shown in figure D.21. The full schematic of the circuit board is shown in figure D.22 and the corresponding board layouts in figure D.23. The pinout of the connectors is shown in figure D.24 and photos of the front, back, and inside are shown in figures D.25 and D.26.

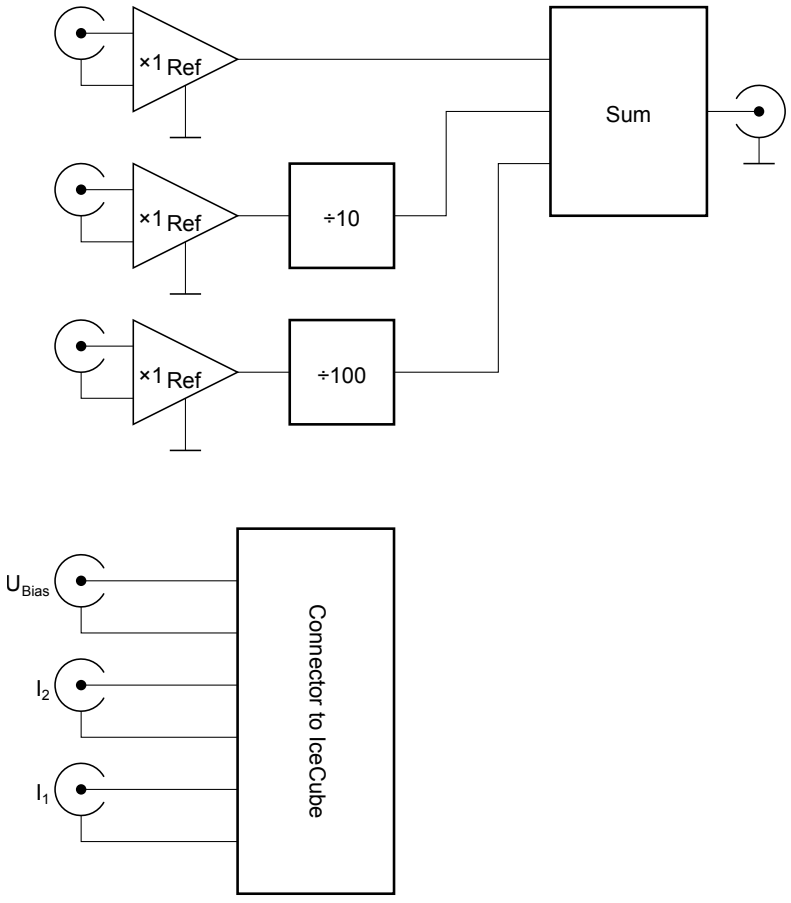



**Figure D.21.:** Simplified block diagram of the source NIM insert

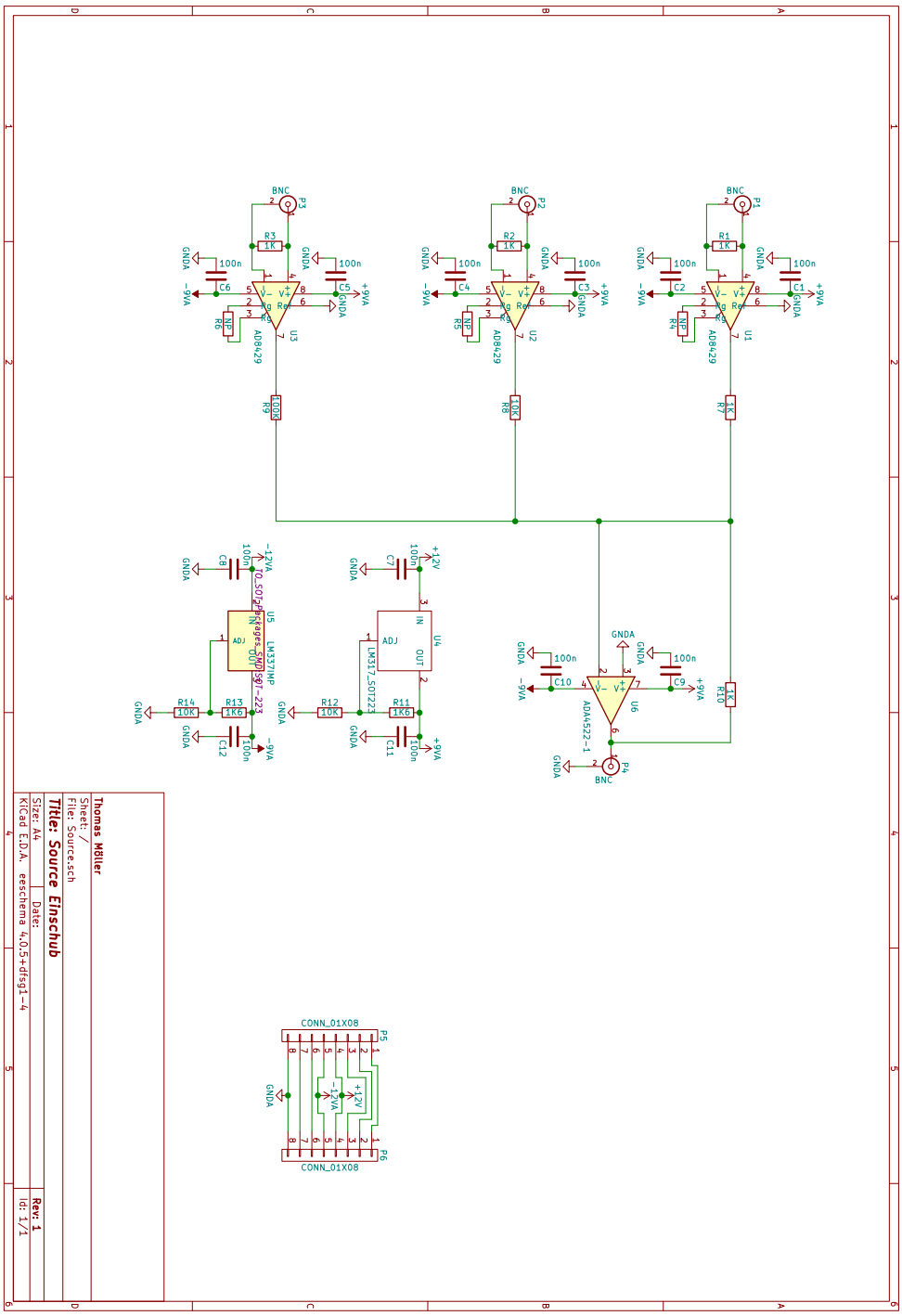
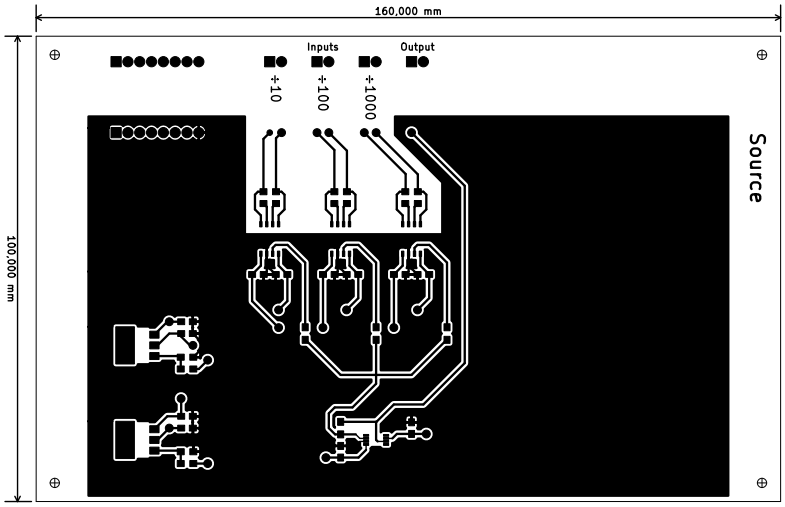
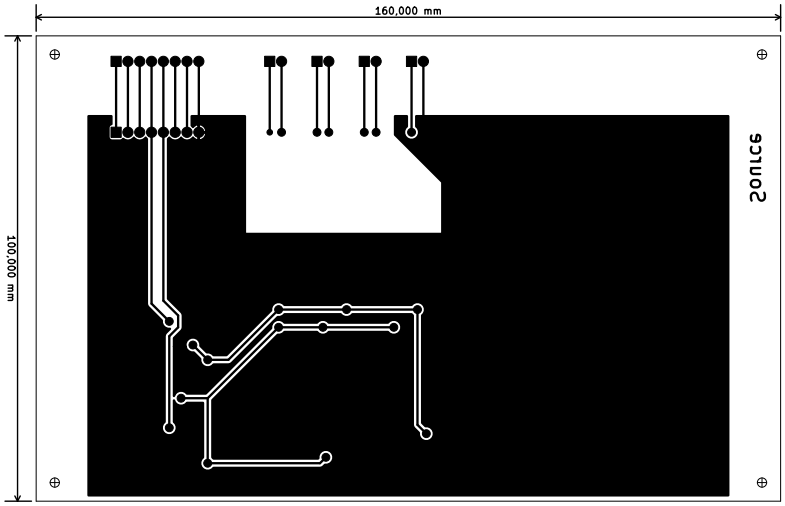


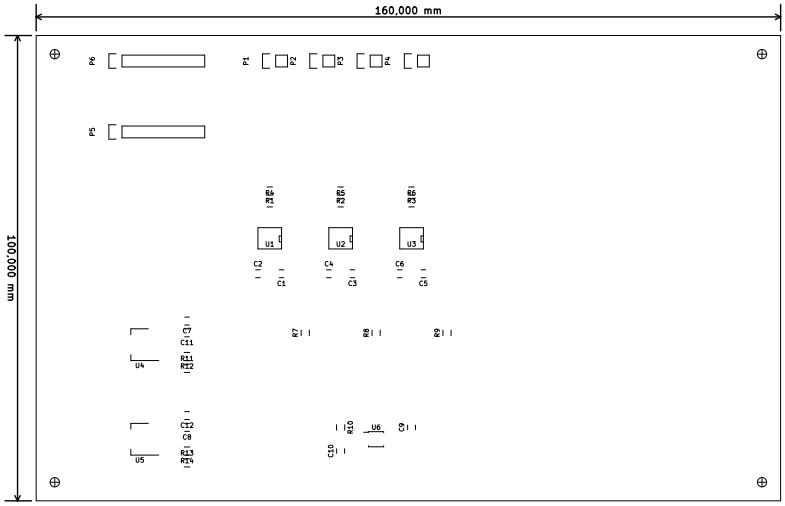

**Figure D.22.:** Schematic of the source NIM insert




**(a)** Top side copper structure

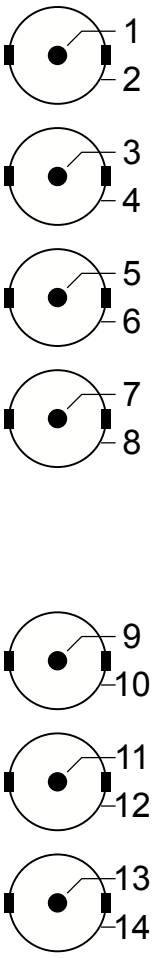



**(b)** Bottom side copper structure



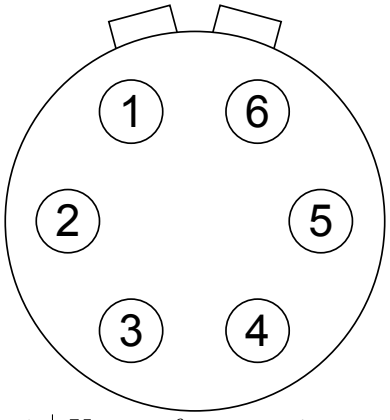

**(c)** Component references

**Figure D.23.:** Circuit board layout of the source NIM insert




| | | |
|---|---|---|
| 1 | Input ÷1 signal | input |
| 2 | Input ÷1 reference | input |
| 3 | Input ÷10 signal | input |
| 4 | Input ÷10 reference | input |
| 5 | Input ÷100 signal | input |
| 6 | Input ÷100 reference | input |
| 7 | Output sum signal | output |
| 8 | Output sum ground | output |
| 9 | $U_{Bias}$ signal | input |
| 10 | $U_{Bias}$ reference | input |
| 11 | $I_2$ signal | output |
| 12 | $I_2$ reference | output |
| 13 | $I_1$ signal | output |
| 14 | $I_1$ reference | output |

**(a)** front




| | | |
|---|---|---|
| 1 | $U_{Bias}$ reference | input |
| 2 | $I_2$ reference | output |
| 3 | $I_2$ signal | output |
| 4 | $I_1$ signal | output |
| 5 | $I_1$ reference | output |
| 6 | $U_{Bias}$ signal | input |

**(b)** back

**Figure D.24.:** Pin configuration of the source NIM insert


**(a)** front **(b)** back

**Figure D.25.:** Photos of the outside of the source NIM insert

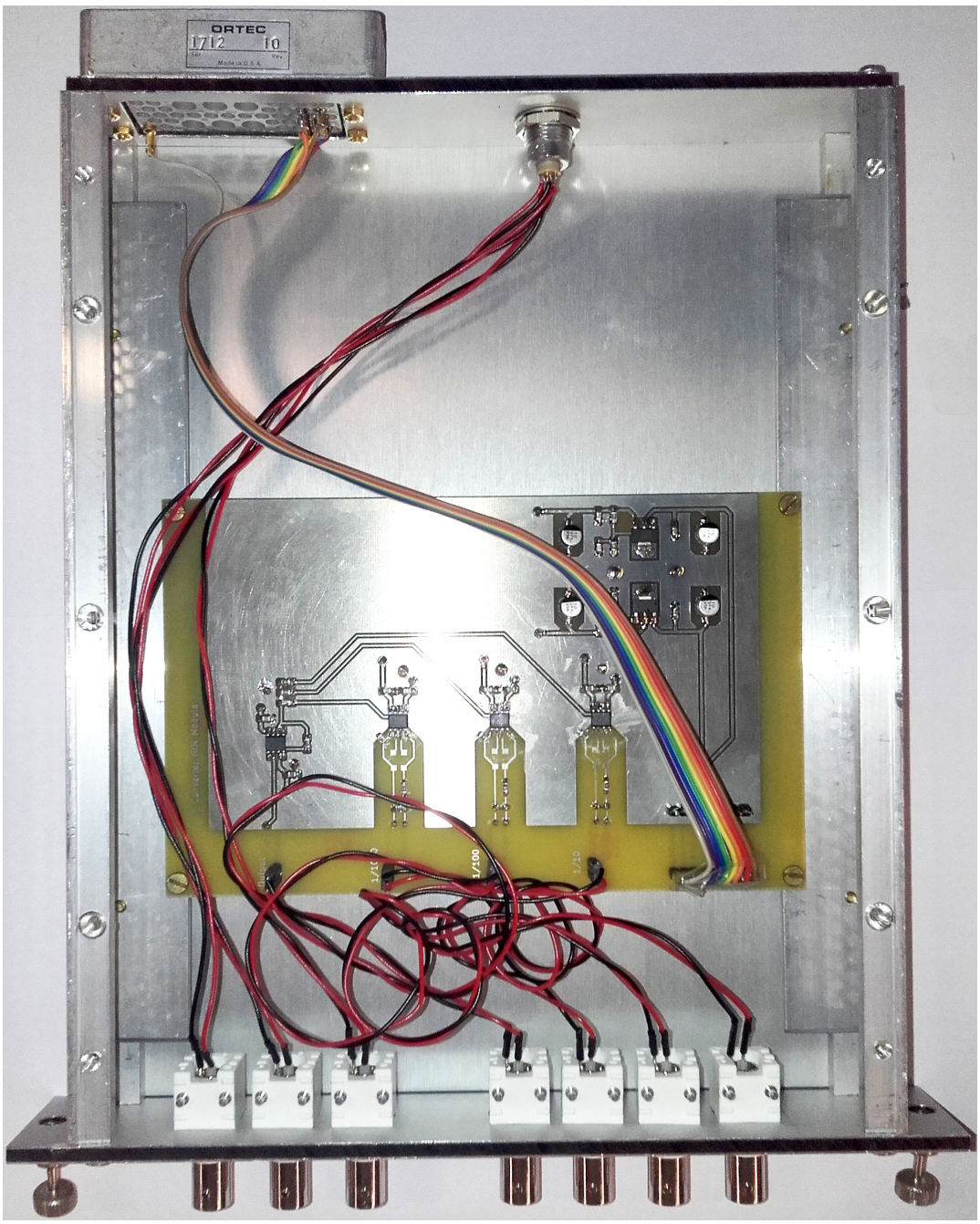



**Figure D.26.:** Photo of the inside of the source NIM insert

## D.4. IceCube Supply

The main purpose of the IceCube supply insert is to provide the clean symmetric ±24 V supply rail of the NIM crate to the IceCube.
Since the power supply insert in the IceCube (see appendix C.1 on page 143) contains electronics for checking the temperature and the current consumption, these signals are fed back to the NIM module. There, the analog signals are digitized and are available in the software.
The digitizer has additional inputs which are used to measure the analog outputs of the two pressure sensors for the helium bath and the inner tube.
To get the analog values, the insert handles the following commands.

- `*IDN?`
  This command returns the name of the insert, the author, the version, and date of the firmware. The output is three lines long.

- `:TEMperature?`
  This command returns the temperature in the format `%05.2fdegC` in °C.

- `:CURrent:POSitive?`
  This command returns the current measured on the positive rail in the format `%06.3fmA` in mA.

- `:CURrent:NEGative?`
  This command returns the current measured on the negative rail in the format `%06.3fmA` in mA.

- `:REAding?␣<Number>`
  This command returns the raw reading of the channel with the given number. The output is an unsigned five digit integer.

A block diagram of the module is shown in figure D.27. The full schematics of the circuit boards are shown in figures D.28 to D.30 and the corresponding board layouts in figures D.31 and D.32. The pinout of the connectors is shown in figure D.33 and photos of the front, back, and inside are shown in figures D.34 and D.35. The firmware consists of the following files.

**Source-Code D.18:** adc.h

```
#ifndef ADC_H_
#define ADC_H_

#include "types.h"

extern volatile uint16_t positiveReading;   //right aligned => 1024 is 5V
extern volatile uint16_t negativeReading; //right aligned => 1024 is 5V

void adcInitialize();

#endif /* ADC_H_ */
```

**Source-Code D.19:** adc.h

```
#include "adc.h"
```

```
#include "pinout.h"
#include <avr/interrupt.h>

volatile uint16_t positiveReading;  //right aligned => 1024 is 5V
volatile uint16_t negativeReading;  //right aligned => 1024 is 5V

void adcInitialize(){
  ADMUX |= (1<<REFS0); //5V Referez
  ADMUX &= ~((1<<MUX4) | (1<<MUX3) | (1<<MUX2) | (1<<MUX1) | (1<<MUX0));
  ADMUX |= (PCurrentNegative<<MUX0); //Eingang auf negativ In stellen
  ADCSRA |= (1<<ADPS2) | (1<<ADPS0); //ADC Freqenz = 3686400Hz / 32 = 115200Hz
  ADCSRA |= (1<<ADEN); //ADC anschalten
  ADCSRA |= (1<<ADIE); //ADC Interrupt anschalten
  ADCSRA |= (1<<ADSC); //ADC starten
}

ISR(ADC_vect){
  if((ADMUX & ((1<<MUX4) | (1<<MUX3) | (1<<MUX2) | (1<<MUX1) | (1<<MUX0))) == (
      PCurrentNegative<<MUX0)){
    negativeReading = ADC;
    ADMUX &= ~((1<<MUX4) | (1<<MUX3) | (1<<MUX2) | (1<<MUX1) | (1<<MUX0));
    ADMUX |= (PCurrentPositive<<MUX0); //Eingang auf positiv In
  }else{
    positiveReading = ADC;
    ADMUX &= ~((1<<MUX4) | (1<<MUX3) | (1<<MUX2) | (1<<MUX1) | (1<<MUX0));
    ADMUX |= (PCurrentNegative<<MUX0); //Eingang auf negativ In
  }
  ADCSRA |= (1<<ADSC); //ADC starten
}
```

**Source-Code D.20:** externalADC.h

```
#ifndef EXTERNALADC_H_
#define EXTERNALADC_H_

#include <avr/io.h>

extern volatile float temperature;
extern volatile uint16_t readings[];

void externalADCInitialize();
void externalADCRead();

#endif /* EXTERNALADC_H_ */
```

**Source-Code D.21:** externalADC.c

```
#include "externalADC.h"

#include <util/delay.h>
#include "pinout.h"
#include "types.h"

volatile float temperature;
volatile uint16_t readings[8];

uint8_t input = 0;

uint16_t ADCRead(){
  PORTSPICSADC &= ~(1<<PSPICSADC);  //pull CS line low
  _delay_us(1);
  SPDR = 0;           //send worthless data
  while(~SPSR & (1<<SPIF));   //wait for transmission to complete
  uint16_t reading;
  reading = SPDR << 8;//read high byte
  SPDR = 0;           //send worthless data
  while(~SPSR & (1<<SPIF));   //wait for transmission to complete
  //uint8_t buffer = SPDR;
  reading |= SPDR;        //read low byte
```

```
  PORTSPICSADC |= (1<<PSPICSADC); //pull CS high
  return reading;
}

void externalADCInitialize(){
  DDRSPIMOSI |= (1<<PSPIMOSI);  //MOSI should be an output
  DDRSPISCK |= (1<<PSPISCK);    //SCK must be an output
  PORTSPICSADC |= (1<<PSPICSADC); //Pull CS line up
  DDRSPICSADC |= (1<<PSPICSADC);  //and force it there
  DDRSPISS |= (1<<PSPISS);    //make SS an output to prevent it from interfering with SPI
      transmission
  SPCR |= (1<<SPR0);        //Frequency 3686400 / 16 = 230400Hz
  SPCR |= (1<<MSTR);        //make me "master of transmission"
  SPCR |= (1<<SPE);       //enable SPI
  DDRMUXA0 |= (1<<PMUXA0);    //turn MUXA0 control line into output
  DDRMUXA1 |= (1<<PMUXA1);    //turn MUXA1 control line into output
  DDRMUXA2 |= (1<<PMUXA2);    //turn MUXA2 control line into output
  PORTMUXEN |= (1<<PMUXEN);   //Pull MUXEN high
  DDRMUXEN |= (1<<PMUXEN);    //and force it there
  _delay_ms(24);          //make sure the first conversion took place
  ADCRead();            //and discard the value
  _delay_ms(24);          //and make sure the next conversion is done
}

void externalADCRead(){
  //uint8_t oldinput = input;
  //uint16_t rawreading;
  if(++input >= 8) input = 0;
  //Set new MUX setting
  if(input&(1<<0)){
    PORTMUXA0 |= (1<<PMUXA0);
  }else{
    PORTMUXA0 &= ~(1<<PMUXA0);
  }
  if(input&(1<<1)){
    PORTMUXA1 |= (1<<PMUXA1);
  }else{
    PORTMUXA1 &= ~(1<<PMUXA1);
  }
  if(input&(1<<2)){
    PORTMUXA2 |= (1<<PMUXA2);
  }else{
    PORTMUXA2 &= ~(1<<PMUXA2);
  }
  _delay_ms(1);
  ADCRead();
  _delay_ms(24); //Make sure that the new converison is complete
  ADCRead();
  _delay_ms(24); //Make sure that the new converison is complete
  readings[input] = ADCRead();
  if (input == 0){
    //temperature = (2637.0-readings[0])/13.6; //see http://www.ti.com/lit/ds/symlink/
        lmt87.pdf equation 6
    temperature = (2637.0-(readings[0]/32.768*5-5000.0))/13.6; //see http://www.ti.com/
        lit/ds/symlink/lmt87.pdf equation 6
  }
}
```

**Source-Code D.22:** hardware.h

```
#ifndef HARDWARE_H_
#define HARDWARE_H_

void enableGreenLED();
void disableGreenLED();

void enableRedLED();
void disableRedLED();
```

```
void hardwareInitialize();

#endif /* HARDWARE_H_ */
```

**Source-Code D.23:** hardware.c

```
#include "pinout.h"

void enableGreenLED(){
  PORTLEDGreen |= (1<<PLEDGreen);
}
void disableGreenLED(){
  PORTLEDGreen &= ~(1<<PLEDGreen);
}

void enableRedLED(){
  PORTLEDRed |= (1<<PLEDRed);
}
void disableRedLED(){
  PORTLEDRed &= ~(1<<PLEDRed);
}

void hardwareInitialize(){
  DDRLEDRed |= (1<<PLEDRed);
  DDRLEDGreen |= (1<<PLEDGreen);
}
```

**Source-Code D.24:** main.h

```
#ifndef MAIN_H_
#define MAIN_H_

#include "types.h"

extern bool requestPicture;

bool startExposure();
bool stopExposure();
bool getExposure();
bool startFocus();
bool stopFocus();
bool getFocus();

#endif /* MAIN_H_ */
```

**Source-Code D.25:** main.c

```
#include "serial.h"
#include <avr/interrupt.h>
#include "parser.h"
#include "types.h"
#include <util/delay.h>
#include "main.h"

#include "pinout.h"
#include "adc.h"
#include "externalADC.h"
#include "hardware.h"

int main(){
  externalADCInitialize();
  initializeSerialPort();
  adcInitialize();
  hardwareInitialize();
  sei();
  while(True){
    _delay_ms(20);
    //if((positiveReading>10) || (negativeReading > 10)){
```

```
    if(positiveReading>10){
      enableGreenLED();
    }else{
      disableGreenLED();
    }
    externalADCRead();
    if(temperature>50.0){
      enableRedLED();
    }else{
      disableRedLED();
    }
  }
}
```

**Source-Code D.26:** parser.h

```
#ifndef PARSER_H_
#define PARSER_H_

#include "types.h"

extern volatile char command[100];

void parseCommand();

#endif /* PARSER_H_ */
```

**Source-Code D.27:** parser.c

```
#include "parser.h"
#include "types.h"
#include "serial.h"
#include "main.h"
#include <stdio.h>
#include "adc.h"
#include "externalADC.h"
#include "pinout.h"

volatile char command[100];
//volatile char reading = '0';

bool compareCommand(char reference[]){
  uint8_t i = 0;
  while(reference[i]){
    if(command[i] != reference[i]) return false;
    i++;
  }
  return true;
}

void respond(bool result){
  if(result){
    sendString("done");
  }else{
    sendString("failed");
  }
}

void parseCommand(){
  char out[] = {0,0,0,0,0,0,0,0,0,0,0,0};
  if(compareCommand("*IDN?")){
    sendString("Power␣supply␣unit\r\nby␣Thomas.Moeller@uni.kn\r\nFirmware␣5␣(01.10.2018)"
        );
    return;
  }
  if(compareCommand(":TEM?")){
    sprintf(out,"%05.2fdegC",temperature);
    sendString(out);
    return;
```

```
  }
  if(compareCommand(":CUR:POS?")){
    sprintf(out,"%06.3fmA",positiveReading/1024.0*5*mAperV);
    sendString(out);
    return;
  }
  if(compareCommand(":CUR:NEG?")){
    sprintf(out,"%06.3fmA",negativeReading/1024.0*5*mAperV);
    sendString(out);
    return;
  }
    if(compareCommand(":REA? ")){
        uint8_t temp;
        int8_t ret = sscanf(&command[6],"%u",&temp);
        if((ret <= 0) || (temp >= 8)){
          respond(false);
        }else{
          sprintf(out,"%05u",readings[temp]);
          sendString(out);
        }
        return;
    }
  sendString("command unknown");
}
```

**Source-Code D.28:** pinout.h

```
#ifndef PINOUT_H_
#define PINOUT_H_

#include <avr/io.h>

#define PORT_RX PORTD
#define DDR_RX  DDRD
#define PIN_RX  PIND
#define P_RX  PD0

#define PORT_TX PORTD
#define DDR_TX  DDRD
#define PIN_TX  PIND
#define P_TX  PD1

#define PLEDGreen     PC0
#define PORTLEDGreen    PORTC
#define DDRLEDGreen     DDRC
#define PINLEDGreen     PINC

#define PLEDRed       PC1
#define PORTLEDRed      PORTC
#define DDRLEDRed     DDRC
#define PINLEDRed     PINC

#define PSPIMOSI      PB5
#define PORTSPIMOSI     PORTB
#define DDRSPIMOSI      DDRB
#define PINSPIMOSI      PINB

#define PSPIMISO      PB6
#define PORTSPIMISO     PORTB
#define DDRSPIMISO      DDRB
#define PINSPIMISO      PINB

#define PSPISCK       PB7
#define PORTSPISCK      PORTB
#define DDRSPISCK     DDRB
#define PINSPISCK     PINB

#define PSPISS        PB4
#define PORTSPISS     PORTB
```

```
#define DDRSPISS      DDRB
#define PINSPISS      PINB

#define PSPICSADC     PC7
#define PORTSPICSADC    PORTC
#define DDRSPICSADC     DDRC
#define PINSPICSADC     PINC

#define PCurrentNegative  PA0
#define PORTCurrentNegative PORTA
#define DDRCurrentNegative  DDRA
#define PINCurrentNegative  PINA

#define PCurrentPositive  PA1
#define PORTCurrentPositive PORTA
#define DDRCurrentPositive  DDRA
#define PINCurrentPositive  PINA

#define PMUXA0        PA4
#define PORTMUXA0     PORTA
#define DDRMUXA0      DDRA
#define PINMUXA0      PINA

#define PMUXA1        PA5
#define PORTMUXA1     PORTA
#define DDRMUXA1      DDRA
#define PINMUXA1      PINA

#define PMUXA2        PA6
#define PORTMUXA2     PORTA
#define DDRMUXA2      DDRA
#define PINMUXA2      PINA

#define PMUXEN        PA7
#define PORTMUXEN     PORTA
#define DDRMUXEN      DDRA
#define PINMUXEN      PINA


#define mAperV  10

#endif /* PINOUT_H_ */
```

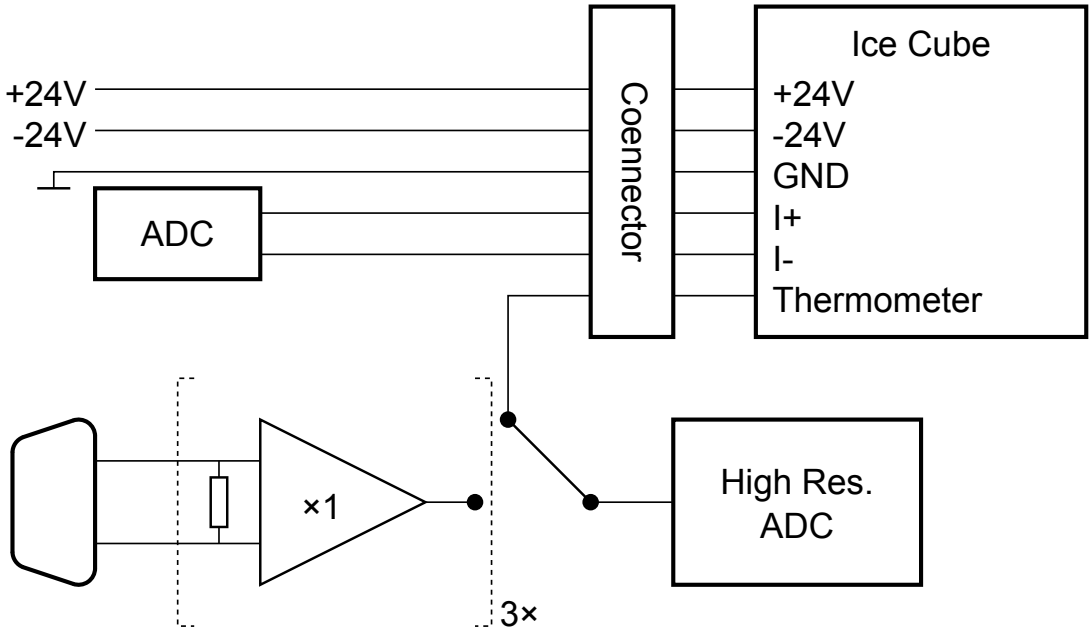



**Figure D.27.:** Simplified block diagram of the IceCube supply NIM insert

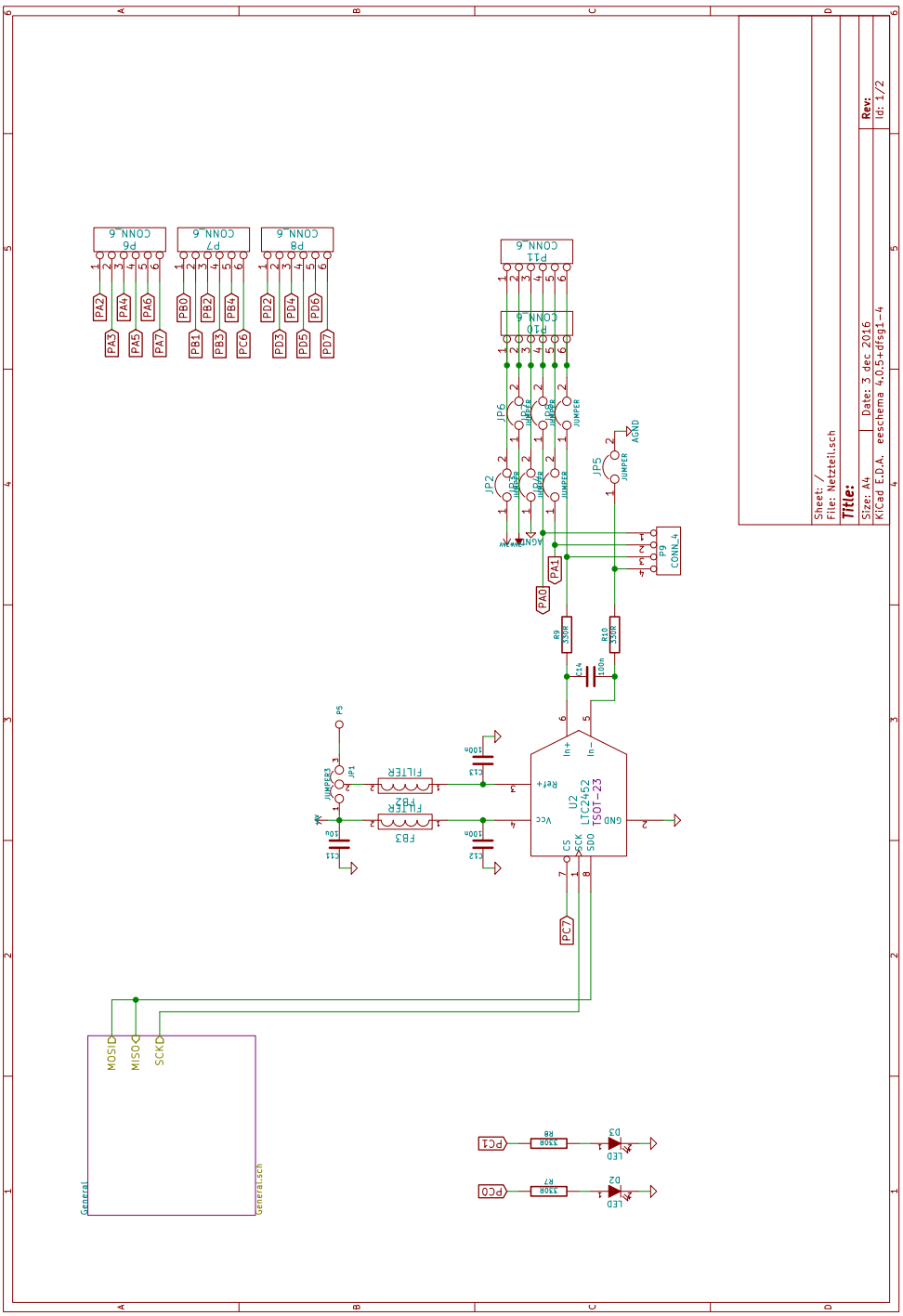



**Figure D.28.:** Schematic of the IceCube supply NIM insert

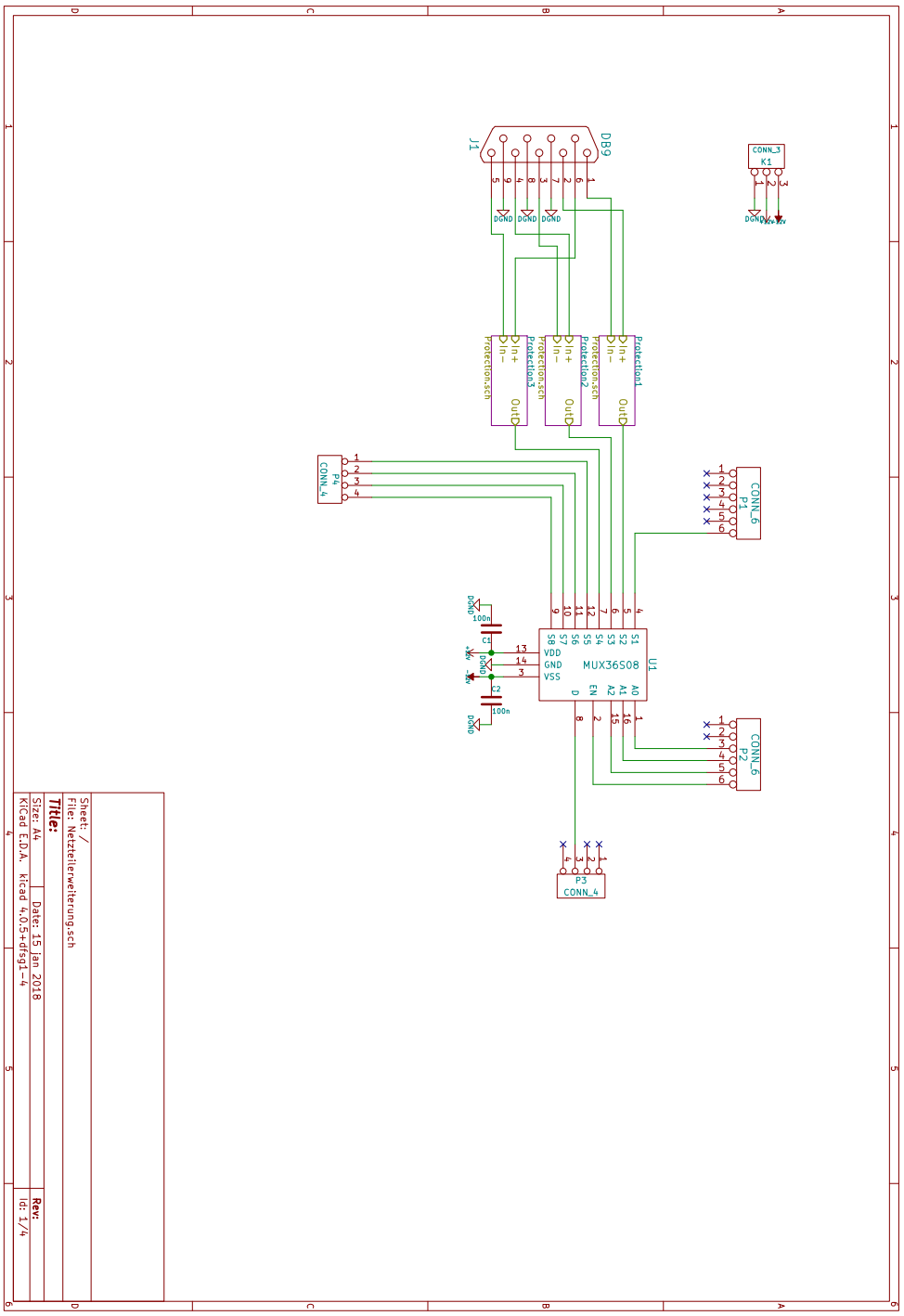



**Figure D.29.:** Schematic of the extension board for the IceCube supply NIM insert

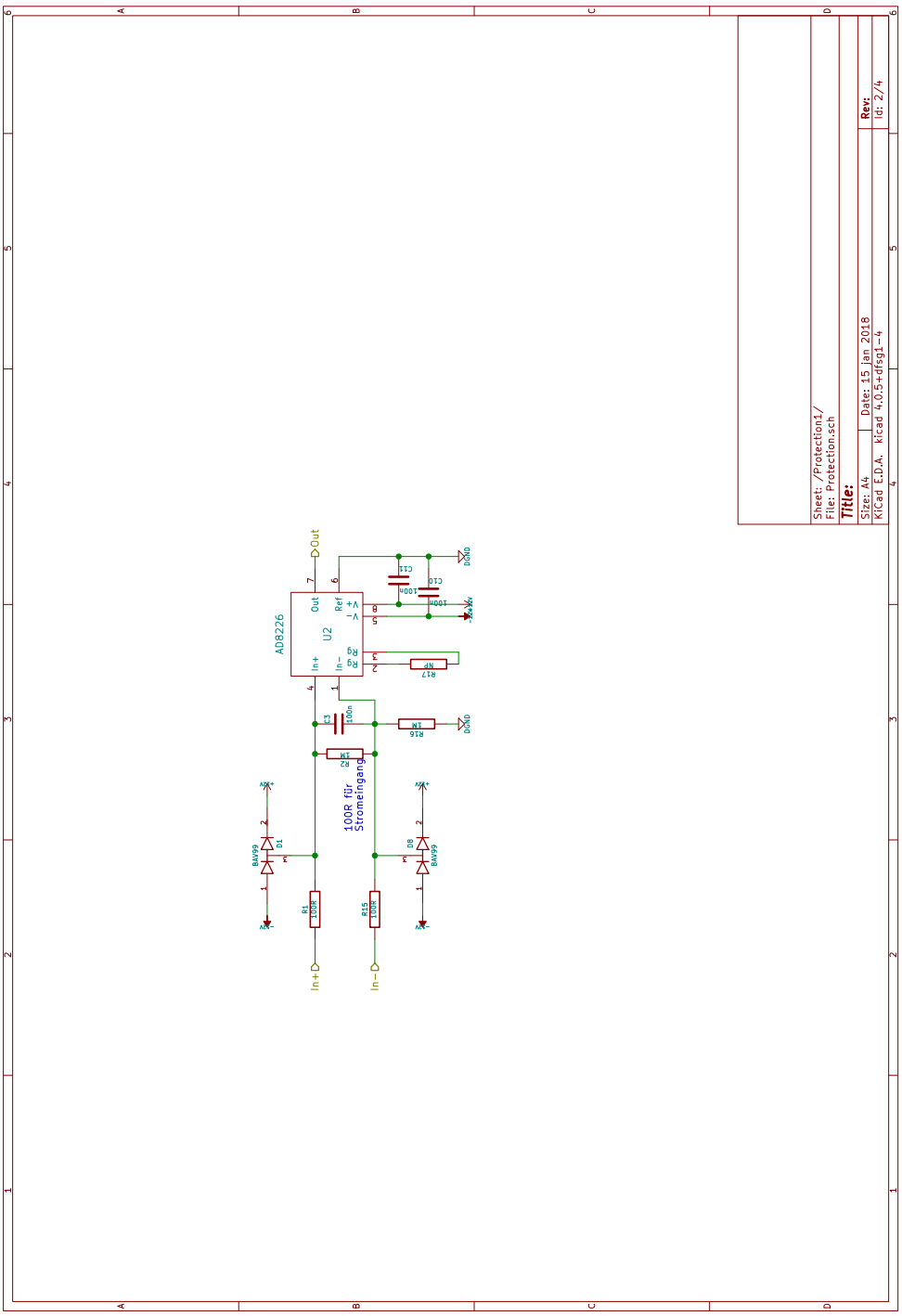
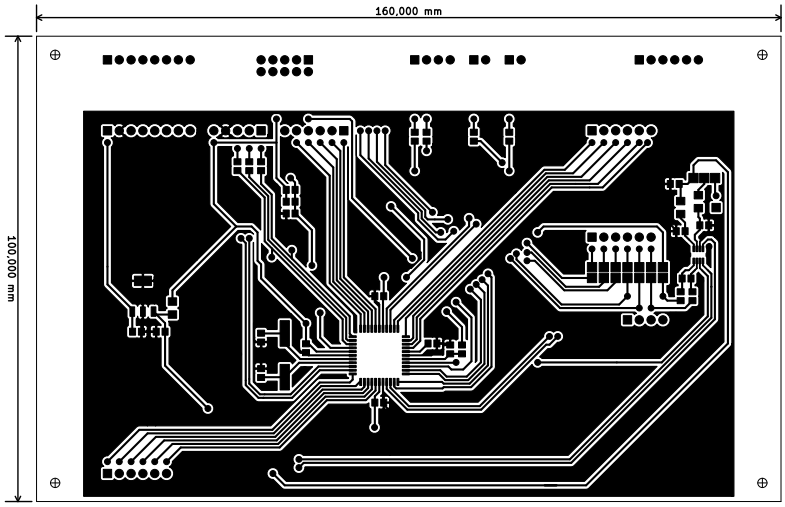
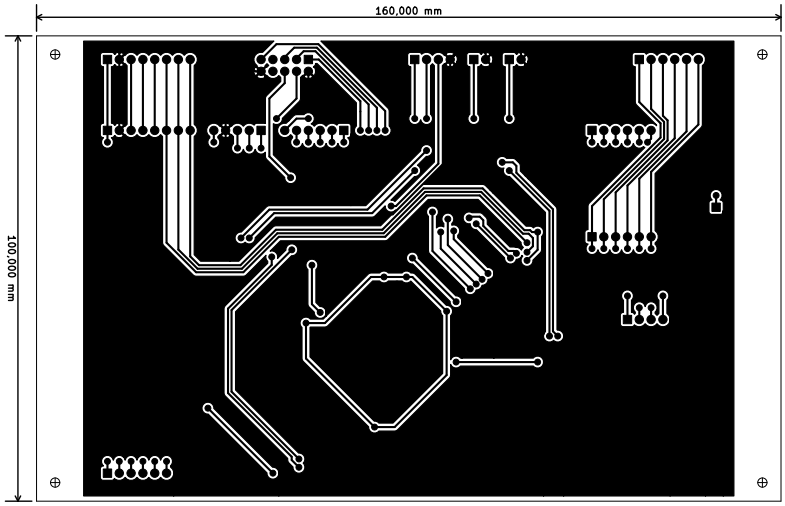


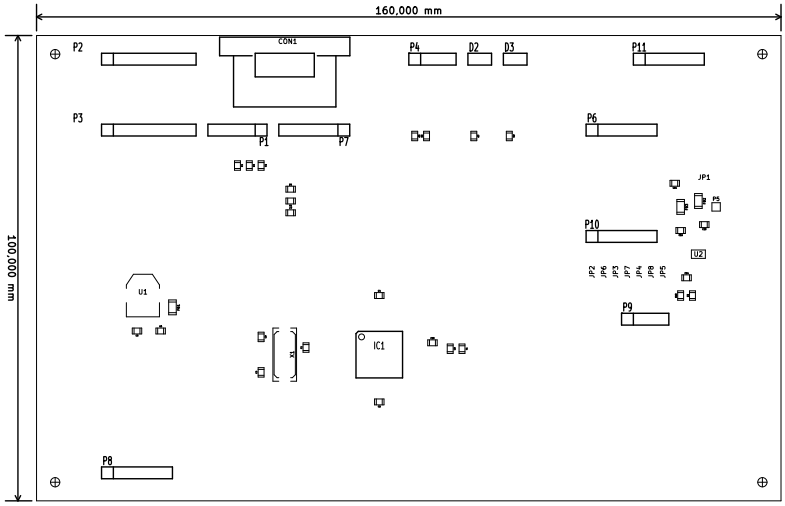

**Figure D.30.:** Schematic of the extension board for the IceCube supply NIM insert




**(a)** Top side copper structure

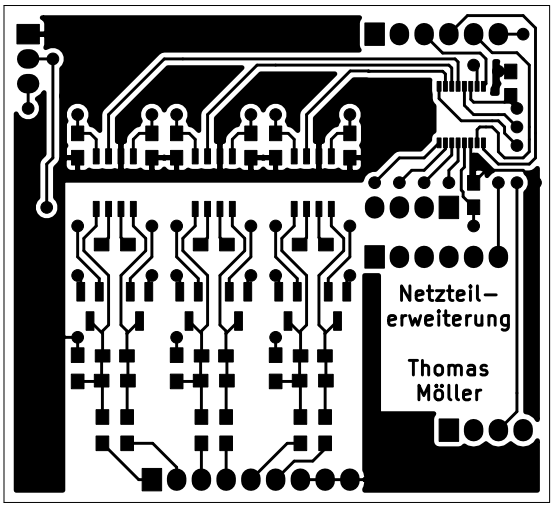



**(b)** Bottom side copper structure

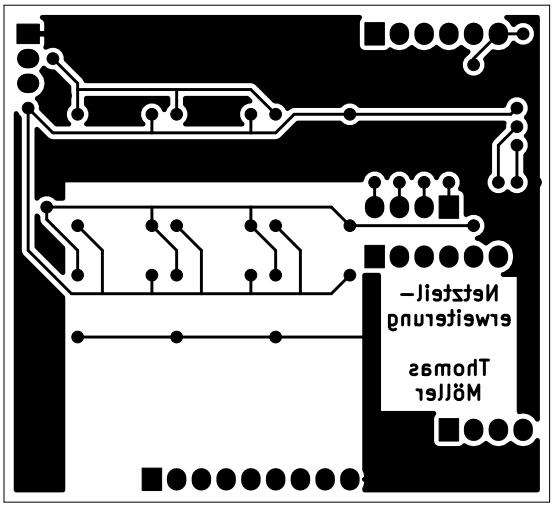



**(c)** Component references

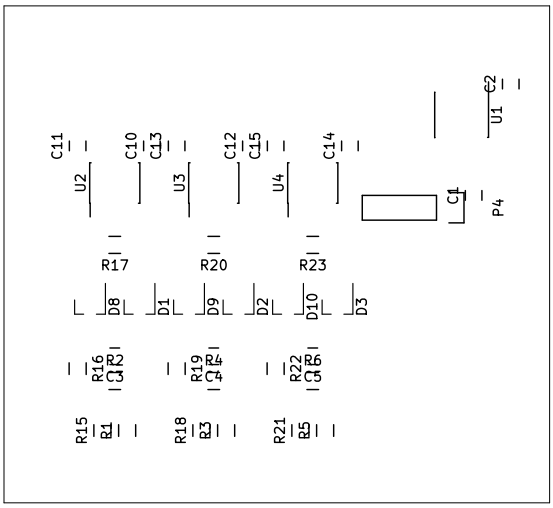
**Figure D.31.:** Circuit board layout of the IceCube supply NIM insert




**(a)** Top side copper structure

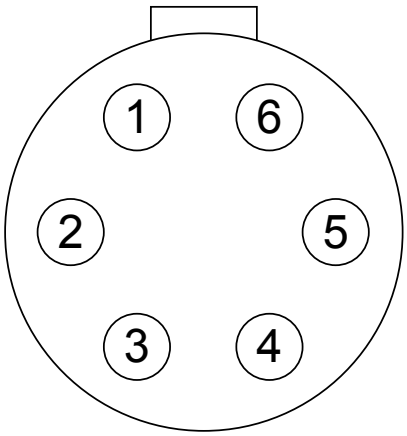



**(b)** Bottom side copper structure



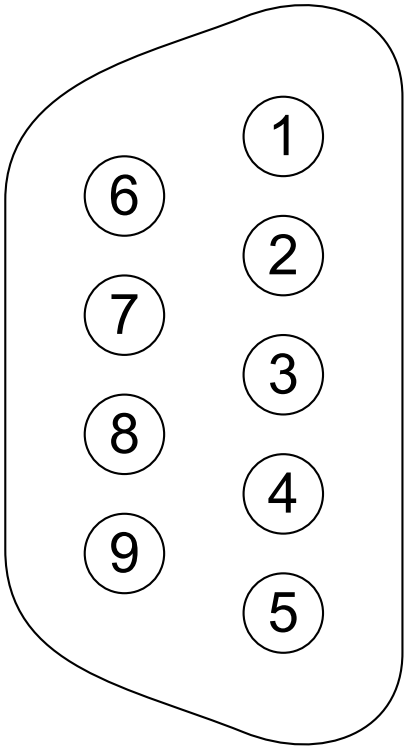

**(c)** Component references

**Figure D.32.:** Circuit board layout of the extension board for the IceCube supply NIM insert




| | | |
|---|---|---|
| 1 | I- | output |
| 2 | I+ | output |
| 3 | V thermometer | output |
| 4 | 24 V | input |
| 5 | GND | input |
| 6 | −24 V | input |

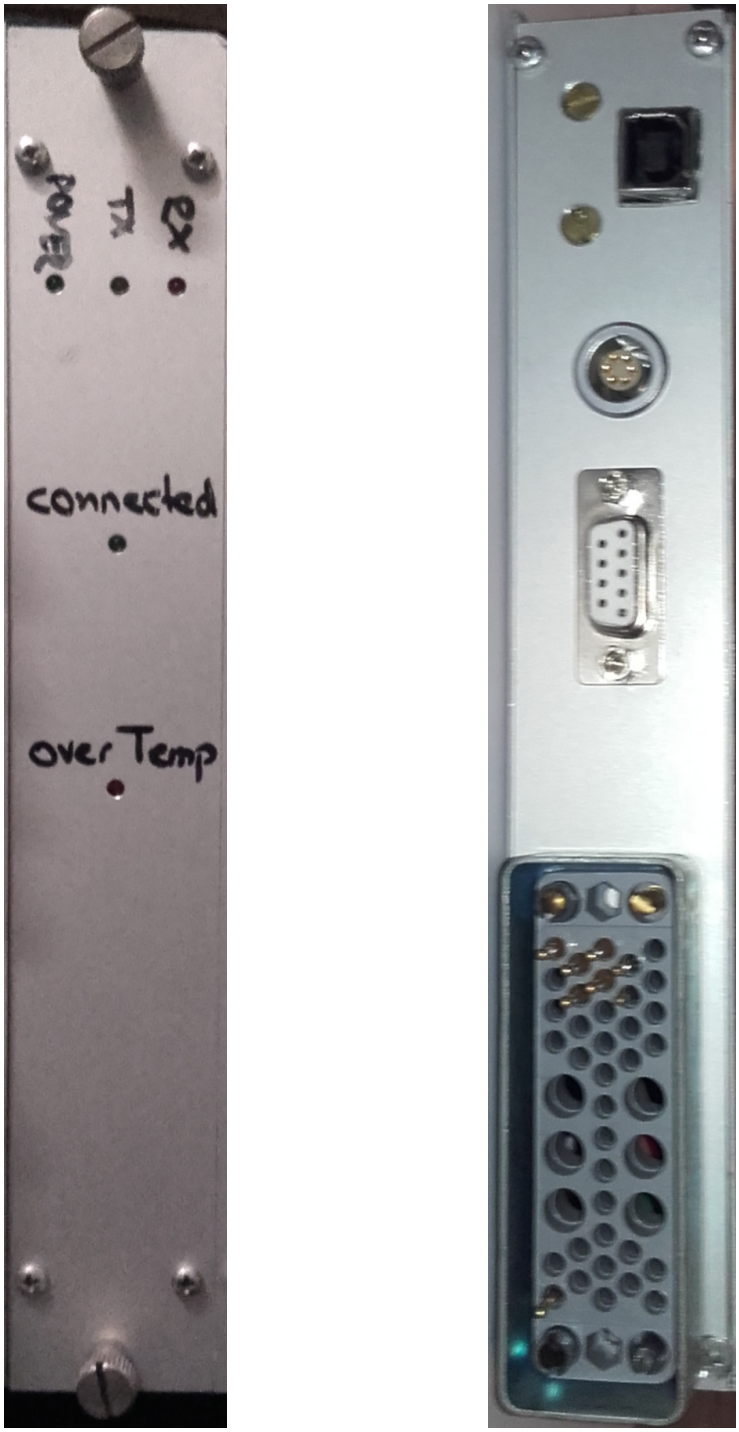
**(a)** IceCube connection




| | | |
|---|---|---|
| 1 | Input 1− | input |
| 2 | Input 1+ | input |
| 3 | Input 2− | input |
| 4 | Input 2− | input |
| 5 | Input 3− | input |
| 6 | Input 3+ | input |
| 7 | Ground | |
| 8 | Ground | |
| 9 | Ground | |

**(b)** Auxiliary inputs

**Figure D.33.:** Pin configuration of the IceCube supply NIM insert




**(a)** front **(b)** back

**Figure D.34.:** Photos of the outside of the IceCube supply NIM insert

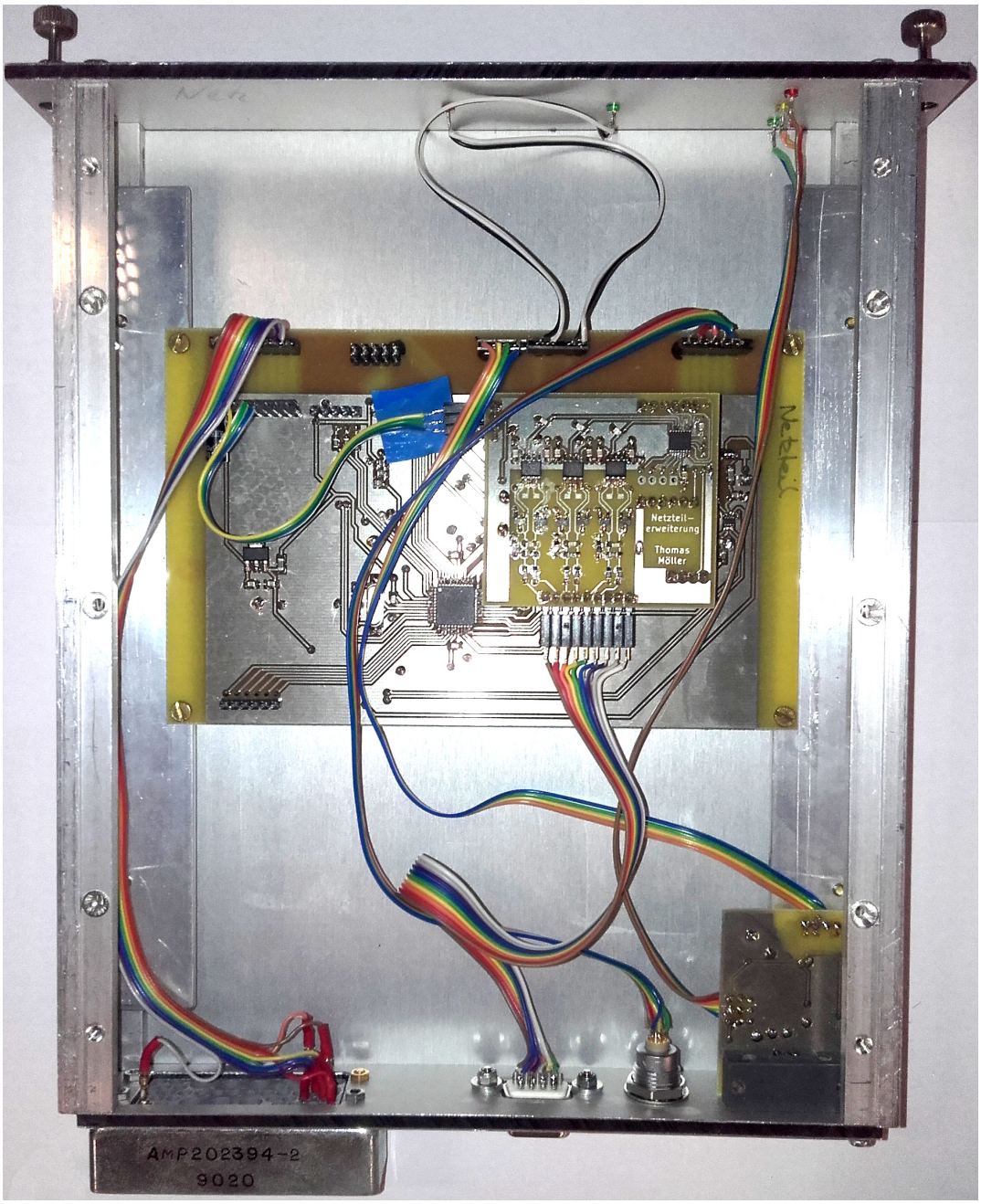



**Figure D.35.:** Photo of the inside of the IceCube supply NIM insert

## D.5. Motor

The purpose of the IceCube motor insert is to provide power and data to the Faulhaber motor controller. Additionally it also checks the limit switches and the fault output of the controller and makes their state available to the software.
The motor is powered from the symmetric 12 V rails, therefore all motor signal levels are shifted. This creates no problem since the communication is performed through two separate USB connections: one for the controller itself and one for the surveillance logic. These USB connectors are galvanically isolated from the motor ground and the NIM crate ground. All signals from the controller to the surveillance logic go through optical isolation.
The USB port that connects to the Faulhaber motor controller is recognized as a serial port, hence all the normal commands to the controller and even the software available from the manufacturer can be used. The surveillance logic supports the following commands.

- `*IDN?`
  This command returns the name of the insert, the author, the version, and date of the firmware. The output is three lines long.

- `:UPPer?`
  This command returns if the upper limit switch is currently active

- `:UPPer:TRIpped?`
  This command returns if the upper limit switch was active since the last reset.

- `:UPPer:RESet`
  This command resets the upper limit tripped state. It has no effect if the switch is currently active

- `:LOWer?`
  This command returns if the lower limit switch is currently active

- `:LOWer:TRIpped?`
  This command returns if the lower limit switch was active since the last reset.

- `:LOWer:RESet`
  This command resets the lower limit tripped state. It has no effect if the switch is currently active

- `:ERRor?`
  This command returns the state of the fault output of the controller.

The surveillance logic has a green LED in the front that indicates that there is no fault indicated by the controller. For the upper and lower limit switches, there are two LEDs and one switch each. The red LED indicates that the limit switch is currently active and the yellow LED is turned on if the limit switch is active and stays on to indicate that this limit was reached. It can be reset by the switch on the front. All manual controls and outputs can also be obtained with the aforementioned commands.
A block diagram of the module is shown in figure D.36. The full schematic of the circuit

board is shown in figure D.37 and the corresponding board layout in figure D.38. Photos of the front, back, and inside are shown in figures D.39 and D.40. The firmware for the motor insert consists of the following files.

**Source-Code D.29:** hardware.h

```
#ifndef HARDWARE_H_
#define HARDWARE_H_

#include "types.h"

extern volatile bool upperLimit, lowerLimit;        //True if upper/lower limit switch is
     currently pressed
extern volatile bool upperLimitTripped, lowerLimitTripped;  //True if upper/lower limit
    switch was pressed
extern volatile bool noError;               //True if motor does not signal an error

void hardwareInitialize();
void resetUpperLimit();
void resetLowerLimit();

void updateLEDs();
void readErrors();
void readUserSwitches();

#endif /* HARDWARE_H_ */
```

**Source-Code D.30:** hardware.c

```
#include "hardware.h"
#include "pinout.h"
#include "switch.h"

volatile bool upperLimit, lowerLimit;
volatile bool upperLimitTripped, lowerLimitTripped;
volatile bool noError;

void hardwareInitialize(){
  DDRLEDNoError |= (1<<PLEDNoError);
  DDRLEDNoErrorRemote |= (1<<PLEDNoErrorRemote);
  DDRLEDUpperLimit |= (1<<PLEDUpperLimit);
  DDRLEDUpperLimitRemote |= (1<<PLEDUpperLimitRemote);
  DDRLEDUpperLimitTripped |= (1<<PLEDUpperLimitTripped);
  DDRLEDLowerLimit |= (1<<PLEDLowerLimit);
  DDRLEDLowerLimitRemote |= (1<<PLEDLowerLimitRemote);
  DDRLEDLowerLimitTripped |= (1<<PLEDLowerLimitTripped);
  PORTInputError |= (1<<PInputError);
  PORTInputLowerLimit |= (1<<PInputLowerLimit);
  PORTInputUpperLimit |= (1<<PInputUpperLimit);
  switchInitialize();
}
void resetUpperLimit(){
  upperLimitTripped = False;
  readErrors();
}
void resetLowerLimit(){
  lowerLimitTripped = False;
  readErrors();
}

void updateLEDs(){
  if(noError){
    PORTLEDNoError |= (1<<PLEDNoError);
    PORTLEDNoErrorRemote |= (1<<PLEDNoErrorRemote);
  }else{
    PORTLEDNoError &= ~(1<<PLEDNoError);
    PORTLEDNoErrorRemote &= ~(1<<PLEDNoErrorRemote);
  }
```

```
  if(upperLimit){
    PORTLEDUpperLimit |= (1<<PLEDUpperLimit);
    PORTLEDUpperLimitRemote |= (1<<PLEDUpperLimitRemote);
  }else{
    PORTLEDUpperLimit &= ~(1<<PLEDUpperLimit);
    PORTLEDUpperLimitRemote &= ~(1<<PLEDUpperLimitRemote);
  }
  if(upperLimitTripped){
    PORTLEDUpperLimitTripped |= (1<<PLEDUpperLimitTripped);
  }else{
    PORTLEDUpperLimitTripped &= ~(1<<PLEDUpperLimitTripped);
  }
  if(lowerLimit){
    PORTLEDLowerLimit |= (1<<PLEDLowerLimit);
    PORTLEDLowerLimitRemote |= (1<<PLEDLowerLimitRemote);
  }else{
    PORTLEDLowerLimit &= ~(1<<PLEDLowerLimit);
    PORTLEDLowerLimitRemote &= ~(1<<PLEDLowerLimitRemote);
  }
  if(lowerLimitTripped){
    PORTLEDLowerLimitTripped |= (1<<PLEDLowerLimitTripped);
  }else{
    PORTLEDLowerLimitTripped &= ~(1<<PLEDLowerLimitTripped);
  }
}
void readErrors(){
  if(PINInputUpperLimit & (1<<PInputUpperLimit)){
    upperLimit = True;
    upperLimitTripped = True;
  }else{
    upperLimit = False;
  }
  if(PINInputLowerLimit & (1<<PInputLowerLimit)){
    lowerLimit = True;
    lowerLimitTripped = True;
  }else{
    lowerLimit = False;
  }
  if(PINInputError & (1<<PInputError)){
    noError = False;
  }else{
    noError = True;
  }
}
void readUserSwitches(){
  switchUpdate();
}
```

**Source-Code D.31:** main.c

```
#include "serial.h"
#include <avr/interrupt.h>
#include <util/delay.h>

#include "hardware.h"

int main(){
  hardwareInitialize();
  initializeSerialPort();
  sei();
  while(True){
    readUserSwitches();
    readErrors();
    updateLEDs();
  }
}
```

**Source-Code D.32:** parser.h

```
#ifndef PARSER_H_
#define PARSER_H_

#include "types.h"

extern volatile char command[100];

void parseCommand();

#endif /* PARSER_H_ */
```

**Source-Code D.33:** parser.c

```
#include "parser.h"
#include "types.h"
#include "serial.h"
#include "hardware.h"
#include <stdio.h>

volatile char command[100];

bool compareCommand(char reference[]){
  uint8_t i = 0;
  while(reference[i]){
    if(command[i] != reference[i]) return false;
    i++;
  }
  return true;
}

void respond(bool result){
  if(result){
    sendString("done");
  }else{
    sendString("failed");
  }
}

void respondYesNo(bool result){
  if(result){
    sendString("YES");
  }else{
    sendString("NO");
  }
}

void parseCommand(){
  if(compareCommand("*IDN?")){
    sendString("Motor␣surveillance␣unit\r\nby␣Thomas.Moeller@uni.kn\r\nFirmware␣2␣
        (12.06.2017)");
    return;
  }
  if(compareCommand(":UPP?")){
    respondYesNo(upperLimit);
    return;
  }
  if(compareCommand(":UPP:TRI?")){
    respondYesNo(upperLimitTripped);
    return;
  }
  if(compareCommand(":UPP:RES")){
    resetUpperLimit();
    respond(!upperLimitTripped);
    return;
  }
  if(compareCommand(":LOW?")){
    respondYesNo(lowerLimit);
    return;
  }
```

```
  if(compareCommand(":LOW:TRI?")){
    respondYesNo(lowerLimitTripped);
    return;
  }
  if(compareCommand(":LOW:RES")){
    resetLowerLimit();
    respond(!lowerLimitTripped);
    return;
  }
  if(compareCommand(":ERR?")){
    respondYesNo(!noError);
    return;
  }
  sendString("command␣unknown");
}
```

**Source-Code D.34:** pinout.h

```
#ifndef PINOUT_H_
#define PINOUT_H_

#include <avr/io.h>

#define PORT_RX PORTD
#define DDR_RX  DDRD
#define PIN_RX  PIND
#define P_RX  PD0

#define PORT_TX PORTD
#define DDR_TX  DDRD
#define PIN_TX  PIND
#define P_TX  PD1

#define PSwitchUpperLimit     PA1
#define PORTSwitchUpperLimit    PORTA
#define DDRSwitchUpperLimit     DDRA
#define PINSwitchUpperLimit     PINA

#define PSwitchLowerLimit     PA0
#define PORTSwitchLowerLimit    PORTA
#define DDRSwitchLowerLimit     DDRA
#define PINSwitchLowerLimit     PINA

#define PInputUpperLimit      PB0
#define PORTInputUpperLimit     PORTB
#define DDRInputUpperLimit      DDRB
#define PINInputUpperLimit      PINB

#define PInputLowerLimit      PB1
#define PORTInputLowerLimit     PORTB
#define DDRInputLowerLimit      DDRB
#define PINInputLowerLimit      PINB

#define PInputError         PD2
#define PORTInputError        PORTD
#define DDRInputError       DDRD
#define PINInputError       PIND

#define PLEDUpperLimit        PD7
#define PORTLEDUpperLimit     PORTD
#define DDRLEDUpperLimit      DDRD
#define PINLEDUpperLimit      PIND

#define PLEDUpperLimitRemote    PD6
#define PORTLEDUpperLimitRemote   PORTD
#define DDRLEDUpperLimitRemote    DDRD
#define PINLEDUpperLimitRemote    PIND

#define PLEDLowerLimit        PC7
```

```
#define PORTLEDLowerLimit     PORTC
#define DDRLEDLowerLimit      DDRC
#define PINLEDLowerLimit      PINC

#define PLEDLowerLimitRemote    PD4
#define PORTLEDLowerLimitRemote   PORTD
#define DDRLEDLowerLimitRemote    DDRD
#define PINLEDLowerLimitRemote    PIND

#define PLEDNoError         PC1
#define PORTLEDNoError        PORTC
#define DDRLEDNoError       DDRC
#define PINLEDNoError       PINC

#define PLEDNoErrorRemote     PD5
#define PORTLEDNoErrorRemote    PORTD
#define DDRLEDNoErrorRemote     DDRD
#define PINLEDNoErrorRemote     PIND

#define PLEDUpperLimitTripped   PC0
#define PORTLEDUpperLimitTripped  PORTC
#define DDRLEDUpperLimitTripped   DDRC
#define PINLEDUpperLimitTripped   PINC

#define PLEDLowerLimitTripped   PC6
#define PORTLEDLowerLimitTripped  PORTC
#define DDRLEDLowerLimitTripped   DDRC
#define PINLEDLowerLimitTripped   PINC

#endif /* PINOUT_H_ */
```

**Source-Code D.35:** switch.h

```
#ifndef SWITCH_H_
#define SWITCH_H_

void switchInitialize();
void switchUpdate();

#endif /* SWITCH_H_ */
```

**Source-Code D.36:** switch.c

```
#include "switch.h"
#include "hardware.h"
#include "pinout.h"

void switchInitialize(){
  PORTSwitchUpperLimit |= (1<<PSwitchUpperLimit);
  PORTSwitchLowerLimit |= (1<<PSwitchLowerLimit);
}

void switchUpdate(){
  if((~PINSwitchUpperLimit) & (1<<PSwitchUpperLimit)){
    resetUpperLimit();
  }
  if((~PINSwitchLowerLimit) & (1<<PSwitchLowerLimit)){
    resetLowerLimit();
  }
}
```

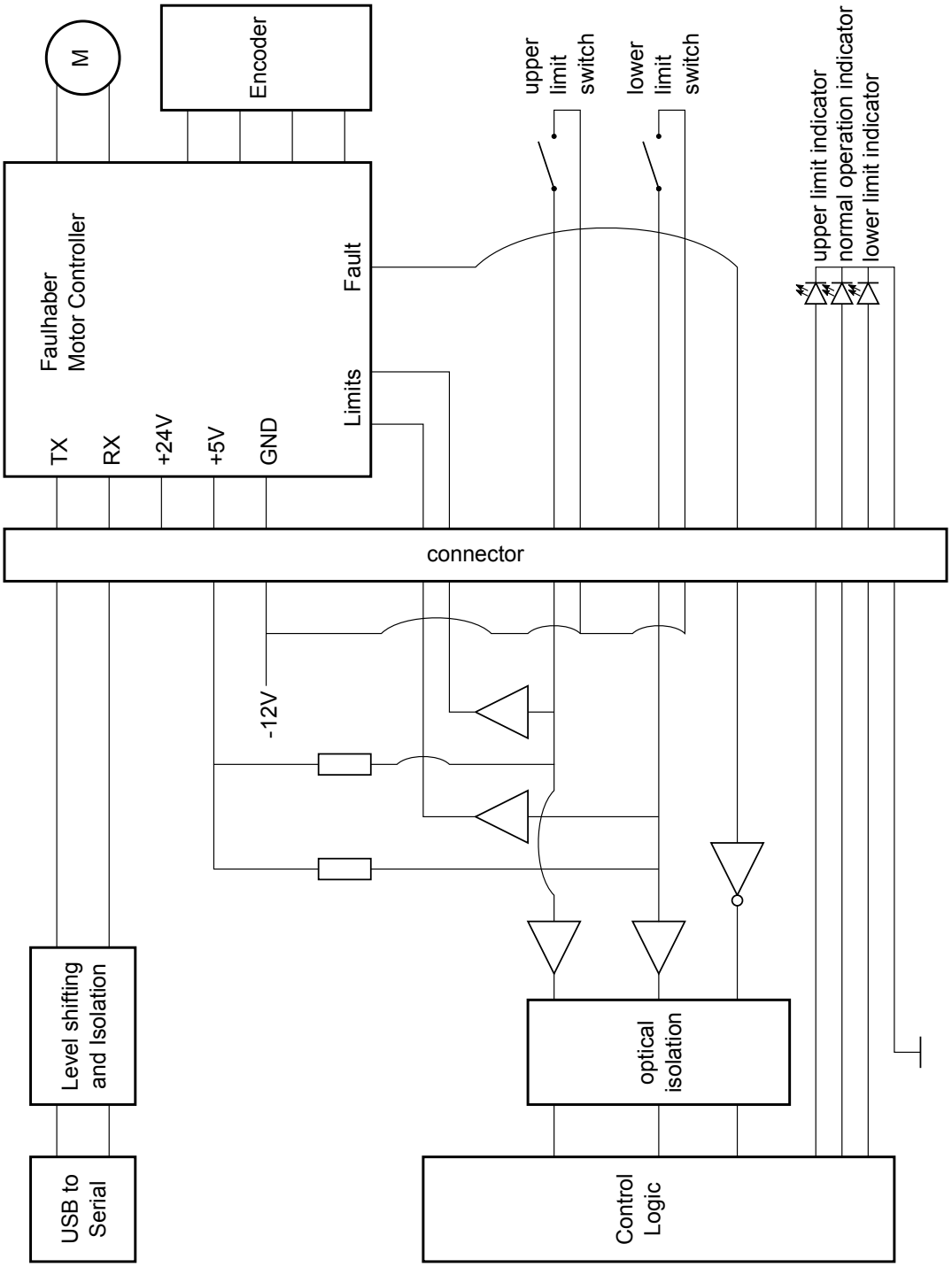



**Figure D.36.:** Simplified block diagram of the IceCube motor NIM insert

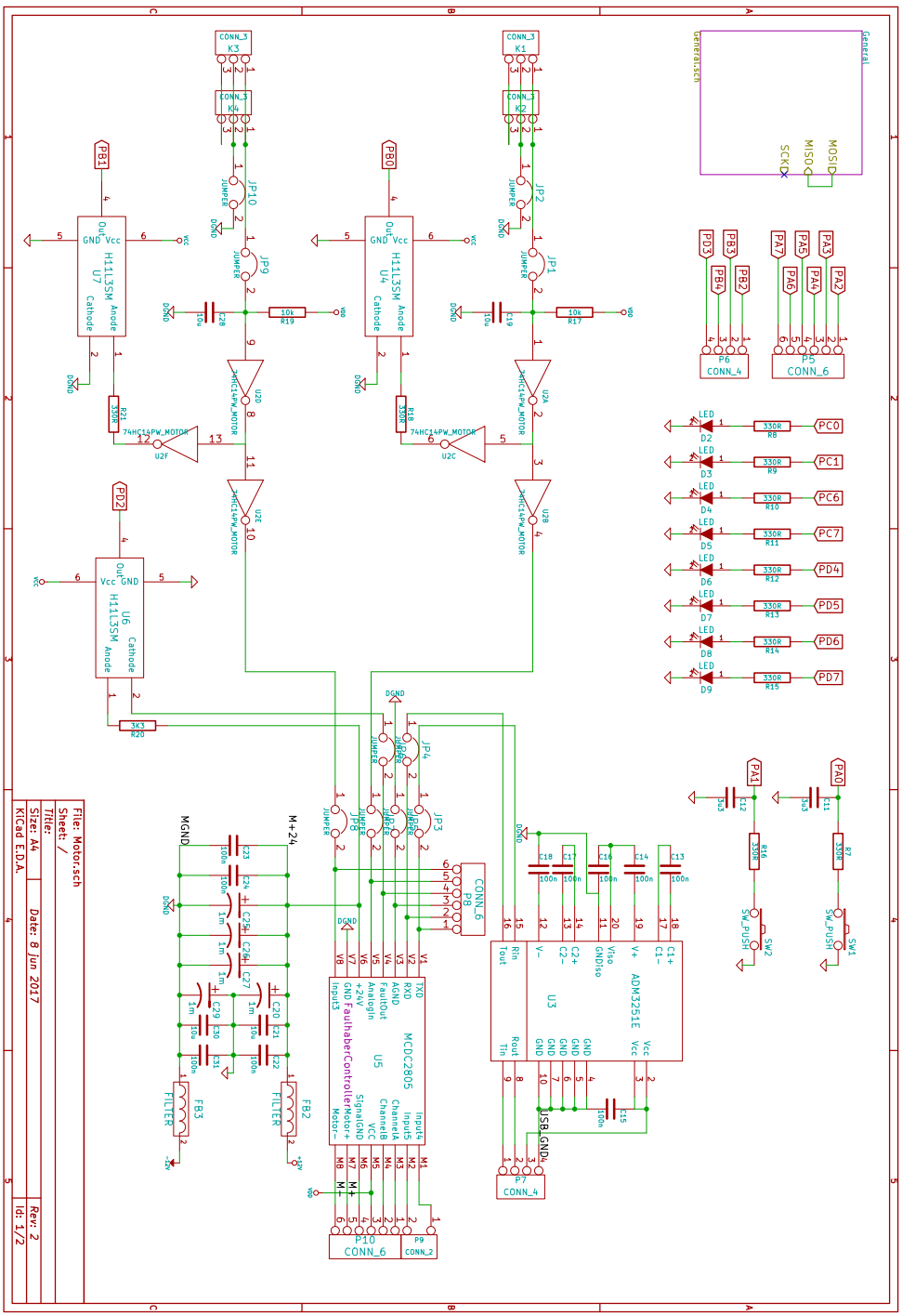
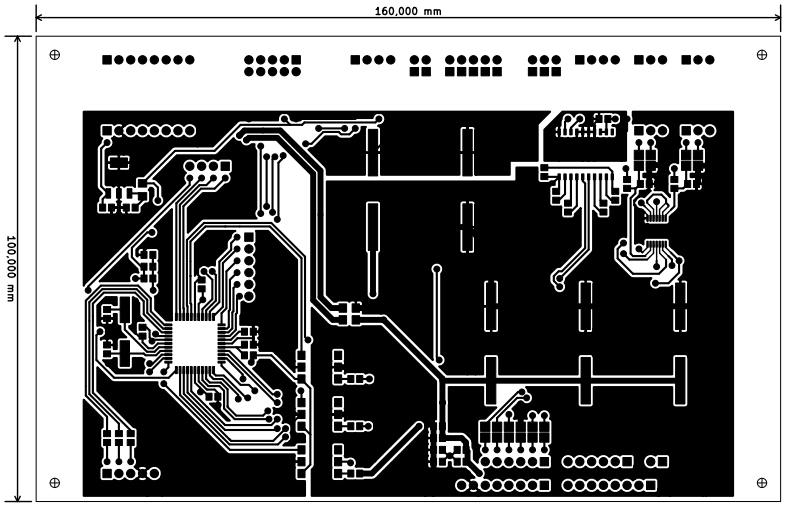
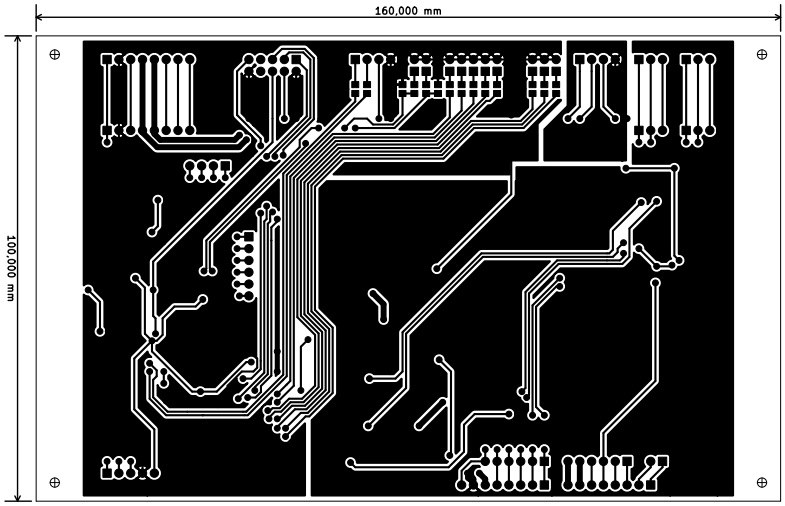


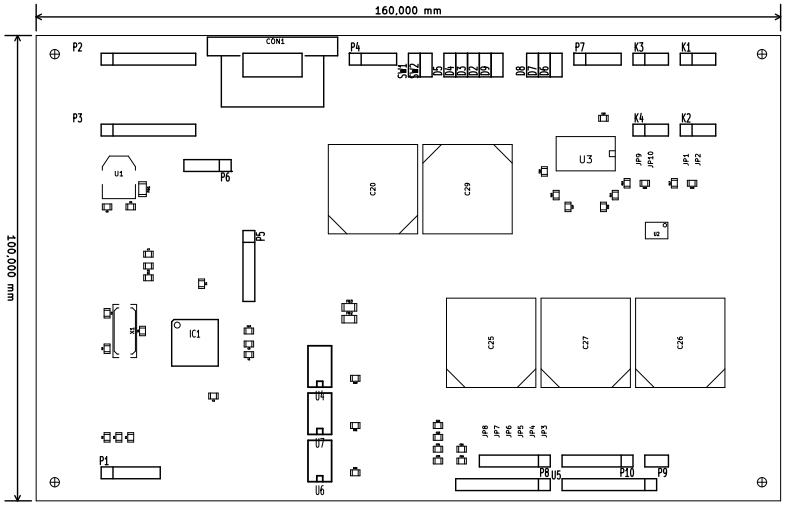

**Figure D.37.:** Schematic of the IceCube motor NIM insert




**(a)** Top side copper structure




**(b)** Bottom side copper structure




**(c)** Component references

**Figure D.38.:** Circuit board layout of the IceCube motor NIM insert


**(a)** front **(b)** back

**Figure D.39.:** Photos of the outside of the IceCube motor NIM insert

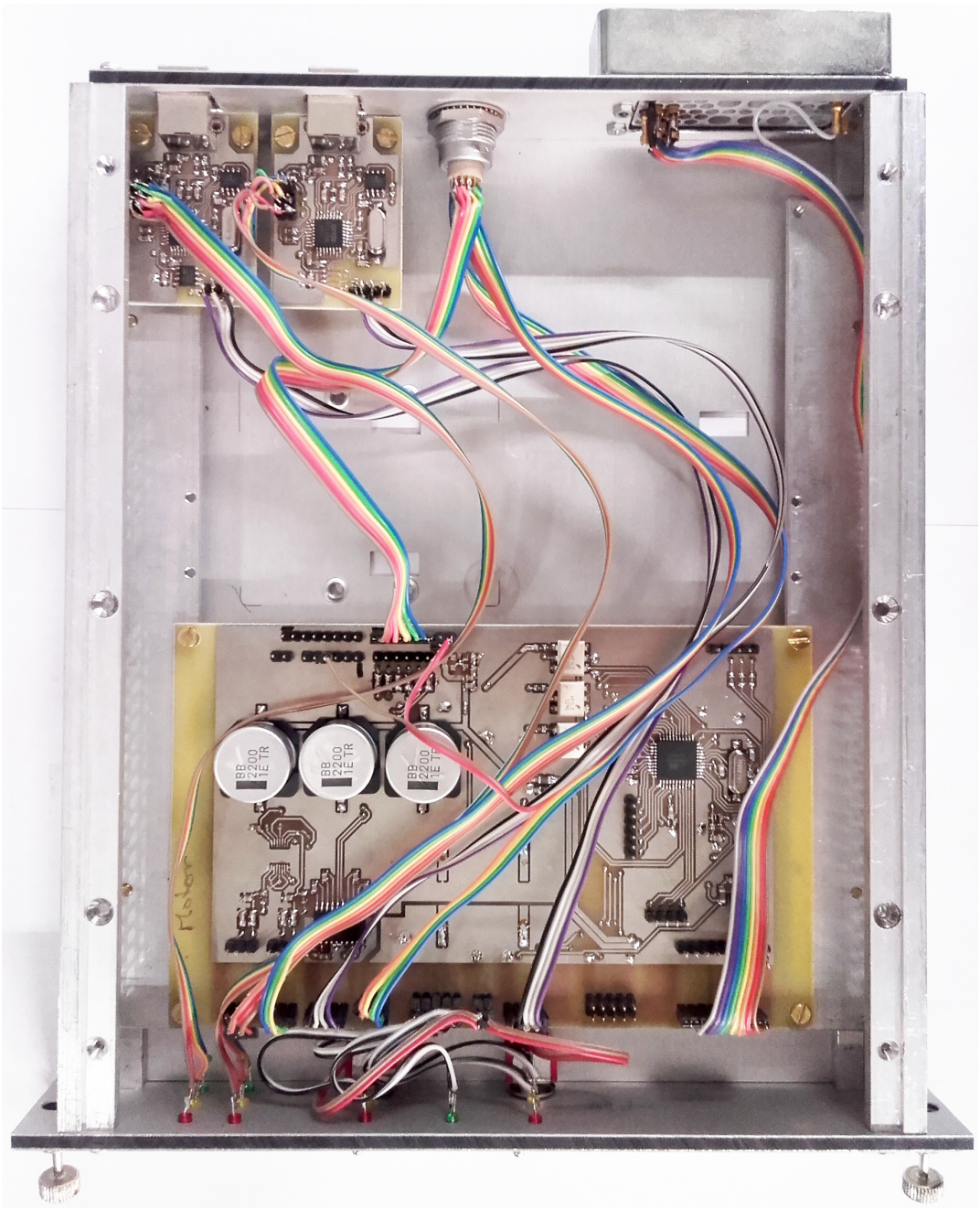



**Figure D.40.:** Photo of the inside of the IceCube motor NIM insert

## D.6. Auxiliary wiring

The purpose of the auxiliary wiring insert is to serve as an adapter between the LakeShore temperature controller and the heater and thermometer inside of the sample chamber. Additionally it includes the power supply for the two white LEDs mounted inside the chamber.
The LEDs are current driven and this current can be set in steps with the switch on the front. An additional green LED indicates if the LEDs inside the chamber are turned on. A USB port allows to connect to the microcontroller which drives the LEDs. It supports the following commands.

- `*IDN?`
  This command returns the name of the insert, the author, the version, and date of the firmware. The output is three lines long.
- `:LED OFF`
  This command turns the LEDs off.
- `:LED?`
  This command returns the brightness level of the LEDs.
- `:LED␣<Number>`
  This command sets the brightness level.
- `:BASevoltage`
  This command returns the voltage at the base of the bipolar junction transistor which drives the LEDs in the format `%06.4fV` in V.
- `:EMIttervoltage`
  This command returns the voltage at the emitter of the bipolar junction transistor which drives the LEDs in the format `%06.4fV` in V.
- `:CURrent?`
  This command returns the current that is used to drive the LEDs. The response has the format `%06.3fmA` in mA.

A block diagram of the module is shown in figure D.41. The full schematic of the circuit board is shown in figure D.42 and the corresponding board layout in figure D.43. The pinout of the connectors is shown in figure D.44 and photos of the front, back, and inside are shown in figures D.45 and D.46. The firmware consists of the following files.

**Source-Code D.37:** adc.h

```
#ifndef ADC_H_
#define ADC_H_

#include "types.h"

extern volatile uint16_t baseReading;   //right aligned => 1024 is 5V
extern volatile uint16_t emitterReading;  //right aligned => 1024 is 5V

void adcInitialize();

#endif /* ADC_H_ */
```

**Source-Code D.38:** adc.c

```
#include "adc.h"
#include "pinout.h"
#include <avr/interrupt.h>

volatile uint16_t baseReading;    //right aligned => 1024 is 5V
volatile uint16_t emitterReading; //right aligned => 1024 is 5V

void adcInitialize(){
  ADMUX |= (1<<REFS0); //5V Referece
  ADMUX &= ~((1<<MUX4) | (1<<MUX3) | (1<<MUX2) | (1<<MUX1) | (1<<MUX0));
  ADMUX |= (PBaseSense<<MUX0); //set input to BaseSense
  ADCSRA |= (1<<ADPS2) | (1<<ADPS0); //ADC Freqency = 3686400Hz / 32 = 115200Hz
  ADCSRA |= (1<<ADEN); //ADC enable
  ADCSRA |= (1<<ADIE); //ADC Interrupt enable
  ADCSRA |= (1<<ADSC); //ADC start
}

ISR(ADC_vect){
  if((ADMUX & ((1<<MUX4) | (1<<MUX3) | (1<<MUX2) | (1<<MUX1) | (1<<MUX0))) == (PBaseSense
      <<MUX0)){
    baseReading = ADC;
    ADMUX &= ~((1<<MUX4) | (1<<MUX3) | (1<<MUX2) | (1<<MUX1) | (1<<MUX0));
    ADMUX |= (PEmitterSense<<MUX0); //Set input to EmitterSense
  }else{
    emitterReading = ADC;
    ADMUX &= ~((1<<MUX4) | (1<<MUX3) | (1<<MUX2) | (1<<MUX1) | (1<<MUX0));
    ADMUX |= (PBaseSense<<MUX0); //Set input to BaseSense
  }
  ADCSRA |= (1<<ADSC); //start ADC
}
```

**Source-Code D.39:** brightness.h

```
#ifndef BRIGHTNESS_H_
#define BRIGHTNESS_H_

#define BRIGHTNESSMAXIMUM 12

#include "types.h"

extern volatile uint8_t brightness;
extern uint32_t resistances[];

void brightnessInitialize();
void brightnessBrighter();
void brightnessDimmer();
void brightnessSet(uint8_t new);
void brightnessOff();

#endif /* BRIGHTNESS_H_ */
```

**Source-Code D.40:** brightness.c

```
#include "brightness.h"
#include "pinout.h"
#include "types.h"

volatile uint8_t brightness = 0;

uint32_t resistances[]={
    1000001,
    470001,
    270001,
    120001,
    68001,
    33001,
    15001,
```

```
    8201,
    3901,
    2201,
    1001,
    471
};

void resistancesOff(){ //deactivate all pull-down resistances
  DDRBrightness01 &= ~(1<<PBrightness01);
  DDRBrightness02 &= ~(1<<PBrightness02);
  DDRBrightness03 &= ~(1<<PBrightness03);
  DDRBrightness04 &= ~(1<<PBrightness04);
  DDRBrightness05 &= ~(1<<PBrightness05);
  DDRBrightness06 &= ~(1<<PBrightness06);
  DDRBrightness07 &= ~(1<<PBrightness07);
  DDRBrightness08 &= ~(1<<PBrightness08);
  DDRBrightness09 &= ~(1<<PBrightness09);
  DDRBrightness10 &= ~(1<<PBrightness10);
  DDRBrightness11 &= ~(1<<PBrightness11);
  DDRBrightness12 &= ~(1<<PBrightness12);
}

void LEDoff(){ //For turn-off
  DDRBase &= ~(1<<PBase); //hold the trimmed base-voltage up, but only by the internal
      PullUp,
  DDRBaseSense |= (1<<PBaseSense); //until it is activly forced to GND.
  PORTEmitterSense |= (1<<PEmitterSense); //The emitter is pulled up with the internal
      pull-up against the pull-down resistances
  resistancesOff(); //until they are turned off.
  PORTStatusLED &= ~(1<<PStatusLED);
}

void LEDon(){
  resistancesOff();

  switch (brightness) {
  case 1:
    DDRBrightness01 |= (1<<PBrightness01);
    break;
  case 2:
    DDRBrightness02 |= (1<<PBrightness02);
    break;
  case 3:
    DDRBrightness03 |= (1<<PBrightness03);
    break;
  case 4:
    DDRBrightness04 |= (1<<PBrightness04);
    break;
  case 5:
    DDRBrightness05 |= (1<<PBrightness05);
    break;
  case 6:
    DDRBrightness06 |= (1<<PBrightness06);
    break;
  case 7:
    DDRBrightness07 |= (1<<PBrightness07);
    break;
  case 8:
    DDRBrightness08 |= (1<<PBrightness08);
    break;
  case 9:
    DDRBrightness09 |= (1<<PBrightness09);
    break;
  case 10:
    DDRBrightness10 |= (1<<PBrightness10);
    break;
  case 11:
    DDRBrightness11 |= (1<<PBrightness11);
    break;
  case 12:
```

```
    DDRBrightness12 |= (1<<PBrightness12);
    break;
  }

  PORTEmitterSense &= ~(1<<PEmitterSense); //Reverse the procedure of the turn-off
  DDRBaseSense &= ~(1<<PBaseSense);
  DDRBase |= (1<<PBase);
  PORTStatusLED |= (1<<PStatusLED);
}

void brightnessInitialize(){
  DDRStatusLED |= (1<<PStatusLED);
  LEDoff();
  PORTBase |= (1<<PBase);
}

void brightnessBrighter(){
  if (brightness<BRIGHTNESSMAXIMUM){
    brightnessSet(brightness+1);
  }
}

void brightnessDimmer(){
  if (brightness>0){
    brightnessSet(brightness-1);
  }
}

void brightnessSet(uint8_t new){
  if (new<=0){
    brightness = 0;
    LEDoff();
  }else{
    if(new>BRIGHTNESSMAXIMUM){
      brightness=BRIGHTNESSMAXIMUM;
    }else{
      brightness=new;
    }
    LEDon();
  }
}

void brightnessOff(){
  brightness = 0;
  LEDoff();
}
```

**Source-Code D.41:** main.c

```
#include "serial.h"
#include <avr/interrupt.h>
#include "parser.h"
#include "types.h"

#include "brightness.h"
#include "pinout.h"
#include "switch.h"
#include "adc.h"

int main(){
  switchInitialize();
  initializeSerialPort();
  brightnessInitialize();
  adcInitialize();
  sei();
  while(True){
    switchLoop(); //Check for switch changes
    if (switchUpFlag) {
      brightnessBrighter();
```

```
      switchUpFlag = False;
    }
    if (switchDownFlag) {
      brightnessDimmer();
      switchDownFlag = False;
    }
  }
}
```

**Source-Code D.42:** parser.h

```
#ifndef PARSER_H_
#define PARSER_H_

#include "types.h"

extern volatile char command[100];

void parseCommand();

#endif /* PARSER_H_ */
```

**Source-Code D.43:** parser.c

```
#include "parser.h"
#include "types.h"
#include "serial.h"
#include <stdio.h>
#include "adc.h"
#include "brightness.h"

volatile char command[100];

bool compareCommand(char reference[]){
  uint8_t i = 0;
  while(reference[i]){
    if(command[i] != reference[i]) return false;
    i++;
  }
  return true;
}

void respond(bool result){
  if(result){
    sendString("done");
  }else{
    sendString("failed");
  }
}

void parseCommand(){
  if(compareCommand("*IDN?")){
    sendString("LED␣supply␣unit\r\nby␣Thomas.Moeller@uni.kn\r\nFirmware␣1␣(20.05.2017)");
    return;
  }
  if(compareCommand(":LED␣OFF")){
    brightnessOff();
    respond(true);
    return;
  }
  if(compareCommand(":LED?")){
    char temp[4];
    sprintf(temp,"%02d",brightness);
    sendString(temp);
    return;
  }
  if(compareCommand(":LED␣")){
    uint8_t temp;
    int8_t ret = sscanf(&command[5],"%d",&temp);
```

```
    brightnessSet(temp);
    respond(ret>0);
    return;
  }
  if(compareCommand(":BAS?")){
    char temp[10];
    sprintf(temp,"%06.4fV",baseReading*5/1024.0);
    sendString(temp);
    return;
  }
  if(compareCommand(":EMI?")){
    char temp[10];
    sprintf(temp,"%06.4fV",emitterReading*5/1024.0);
    sendString(temp);
    return;
  }
  if(compareCommand(":CUR?")){
    if(brightness==0){
      sendString("00.000mA");
      return;
    }
    char temp[10];
    sprintf(temp,"%06.3fmA",emitterReading*5/1.024/resistances[brightness-1]);
    sendString(temp);
    return;
  }
  sendString("command␣unknown");
}
```

**Source-Code D.44:** pinout.h

```
#ifndef PINOUT_H_
#define PINOUT_H_

#include <avr/io.h>

#define PORT_RX PORTD
#define DDR_RX  DDRD
#define PIN_RX  PIND
#define P_RX  PD0

#define PORT_TX PORTD
#define DDR_TX  DDRD
#define PIN_TX  PIND
#define P_TX  PD1

#define PSwitchUp     PA6
#define PORTSwitchUp    PORTA
#define DDRSwitchUp     DDRA
#define PINSwitchUp     PINA

#define PSwitchDown     PA7
#define PORTSwitchDown    PORTA
#define DDRSwitchDown   DDRA
#define PINSwitchDown   PINA

#define PBase       PB0
#define PORTBase      PORTB
#define DDRBase       DDRB
#define PINBase       PINB

#define PBaseSense      PA0
#define PORTBaseSense   PORTA
#define DDRBaseSense    DDRA
#define PINBaseSense    PINA

#define PStatusLED      PD7
#define PORTStatusLED   PORTD
#define DDRStatusLED    DDRD
```

```
#define PINStatusLED    PIND

#define PEmitterSense   PA1
#define PORTEmitterSense  PORTA
#define DDREmitterSense   DDRA
#define PINEmitterSense   PINA

#define PBrightness01   PB1
#define PBrightness02   PB2
#define PBrightness03   PB3
#define PBrightness04   PB4
#define PBrightness05   PC0
#define PBrightness06   PC1
#define PBrightness07   PC6
#define PBrightness08   PC7
#define PBrightness09   PD2
#define PBrightness10   PD3
#define PBrightness11   PD4
#define PBrightness12   PD5

#define DDRBrightness01   DDRB
#define DDRBrightness02   DDRB
#define DDRBrightness03   DDRB
#define DDRBrightness04   DDRB
#define DDRBrightness05   DDRC
#define DDRBrightness06   DDRC
#define DDRBrightness07   DDRC
#define DDRBrightness08   DDRC
#define DDRBrightness09   DDRD
#define DDRBrightness10   DDRD
#define DDRBrightness11   DDRD
#define DDRBrightness12   DDRD


#endif /* PINOUT_H_ */
```

**Source-Code D.45:** switch.h

```
#ifndef SWITCH_H_
#define SWITCH_H_

#include "types.h"

extern volatile bool switchUpFlag;
extern volatile bool switchDownFlag;

void switchInitialize();

void switchLoop();

#endif /* SWITCH_H_ */
```

**Source-Code D.46:** switch.c

```
#include "switch.h"

#include "types.h"
#include <util/delay.h>
#include "pinout.h"

bool switchUpPressed = False;
bool switchDownPressed = False;

volatile bool switchUpFlag = False;
volatile bool switchDownFlag = False;

void switchInitialize(){
  //activate Pull Ups
  PORTSwitchUp |= (1<<PSwitchUp);
```

```
  PORTSwitchDown |= (1<<PSwitchDown);
  _delay_ms(2000);
}

void switchLoop(){
  if((~PINSwitchUp) & (1<<PSwitchUp)){ //If switch "Up" is pressed
    if (!switchUpPressed){ //and this is new,
      switchUpFlag = True; //the flag is set
      switchUpPressed = True; //and the status is updated
    }
  }else{ //If the button "up" is not pressed
    if (switchUpPressed) switchUpPressed = False; //and this is new, the status get
        updated
  }
  //The same for the "Down" button
  if((~PINSwitchDown) & (1<<PSwitchDown)){
    if (!switchDownPressed){
      switchDownFlag = True;
      switchDownPressed = True;
    }
  }else{
    if (switchDownPressed) switchDownPressed = False;
  }
}
```

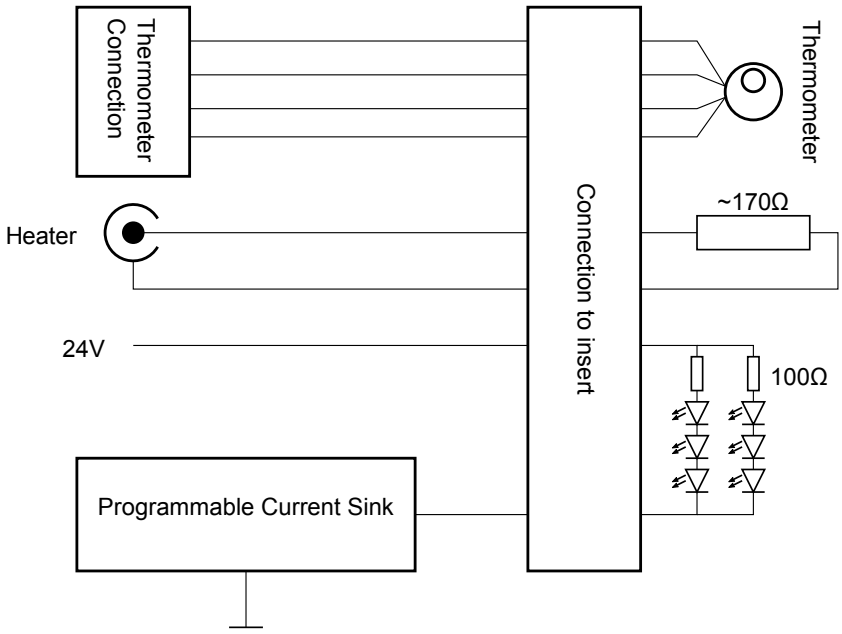



**Figure D.41.:** Simplified block diagram of the auxiliary wiring NIM insert

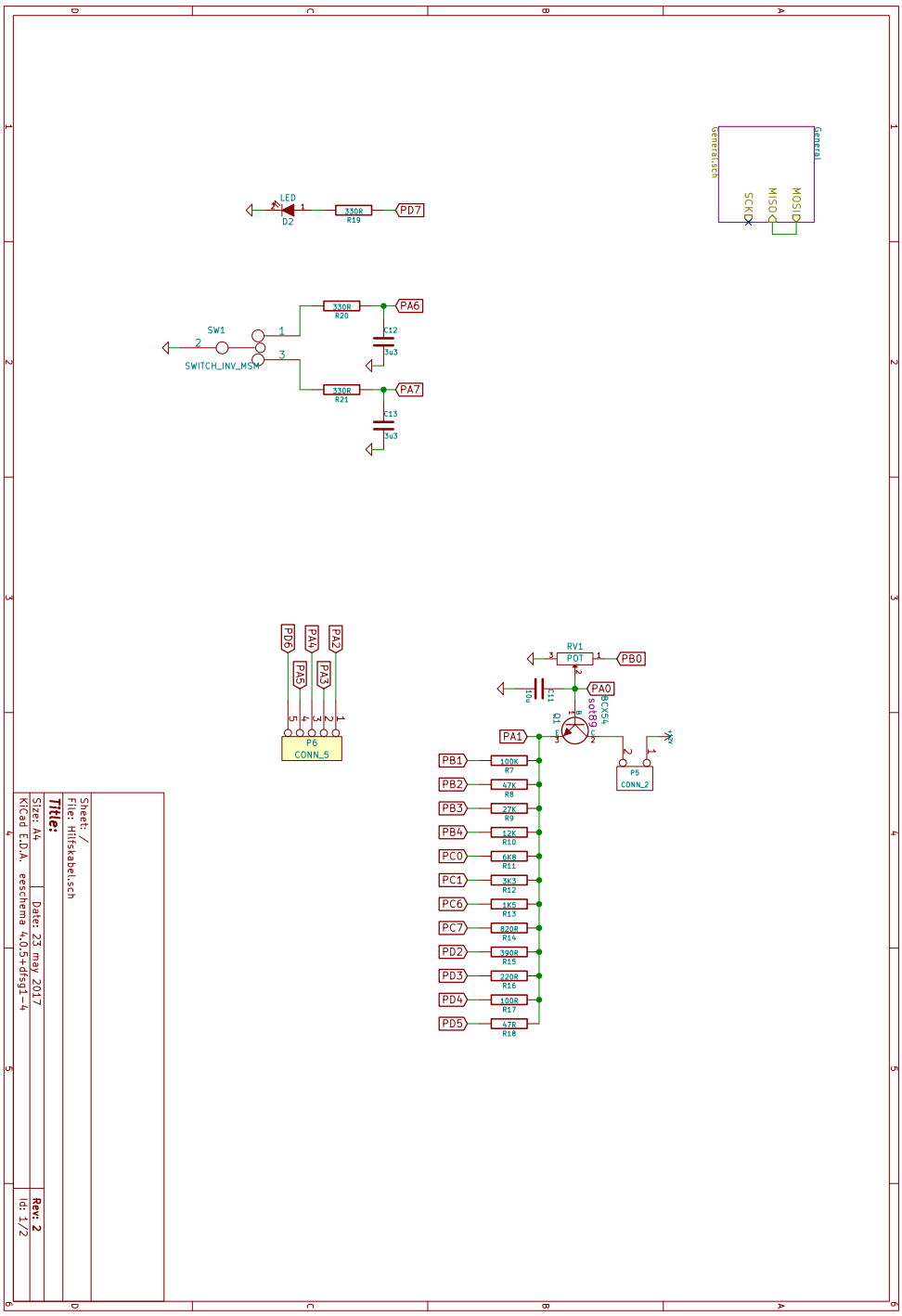
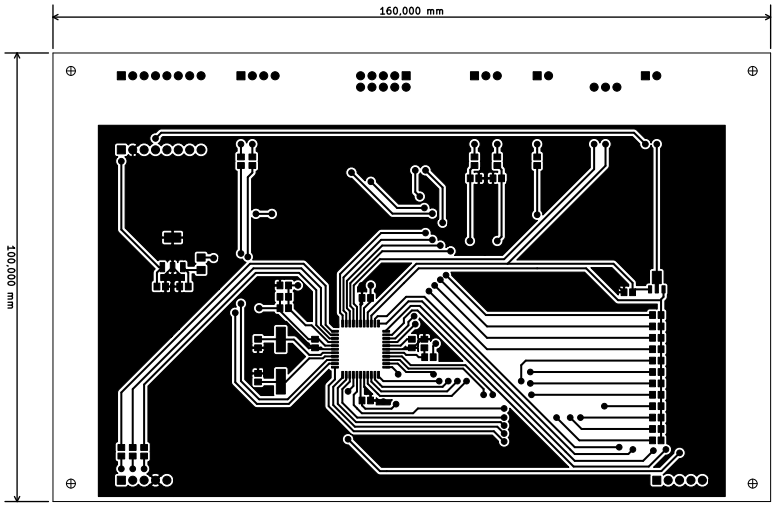
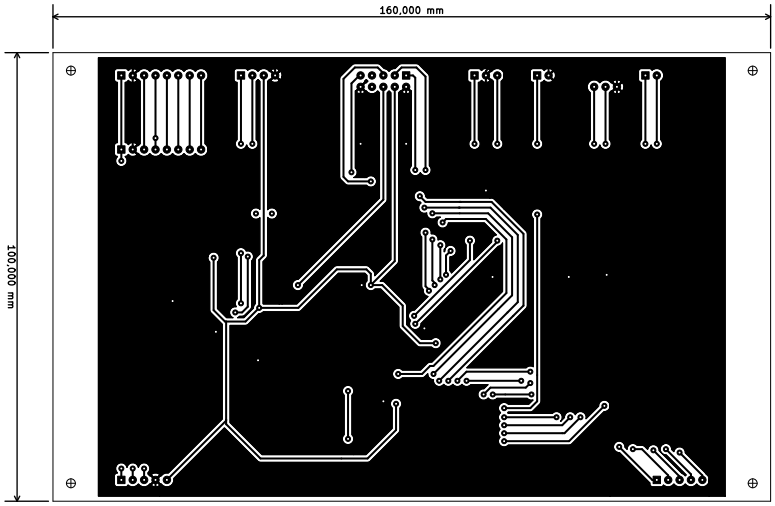


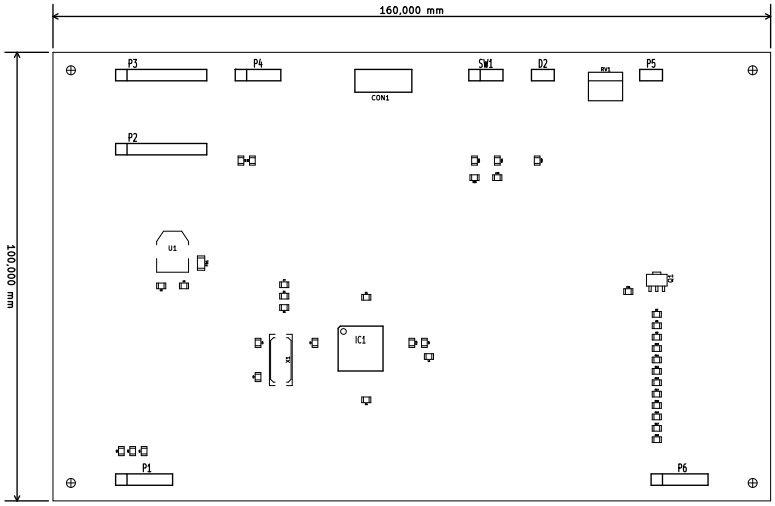

**Figure D.42.:** Schematic of the auxiliary wiring NIM insert




**(a)** Top side copper structure

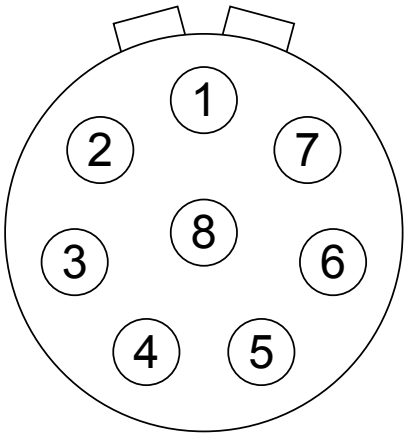



**(b)** Bottom side copper structure



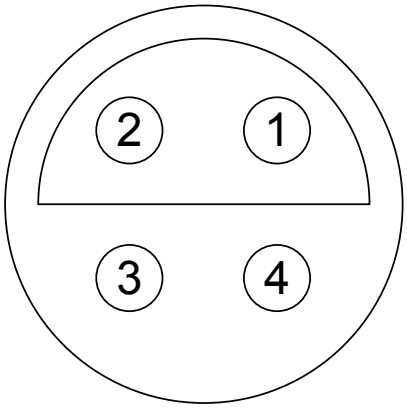

**(c)** Component references

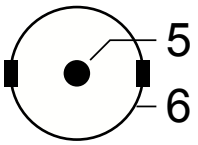
**Figure D.43.:** Circuit board layout of the auxiliary wiring NIM insert




| | | |
|---|---|---|
| 1 | U+ thermometer | |
| 2 | U− thermometer | |
| 3 | I+ thermometer | |
| 4 | I− thermometer | |
| 5 | Heater high | |
| 6 | Heater low | |
| 7 | LED 24 V | output |
| 8 | LED current limited ground | output |

**(a)** Connection to the sample chamber




| | |
|---|---|
| 1 | U+ thermometer |
| 2 | U− thermometer |
| 3 | I− thermometer |
| 4 | I+ thermometer |
| 5 | Heater high |
| 6 | Heater low |

**(b)** Connection to the temperature controller

**Figure D.44.:** Pin configuration of the auxiliary wiring NIM insert


**(a)** front **(b)** back

**Figure D.45.:** Photos of the outside of the auxiliary wiring NIM insert

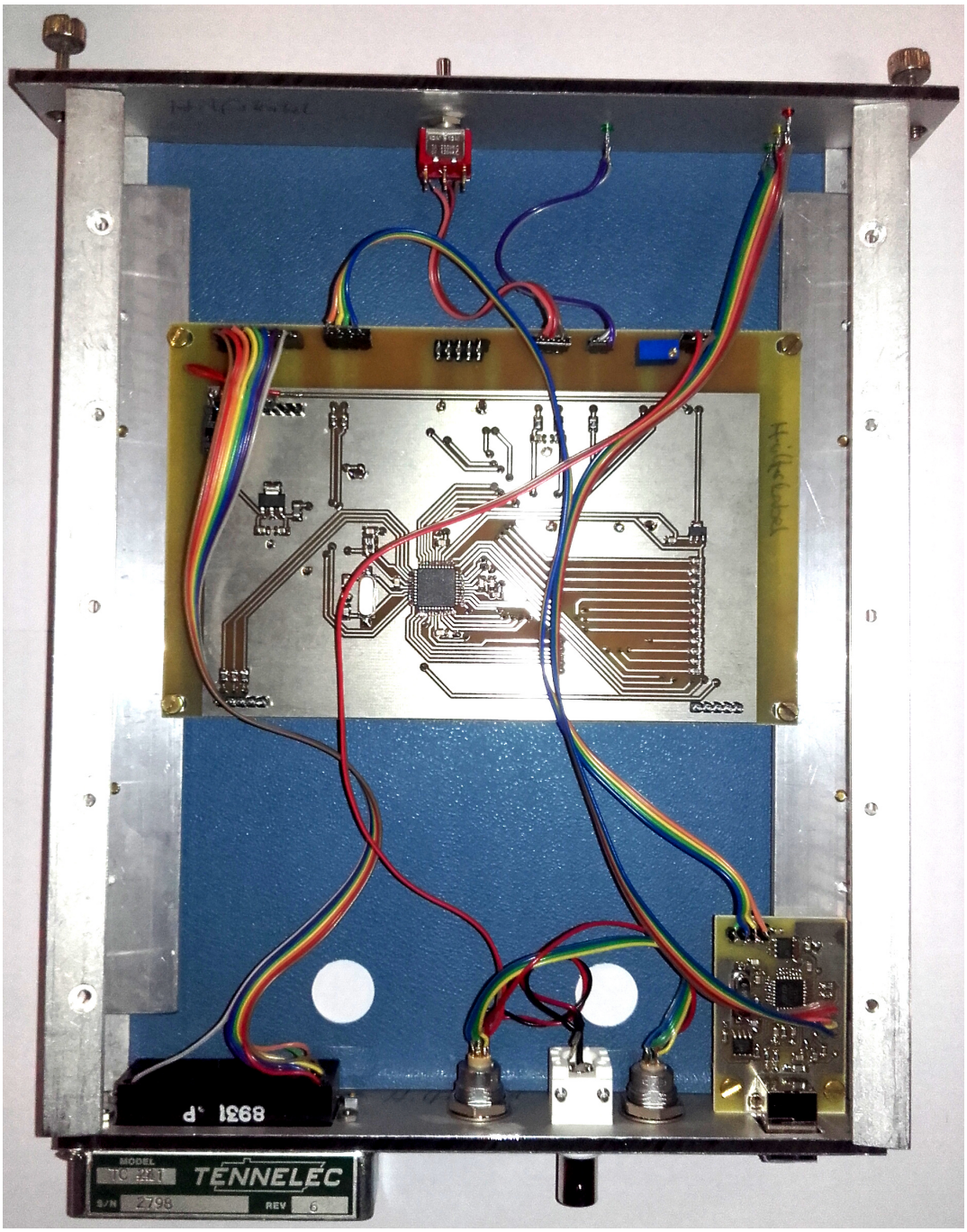



**Figure D.46.:** Photo of the inside of the auxiliary wiring NIM insert

## D.7. Liquid nitrogen level meter

The purpose of the liquid nitrogen insert is to measure the level of the nitrogen in the bath. This is done by a a capacitive sensor: the liquid rises between two tubes which form a cylindrical capacitor. Since the liquid has a different dielectric constant than the gas, the capacitance changes as a function of the nitrogen level. This capacitor is connected to an LC tank circuit which is driven at its resonance frequency. The change of this frequency is easily detected by a hardware counter in the microcontroller.
The frequencies for an empty and a full tank have to be set manually. For this purpose, there are two switches on the front. When they are pressed, the current frequency is stored as the empty or full frequency. Two red LEDs next to the switches indicate if the switch is active. To prevent accidental setting of the limits, there is an additional recessed switch that disables the user input under normal conditions. The level of the bath is indicated by a LED bargraph.
A USB port allows to connect to the microcontroller such that the level can be monitored by software. The controller supports the following commands.

- `*IDN?`
  This command returns the name of the insert, the author, the version, and date of the firmware. The output is three lines long.
- `:LEVel?`
  This command returns the nitrogen level in the format `%03d%%` in %.
- `:FREquency?`
  This command returns the current frequency in the format `%06ulHz` in Hz.
- `:FREquency:EMPty?`
  This command returns the frequency stored for an empty reading in the format `%06ulHz` in Hz.
- `:FREquency:FULl?`
  This command returns the frequency stored for a full reading in the format `%06ulHz` in Hz.

A block diagram of the module is shown in figure D.47. The full schematics of the circuit boards are shown in figures D.48 and D.49 and the corresponding board layout in figures D.50 and D.51. The pinout of the connectors is shown in figure D.52 and photos of the front, back, inside, and the sensor are shown in figures D.53 to D.55. The firmware consists of the following files.

**Source-Code D.47:** bargraph.h

```
#ifndef BARGRAPH_H_
#define BARGRAPH_H_

#include "types.h"

void bargraphInitialize();
void bargraphValue(uint8_t value);

#endif /* BARGRAPH_H_ */
```

**Source-Code D.48:** bargraph.c

```
#include "bargraph.h"
#include "types.h"
#include "pinout.h"

void bargraphInitialize(){
  DDRBargraph0 |= (1<<PBargraph0);
  DDRBargraph1 |= (1<<PBargraph1);
  DDRBargraph2 |= (1<<PBargraph2);
  DDRBargraph3 |= (1<<PBargraph3);
  DDRBargraph4 |= (1<<PBargraph4);
  DDRBargraph5 |= (1<<PBargraph5);
  DDRBargraph6 |= (1<<PBargraph6);
  DDRBargraph7 |= (1<<PBargraph7);
  DDRBargraph8 |= (1<<PBargraph8);
  DDRBargraph9 |= (1<<PBargraph9);
}

void bargraphValue(uint8_t value){
  PORTBargraph0 &= ~(1<<PBargraph0);
  PORTBargraph1 &= ~(1<<PBargraph1);
  PORTBargraph2 &= ~(1<<PBargraph2);
  PORTBargraph3 &= ~(1<<PBargraph3);
  PORTBargraph4 &= ~(1<<PBargraph4);
  PORTBargraph5 &= ~(1<<PBargraph5);
  PORTBargraph6 &= ~(1<<PBargraph6);
  PORTBargraph7 &= ~(1<<PBargraph7);
  PORTBargraph8 &= ~(1<<PBargraph8);
  PORTBargraph9 &= ~(1<<PBargraph9);
  if(value>0) PORTBargraph0 |= (1<<PBargraph0);
  if(value>1) PORTBargraph1 |= (1<<PBargraph1);
  if(value>2) PORTBargraph2 |= (1<<PBargraph2);
  if(value>3) PORTBargraph3 |= (1<<PBargraph3);
  if(value>4) PORTBargraph4 |= (1<<PBargraph4);
  if(value>5) PORTBargraph5 |= (1<<PBargraph5);
  if(value>6) PORTBargraph6 |= (1<<PBargraph6);
  if(value>7) PORTBargraph7 |= (1<<PBargraph7);
  if(value>8) PORTBargraph8 |= (1<<PBargraph8);
  if(value>9) PORTBargraph9 |= (1<<PBargraph9);
}
```

**Source-Code D.49:** eeprom.h

```
/*
 * eeprom.h
 *
 *  Created on: 21.05.2017
 *      Author: thomas
 */

#ifndef EEPROM_H_
#define EEPROM_H_

#include "types.h"

uint8_t read8bit(uint16_t address);
uint16_t read16bit(uint16_t address);
void write8bit(uint16_t address, uint8_t data);
void write16bit(uint16_t address, uint16_t data);

#endif /* EEPROM_H_ */
```

**Source-Code D.50:** eeprom.c

```
#include "types.h"
#include "eeprom.h"
#include "pinout.h"
```

```
uint8_t read8bit(uint16_t address){
  while(EECR & (1<<EEWE));  //Wait for completion of previous write
  EEAR = address;       //set up reading address
  EECR |= (1<<EERE);      //Start eeprom read by writing EERE
  return EEDR;        //Return data from data register
}
uint16_t read16bit(uint16_t address){
  uint16_t result = (read8bit(address)<<8);
  result |= read8bit(address+1);
  return result;
}
void write8bit(uint16_t address, uint8_t data){
  while(EECR & (1<<EEWE));  //Wait for completion of previous write
  EEAR = address;       //Set up address and data registers
  EEDR = data;
  EECR |= (1<<EEMWE);     //Write logical one to EEMWE
  EECR |= (1<<EEWE);      //Start eeprom write by setting EEWE
}
void write16bit(uint16_t address, uint16_t data){
  write8bit(address,data>>8);
  write8bit(address+1,data&0x00ff);
}
```

**Source-Code D.51:** LEDs.h

```
#ifndef LEDS_H_
#define LEDS_H_

void LEDsInitialize();
void LEDsOn();
void LEDsOff();

#endif /* LEDS_H_ */
```

**Source-Code D.52:** LEDs.c

```
#include "LEDs.h"
#include "pinout.h"

void LEDsInitialize(){
  DDRLEDSwitchFullActive |= (1<<PLEDSwitchFullActive);
  DDRLEDSwitchEmptyActive |= (1<<PLEDSwitchEmptyActive);
}

void LEDsOn(){
  PORTLEDSwitchFullActive |= (1<<PLEDSwitchFullActive);
  PORTLEDSwitchEmptyActive |= (1<<PLEDSwitchEmptyActive);
}

void LEDsOff(){
  PORTLEDSwitchFullActive &= ~(1<<PLEDSwitchFullActive);
  PORTLEDSwitchEmptyActive &= ~(1<<PLEDSwitchEmptyActive);
}
```

**Source-Code D.53:** main.h

```
#ifndef MAIN_H_
#define MAIN_H_

#include "types.h"

extern bool requestPicture;

bool startExposure();
bool stopExposure();
bool getExposure();
bool startFocus();
bool stopFocus();
```

```
bool getFocus();

#endif /* MAIN_H_ */
```

**Source-Code D.54:** main.c

```
#include "serial.h"
#include <avr/interrupt.h>
#include <util/delay.h>

#include "timer.h"
#include "bargraph.h"
#include "switch.h"

int main(){
  switchInitialize();
  bargraphInitialize();
  timerInitialize();
  initializeSerialPort();
  sei();
  while(True){
    switchUpdate();
  }
}
```

**Source-Code D.55:** parser.h

```
#ifndef PARSER_H_
#define PARSER_H_

#include "types.h"

extern volatile char command[100];

void parseCommand();

#endif /* PARSER_H_ */
```

**Source-Code D.56:** parser.c

```
#include "parser.h"
#include "types.h"
#include "serial.h"
#include "timer.h"
#include <stdio.h>

volatile char command[100];

bool compareCommand(char reference[]){
  uint8_t i = 0;
  while(reference[i]){
    if(command[i] != reference[i]) return false;
    i++;
  }
  return true;
}

void parseCommand(){
  if(compareCommand("*IDN?")){
    sendString("LN2␣level␣metering␣unit\r\nby␣Thomas.Moeller@uni.kn\r\nFirmware␣1␣
        (21.05.2017)");
    return;
  }
  if(compareCommand(":LEV?")){
    char temp[16];
    sprintf(temp,"%03d%%",percent);
    sendString(temp);
    return;
```

```
  }
  if(compareCommand(":FRE?")){
    char temp[16];
    sprintf(temp,"%06luHz",frequency*3600UL/256); //freq*3686400/1024/256
    sendString(temp);
    return;
  }
  if(compareCommand(":FRE:FUL?")){
    char temp[16];
    sprintf(temp,"%06luHz",fullFrequency*3600UL/256); //freq*3686400/1024/256
    sendString(temp);
    return;
  }
  if(compareCommand(":FRE:EMP?")){
    char temp[16];
    sprintf(temp,"%06luHz",emptyFrequency*3600UL/256); //freq*3686400/1024/256
    sendString(temp);
    return;
  }
  sendString("command␣unknown");
}
```

**Source-Code D.57:** pinout.h

```
#ifndef PINOUT_H_
#define PINOUT_H_

#include <avr/io.h>

#define PORT_RX PORTD
#define DDR_RX  DDRD
#define PIN_RX  PIND
#define P_RX  PD0

#define PORT_TX PORTD
#define DDR_TX  DDRD
#define PIN_TX  PIND
#define P_TX  PD1

#define PSwitchFull     PA2
#define PORTSwitchFull    PORTA
#define DDRSwitchFull   DDRA
#define PINSwitchFull   PINA

#define PSwitchEmpty    PA1
#define PORTSwitchEmpty   PORTA
#define DDRSwitchEmpty    DDRA
#define PINSwitchEmpty    PINA

#define PSwitchUnlock   PA0
#define PORTSwitchUnlock  PORTA
#define DDRSwitchUnlock   DDRA
#define PINSwitchUnlock   PINA

#define PLEDSwitchFullActive  PB0
#define PORTLEDSwitchFullActive PORTB
#define DDRLEDSwitchFullActive  DDRB
#define PINLEDSwitchFullActive  PINB

#define PLEDSwitchEmptyActive   PB2
#define PORTLEDSwitchEmptyActive  PORTB
#define DDRLEDSwitchEmptyActive   DDRB
#define PINLEDSwitchEmptyActive   PINB

#define PBargraph0      PD2
#define PBargraph1      PD3
#define PBargraph2      PD4
#define PBargraph3      PD5
#define PBargraph4      PD6
```

```
#define PBargraph5      PD7
#define PBargraph6      PC0
#define PBargraph7      PC1
#define PBargraph8      PC6
#define PBargraph9      PC7

#define PORTBargraph0   PORTD
#define PORTBargraph1   PORTD
#define PORTBargraph2   PORTD
#define PORTBargraph3   PORTD
#define PORTBargraph4   PORTD
#define PORTBargraph5   PORTD
#define PORTBargraph6   PORTC
#define PORTBargraph7   PORTC
#define PORTBargraph8   PORTC
#define PORTBargraph9   PORTC

#define DDRBargraph0    DDRD
#define DDRBargraph1    DDRD
#define DDRBargraph2    DDRD
#define DDRBargraph3    DDRD
#define DDRBargraph4    DDRD
#define DDRBargraph5    DDRD
#define DDRBargraph6    DDRC
#define DDRBargraph7    DDRC
#define DDRBargraph8    DDRC
#define DDRBargraph9    DDRC

#endif /* PINOUT_H_ */
```

**Source-Code D.58:** switch.h

```
#ifndef SWITCH_H_
#define SWITCH_H_

#include "types.h"

void switchInitialize();

void switchUpdate();

#endif /* SWITCH_H_ */
```

**Source-Code D.59:** switch.c

```
#include "switch.h"

#include "pinout.h"
#include <util/delay.h>
#include "timer.h"
#include "LEDs.h"

void switchInitialize(){
  //activate Pull Ups
  PORTSwitchUnlock |= (1<<PSwitchUnlock);
  PORTSwitchFull |= (1<<PSwitchFull);
  PORTSwitchEmpty |= (1<<PSwitchEmpty);
  _delay_ms(2000);
  LEDsInitialize();
}

void switchUpdate(){
  if(PINSwitchUnlock & (1<<PSwitchUnlock)){
    LEDsOff();
    return;
  }
  LEDsOn();
  if((~PINSwitchFull) & (1<<PSwitchFull)){
    setFull();
```

```
  }
  if((~PINSwitchEmpty) & (1<<PSwitchEmpty)){
    setEmpty();
  }
}
```

**Source-Code D.60:** timer.h

```
#ifndef TIMER_H_
#define TIMER_H_

#include "types.h"

extern volatile uint8_t percent;
extern volatile uint16_t frequency;
extern volatile uint16_t emptyFrequency;
extern volatile uint16_t fullFrequency;

void timerInitialize();
void setFull();
void setEmpty();

#endif /* TIMER_H_ */
```

**Source-Code D.61:** timer.c

```
#include "types.h"
#include "timer.h"
#include "eeprom.h"
#include "pinout.h"
#include <avr/interrupt.h>
#include "bargraph.h"

volatile uint8_t percent;
volatile uint16_t frequency;
volatile uint16_t emptyFrequency;
volatile uint16_t fullFrequency;

void timerInitialize(){
  fullFrequency = read16bit(0);
  emptyFrequency = read16bit(2);
  TCCR0 |= (1<<CS02) | (1<<CS00); //Timer0 prescaler 1024
  TIMSK |= (1<<TOIE0);      //Timer0 overflow interrupt enable
  TCCR1B |= (1<<CS12) | (1<<CS11);  //Timer1 triggered my falling edge on T1
}
void setFull(){
  fullFrequency = frequency;
  write16bit(0,fullFrequency);
}
void setEmpty(){
  emptyFrequency = frequency;
  write16bit(2,emptyFrequency);
}
ISR(TIMER0_OVF_vect){
  frequency = TCNT1;
  TCNT1 = 0;
  if(frequency < fullFrequency){
    percent = 100;
  }else if(frequency > emptyFrequency){
    percent = 0;
  }else{
    percent = (emptyFrequency-frequency)*100ul/(emptyFrequency-fullFrequency);
  }
  bargraphValue(percent/9);
}
```

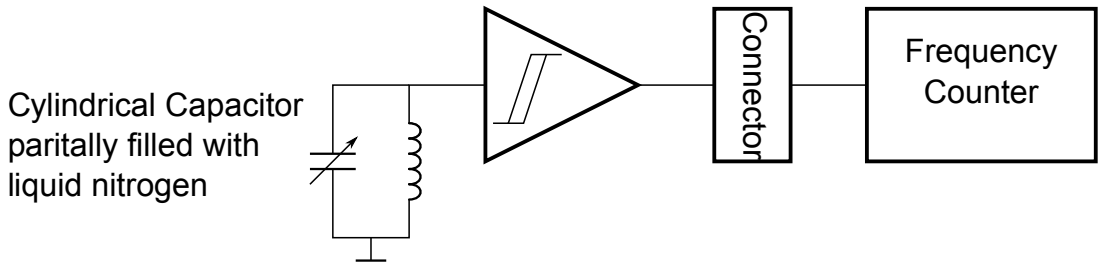



**Figure D.47.:** Simplified block diagram of the liquid nitrogen NIM insert

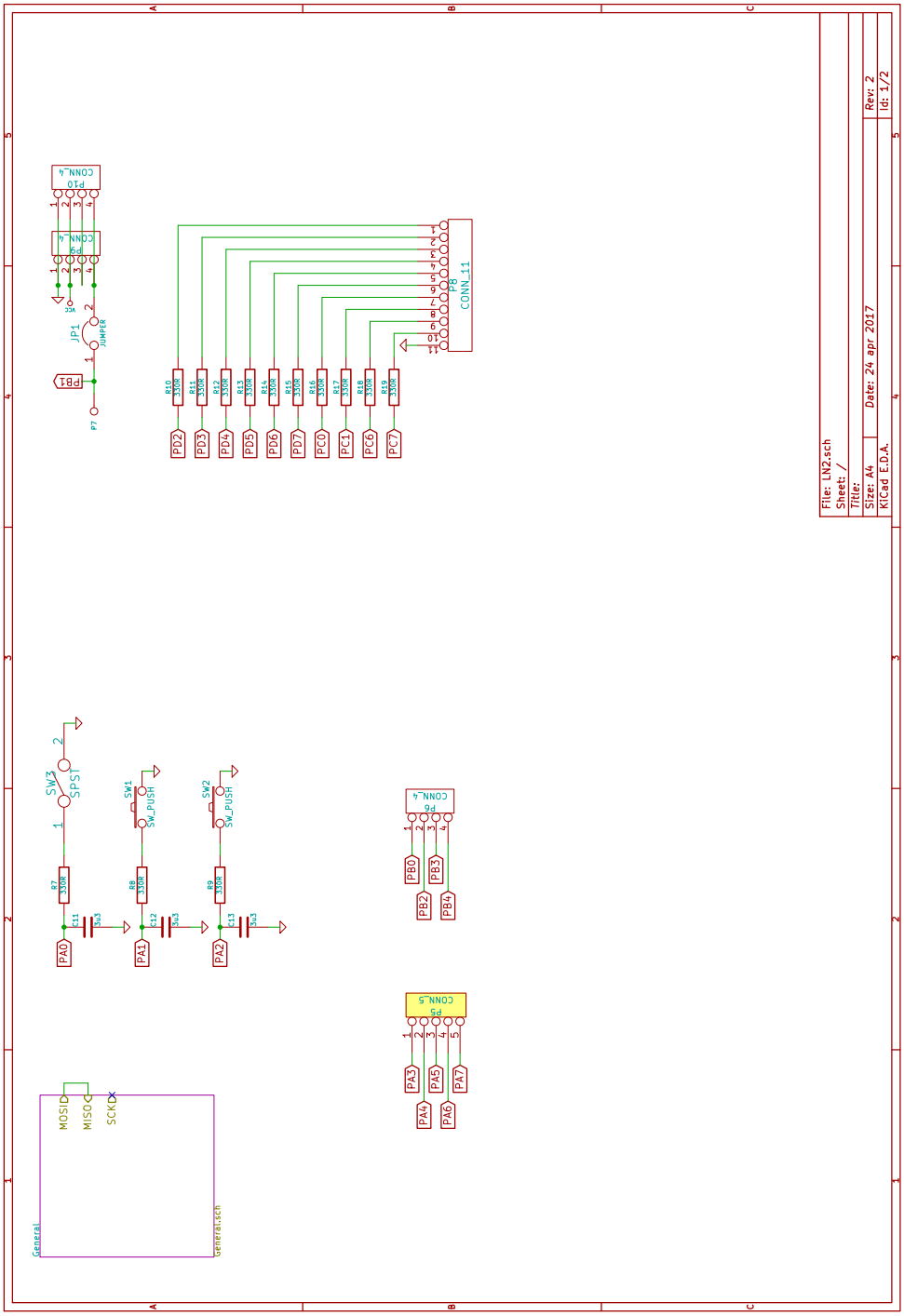



**Figure D.48.:** Schematic of the liquid nitrogen NIM insert

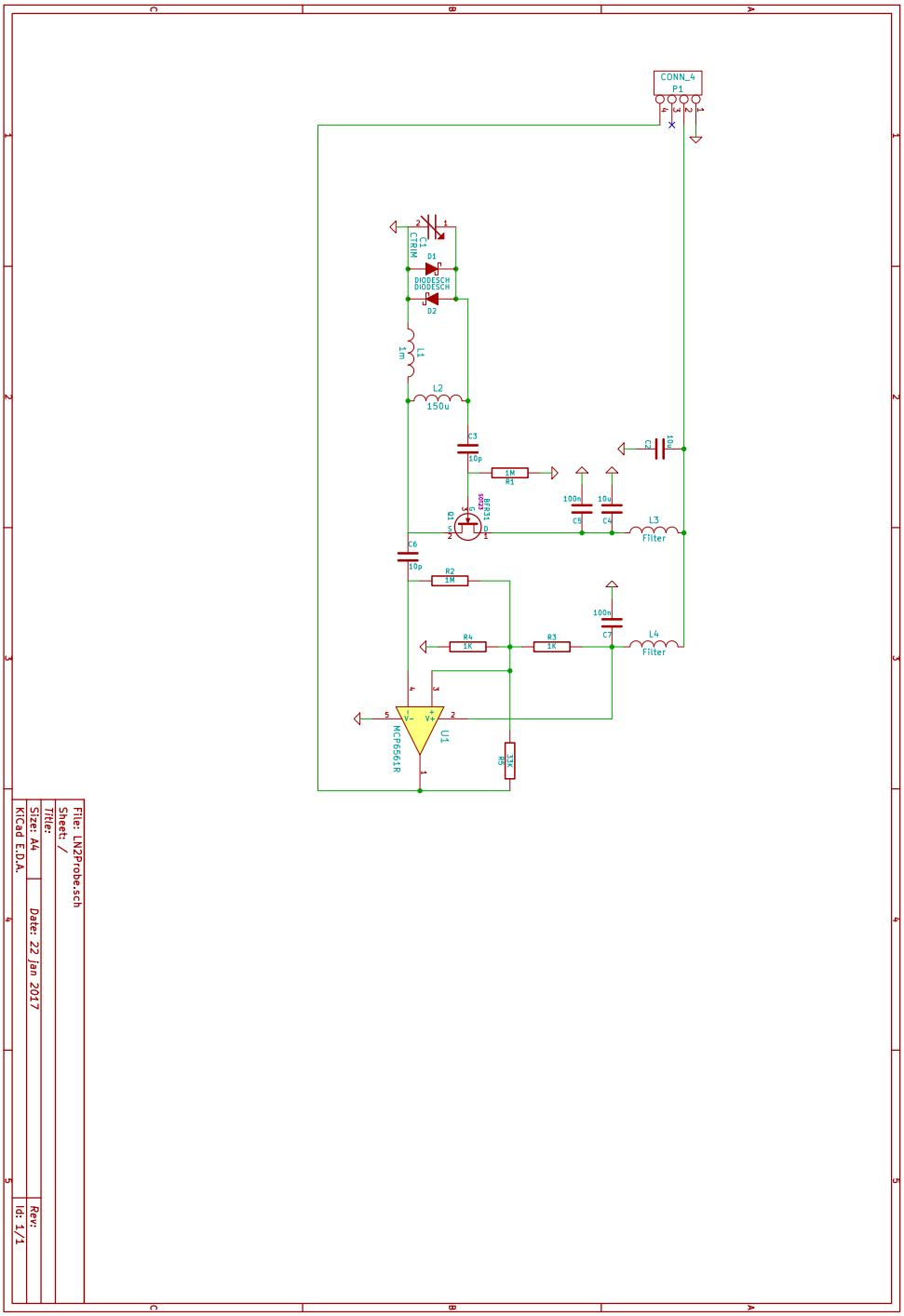
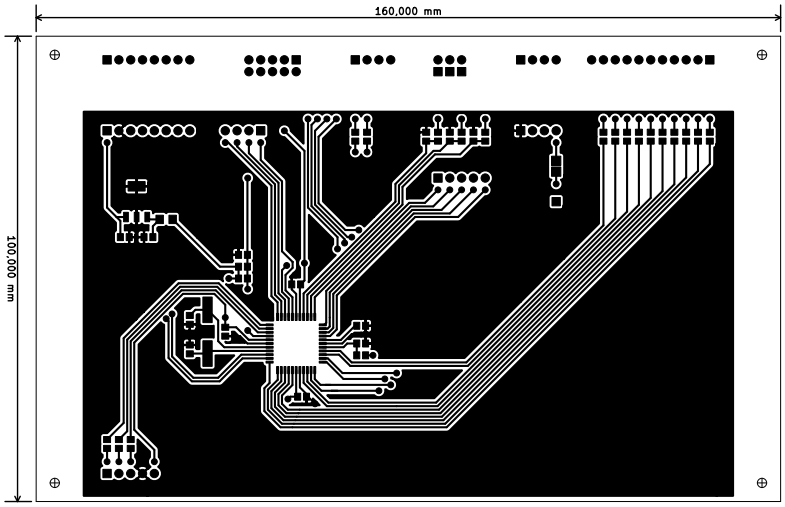
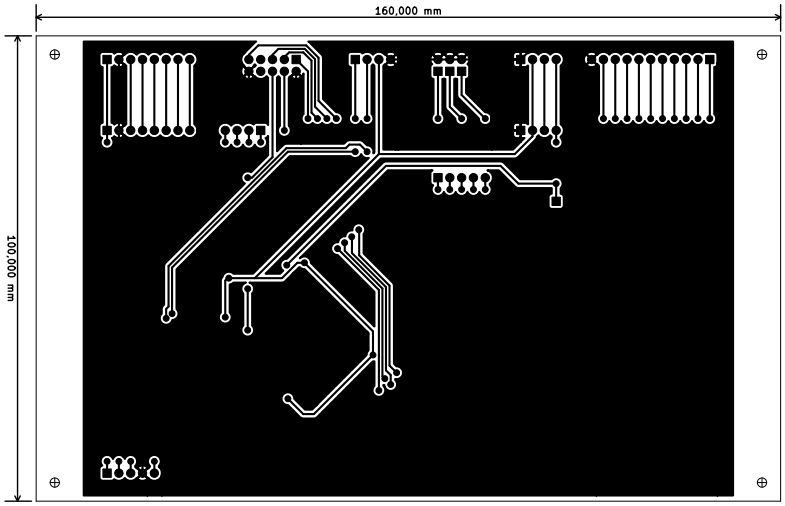


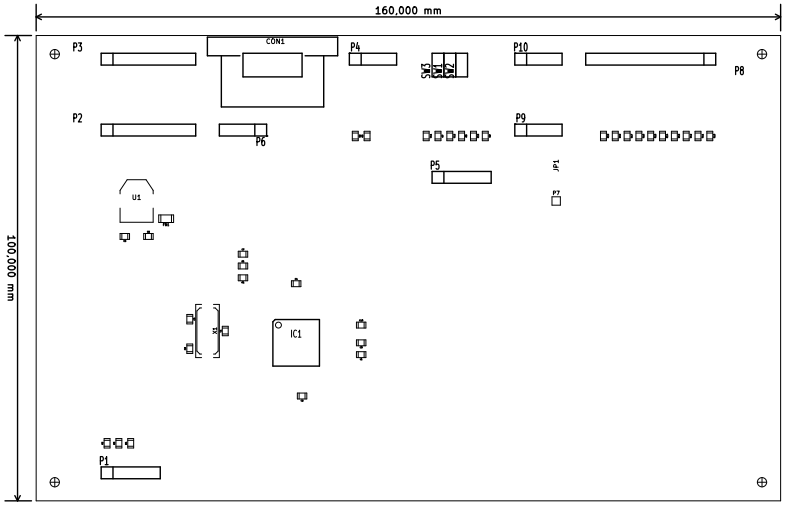

**Figure D.49.:** Schematic of the probe for the liquid nitrogen NIM insert




**(a)** Top side copper structure

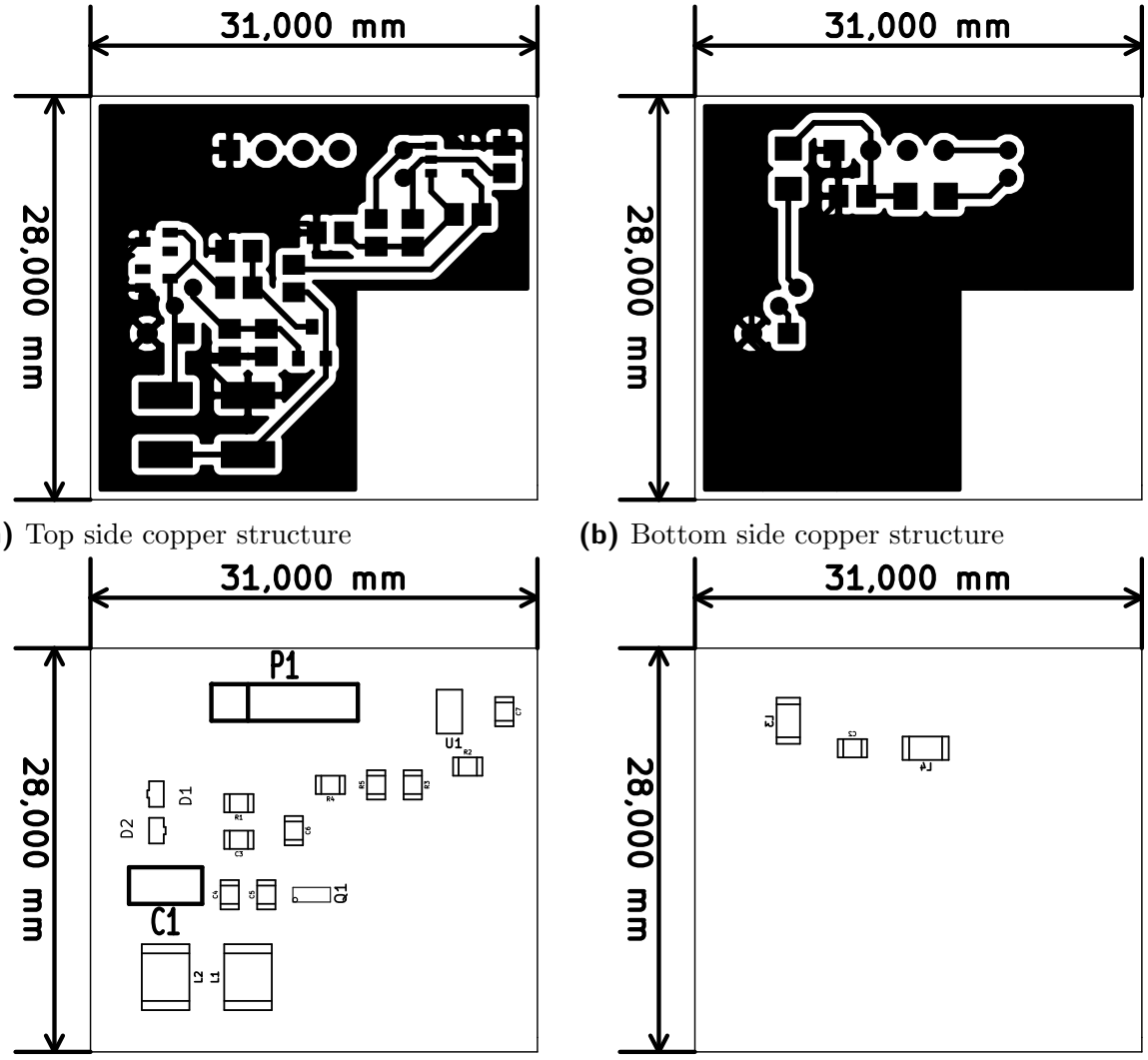



**(b)** Bottom side copper structure




**(c)** Component references

**Figure D.50.:** Circuit board layout of the liquid nitrogen NIM insert



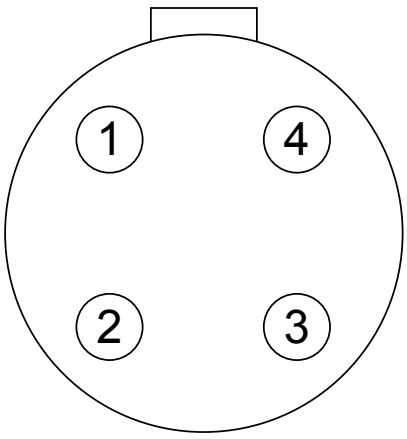

**(a)** Top side copper structure

**(b)** Bottom side copper structure

**(c)** Component references front side

**(d)** Component references back side

**Figure D.51.:** Circuit board layout of the probe for the liquid nitrogen NIM insert




| | | |
|---|---|---|
| 1 | Ground | output |
| 2 | 5 V | output |
| 3 | not connected | |
| 4 | frequency input | input |

**Figure D.52.:** Pin configuration of the liquid nitrogen NIM insert


**(a)** front **(b)** back

**Figure D.53.:** Photos of the outside of the liquid nitrogen NIM insert

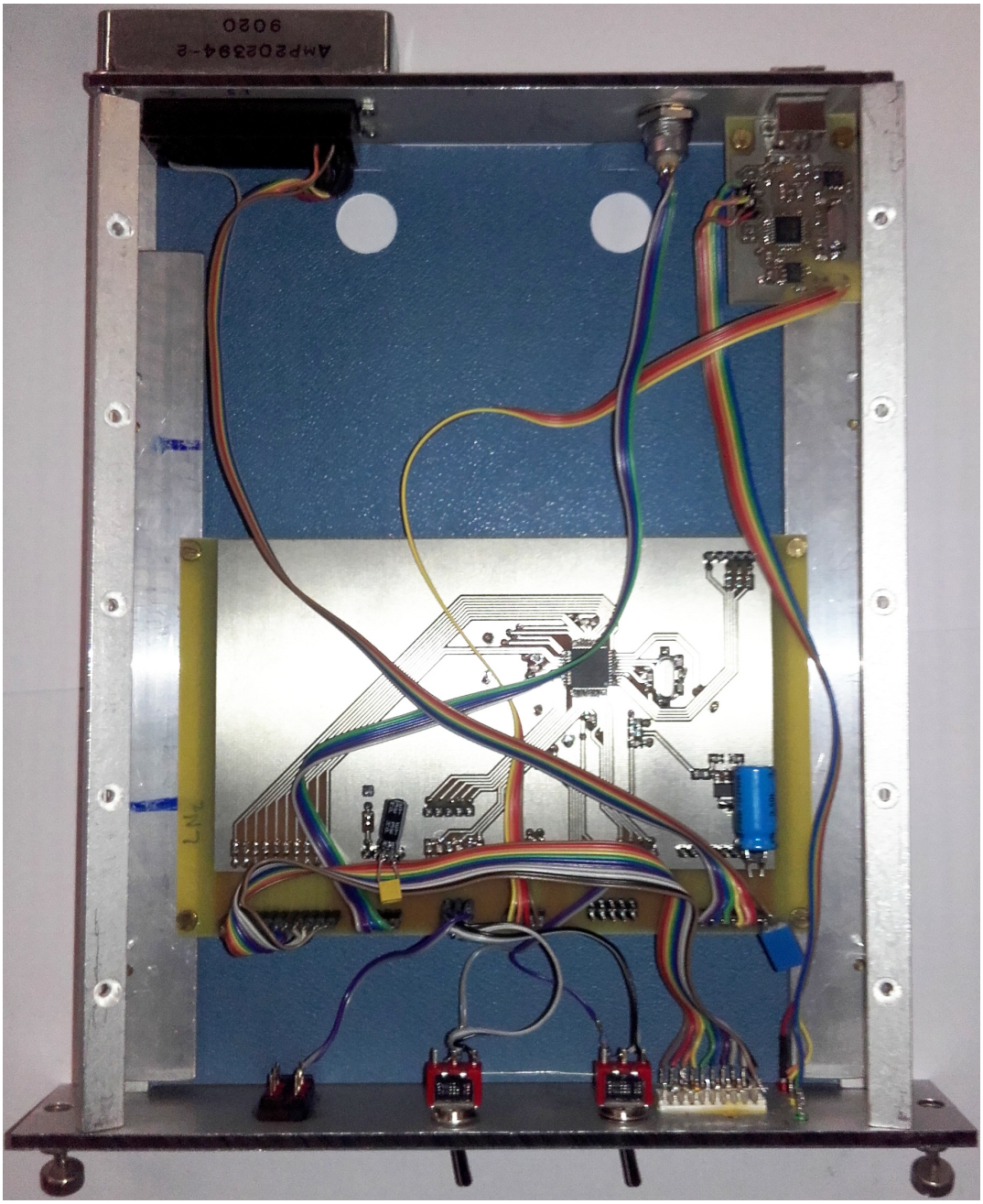



**Figure D.54.:** Photo of the inside of the liquid nitrogen NIM insert

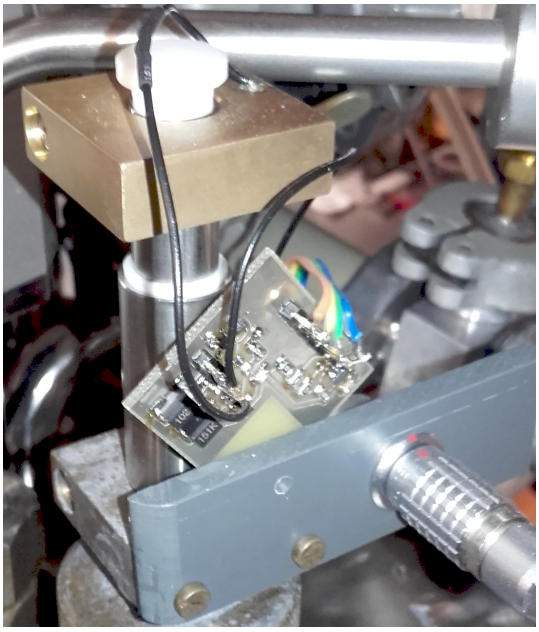


**Figure D.55.:** Photo of the sensor of the liquid nitrogen NIM insert

## D.8. Gascounter

The purpose of the gascounter insert is to measure the rate of helium evaporation. The counter in the recovery line contains a magnet on the last digit. A reed relay is attached to the counter and gives a pulse for every ten liters. Even though this relay is debounced, miscounts can occur at low evaporation rates.
The insert has a display on the front where the current evaporation rate is shown. Since the evaporation rate is calculated from the time between the pulses, there will be some lag in the readout. This is especially noticeable when the rate drops to zero.
A USB port allows to connect to the microcontroller to read out the rate. The microcontroller supports the following commands.

- `*IDN?`
  This command returns the name of the insert, the author, the version, and date of the firmware. The output is three lines long.

- `:RATe?`
  This command returns the current evaporation rate in the format `%09.6f/s` in $\mathrm{L\,s^{-1}}$.

- `:ABSolute?`
  This command returns the approximate reading of the gascounter in the format `%08lul` in L.

- `:ABSulute␣<Number>`
  This command sets the the reading of the gascounter to the given number in L. This is used after the power to the insert was interrupted, or if miscounts occurred.

A block diagram of the module is shown in figure D.56. The full schematic of the circuit board is shown in figure D.57 and the corresponding board layout in figure D.58. The pinout of the connectors is shown in figure D.59 and photos of the front, back, and inside are shown in figures D.60 and D.61. The firmware consists of the following files.

**Source-Code D.62:** display.h

```
#ifndef DISPLAY_H_
#define DISPLAY_H_

void displayInitialize();
void displayUpdate();

void outputString(char* string);

#endif /* DISPLAY_H_ */
```

**Source-Code D.63:** display.c

```
#include "display.h"
#include "pinout.h"
#include <stdio.h>
#include "timer.h"
#include <util/delay.h>

void displayInitialize(){
  PORTOE |= (1<<POE);       //Disable the outputs
  DDRSS |= (1<<PSS);        //Force SS Line up to not interrrupt the transmission
  DDRMOSI |= (1<<PMOSI);
```

```
  DDRSCK |= (1<<PSCK);
  DDROE |= (1<<POE);
  DDRRCK |= (1<<PRCK);
  SPCR |= (1<<SPR0);         //Frequency 3686400 / 16 = 230400Hz
  SPCR |= (1<<MSTR);         //make me "master of transmission"
  SPCR |= (1<<SPE);        //enable SPI
  DDRLEDHours |= (1<<PLEDHours);
  DDRLEDMinutes |= (1<<PLEDMinutes);
  DDRLEDSeconds |= (1<<PLEDSeconds);
}

/*
 *   2222
 *  7    3
 *  7    3
 *   8888
 *  6    4
 *  6    4
 *   5555  1
 */

char numbers[] = {
    0b01111110, //0
    0b00001100, //1
    0b10110110, //2
    0b10011110, //3
    0b11001100, //4
    0b11011010, //5
    0b11111010, //6
    0b00001110, //7
    0b11111110, //8
    0b11011110  //9
};
/*char numbers[] = {
    63,
    6,
    91,
    79,
    102,
    109,
    125,
    7,
    127,
    111
};*/

char charTo7Segment(char input){
  if((input>='0') && (input<='9')){
    return numbers[input-'0'];
  }
  if(input == '.') return 0b00000001;
  return 0;
}

void outputString(char* string){
  uint8_t i=0;
  uint8_t temp;
  while(string[i]){
    temp = charTo7Segment(string[i]);
    if(string[i+1] == '.'){
      temp |= charTo7Segment('.');  //add comma to digit
      i++;
    }
    SPDR = temp;          //send data
    while(~SPSR & (1<<SPIF));   //wait for transmission to complete
    i++;
  }
  PORTRCK |= (1<<PRCK);     //Make rising edge to transfer data to output register
  _delay_us(2);
  PORTRCK &= ~(1<<PRCK);
```

```
  PORTOE &= ~(1<<POE);      //Enable the outputs (Will only have effect on the first call
      )
}

void displayUpdate(){
  double temp = rate;
  uint8_t scale = 0;
  if(temp<1){
    temp *= 60.0;
    scale=1;
  }
  if(temp<1){
    temp *= 60.0;
    scale=2;
  }
  if(scale==0){
    PORTLEDHours &= ~(1<<PLEDHours);
    PORTLEDMinutes &= ~(1<<PLEDMinutes);
    PORTLEDSeconds |= (1<<PLEDSeconds);
  }else if(scale==1){
    PORTLEDHours &= ~(1<<PLEDHours);
    PORTLEDMinutes |= (1<<PLEDMinutes);
    PORTLEDSeconds &= ~(1<<PLEDSeconds);
  }else{
    PORTLEDHours |= (1<<PLEDHours);
    PORTLEDMinutes &= ~(1<<PLEDMinutes);
    PORTLEDSeconds &= ~(1<<PLEDSeconds);
  }
  char string[8];
  if(temp>9.95){
    sprintf(string,"%2.0f",temp);
  }else{
    sprintf(string,"%3.1f",temp);
  }
  outputString(string);
}
```

**Source-Code D.64:** hardware.h

```
#ifndef HARDWARE_H_
#define HARDWARE_H_

void hardwareInitialize();
void hardwareFlash();
void hardwareScan();

#endif /* HARDWARE_H_ */
```

**Source-Code D.65:** hardware.c

```
#include "hardware.h"
#include "pinout.h"
#include <util/delay.h>

void hardwareInitialize(){
  PORTDetectorPresent |= (1<<PDetectorPresent);
  DDRLEDDetectorPresent |= (1<<PLEDDetectorPresent);
  DDRLEDFlash |= (1<<PLEDFlash);
}

void hardwareFlash(){
  PORTLEDFlash |= (1<<PLEDFlash);
  _delay_ms(100);
  PORTLEDFlash &= ~(1<<PLEDFlash);
}

void hardwareScan(){
  if(PINDetectorPresent & (1<<PDetectorPresent)){
    PORTLEDDetectorPresent |= (1<<PLEDDetectorPresent);
```

```
  }else{
    PORTLEDDetectorPresent &= ~(1<<PLEDDetectorPresent);
  }
}
```

### Source-Code D.66: main.c

```
#include "serial.h"
#include <avr/interrupt.h>
#include <util/delay.h>

#include "timer.h"
#include "hardware.h"
#include "display.h"

int main(){
  displayInitialize();
  hardwareInitialize();
  timerInitialize();
  initializeSerialPort();
  sei();
  while(True){
    hardwareScan();
    if(rateChangedFlag){
      rateChangedFlag = False;
      displayUpdate();
      hardwareFlash();
    }
  }
}
```

### Source-Code D.67: parser.h

```
#ifndef PARSER_H_
#define PARSER_H_

#include "types.h"

extern volatile char command[100];

void parseCommand();

#endif /* PARSER_H_ */
```

### Source-Code D.68: parser.c

```
#include "parser.h"
#include "types.h"
#include "serial.h"
#include "timer.h"
#include <stdio.h>

volatile char command[100];

bool compareCommand(char reference[]){
  uint8_t i = 0;
  while(reference[i]){
    if(command[i] != reference[i]) return false;
    i++;
  }
  return true;
}

void respond(bool result){
  if(result){
    sendString("done");
  }else{
    sendString("failed");
```

```
  }
}

void parseCommand(){
  if(compareCommand("*IDN?")){
    sendString("Gas␣counter␣unit\r\nby␣Thomas.Moeller@uni.kn\r\nFirmware␣2␣(12.06.2017)")
        ;
    return;
  }
  if(compareCommand(":RAT?")){
    char temp[16];
    sprintf(temp,"%09.6f/s",rate);
    sendString(temp);
    return;
  }
  if(compareCommand(":ABS?")){
    char temp[16];
    sprintf(temp,"%08lul",absolute);
    sendString(temp);
    return;
  }
  if(compareCommand(":ABS␣")){
    uint32_t temp;
    int8_t ret = sscanf(&command[5],"%lu",&temp);
    absolute = temp;
    respond(ret>0);
    return;
  }
  sendString("command␣unknown");
}
```

**Source-Code D.69:** pinout.h

```
#ifndef PINOUT_H_
#define PINOUT_H_

#include <avr/io.h>

#define PORT_RX PORTD
#define DDR_RX  DDRD
#define PIN_RX  PIND
#define P_RX  PD0

#define PORT_TX PORTD
#define DDR_TX  DDRD
#define PIN_TX  PIND
#define P_TX  PD1

#define PDetector     PD6
#define PORTDetector    PORTD
#define DDRDetector     DDRD
#define PINDetector     PIND

#define PDetectorPresent  PD2
#define PORTDetectorPresent PORTD
#define DDRDetectorPresent  DDRD
#define PINDetectorPresent  PIND

#define PLEDDetectorPresent   PD3
#define PORTLEDDetectorPresent  PORTD
#define DDRLEDDetectorPresent DDRD
#define PINLEDDetectorPresent PIND

#define PLEDFlash   PD4
#define PORTLEDFlash  PORTD
#define DDRLEDFlash   DDRD
#define PINLEDFlash   PIND

#define PSS     PB4
```

```
#define PORTSS    PORTB
#define DDRSS     DDRB
#define PINSS     PINB

#define PMOSI     PB5
#define PORTMOSI  PORTB
#define DDRMOSI   DDRB
#define PINMOSI   PINB

#define PMISO     PB6
#define PORTMISO  PORTB
#define DDRMISO   DDRB
#define PINMISO   PINB

#define PSCK      PB7
#define PORTSCK   PORTB
#define DDRSCK    DDRB
#define PINSCK    PINB

#define POE       PC1
#define PORTOE    PORTC
#define DDROE     DDRC
#define PINOE     PINC

#define PRCK      PC0
#define PORTRCK   PORTC
#define DDRRCK    DDRC
#define PINRCK    PINC

#define PLEDSeconds    PD7
#define PORTLEDSeconds  PORTD
#define DDRLEDSeconds DDRD
#define PINLEDSeconds PIND

#define PLEDMinutes    PB1
#define PORTLEDMinutes  PORTB
#define DDRLEDMinutes DDRB
#define PINLEDMinutes PINB

#define PLEDHours    PD5
#define PORTLEDHours  PORTD
#define DDRLEDHours   DDRD
#define PINLEDHours   PIND

#endif /* PINOUT_H_ */
```

**Source-Code D.70:** timer.h

```
#ifndef TIMER_H_
#define TIMER_H_

#include "types.h"

extern volatile uint16_t overflows;
extern volatile uint32_t absolute;
extern volatile double rate;
extern volatile bool rateChangedFlag;

void timerInitialize();

#endif /* TIMER_H_ */
```

**Source-Code D.71:** timer.c

```
#include "types.h"
#include "timer.h"
#include "pinout.h"
#include <avr/interrupt.h>
#include <util/delay.h>
```

```
volatile uint16_t overflows;
volatile uint32_t absolute;
volatile double rate;
volatile bool rateChangedFlag;

void timerInitialize(){
  PORTDetector |= (1<<PDetector);   //Activate Detector Pullup
  _delay_ms(1000);          //Wait for cap to charge up
  TIMSK |= (1<<TOIE1);        //Timer1 overflow interrupt enable
  TIMSK |= (1<<TICIE1);       //Timer1 input capture interrupt
  TCCR1B |= (1<<ICNC1);       //Input capture Noise canceler
  TCCR1B |= (1<<CS02) | (1<<CS00);  //Timer1 prescaler 1024
}

double calculateRate(uint32_t tics){
  return F_CPU/1024.0*10.0/tics;
}

ISR(TIMER1_OVF_vect){
  double newRate = calculateRate((uint32_t)overflows<<16);
  if(overflows<0xffff){
    overflows++;
    newRate = calculateRate((uint32_t)overflows<<16);
  }else{
    newRate = 0.0;
  }
  if(newRate<rate){
    rate = newRate;
    rateChangedFlag = True;
  }
}

ISR(TIMER1_CAPT_vect){
  uint32_t temp = ICR1 | ((uint32_t)overflows<<16);
  absolute+=10;
  TCNT1 -= ICR1;
  overflows = 0;
  rate = calculateRate(temp);
  rateChangedFlag = True;
}
```

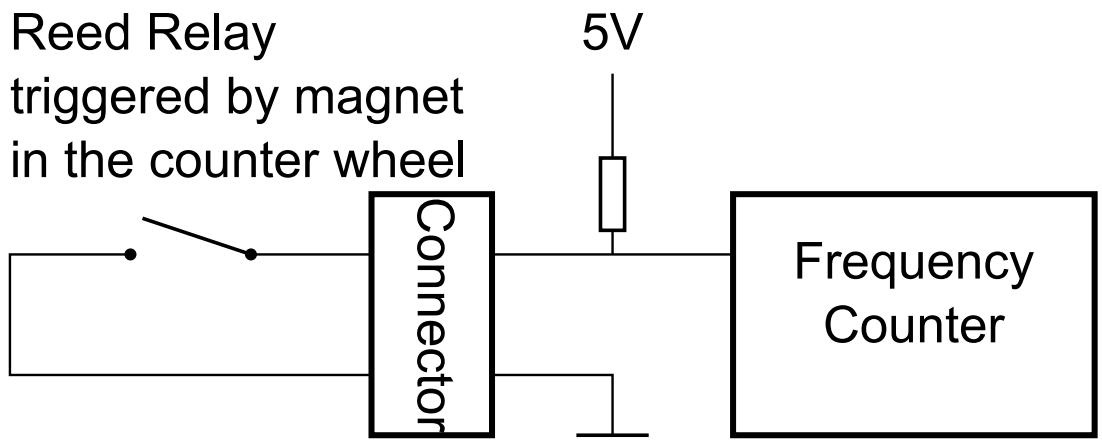



**Figure D.56.:** Simplified block diagram of the gascounter NIM insert

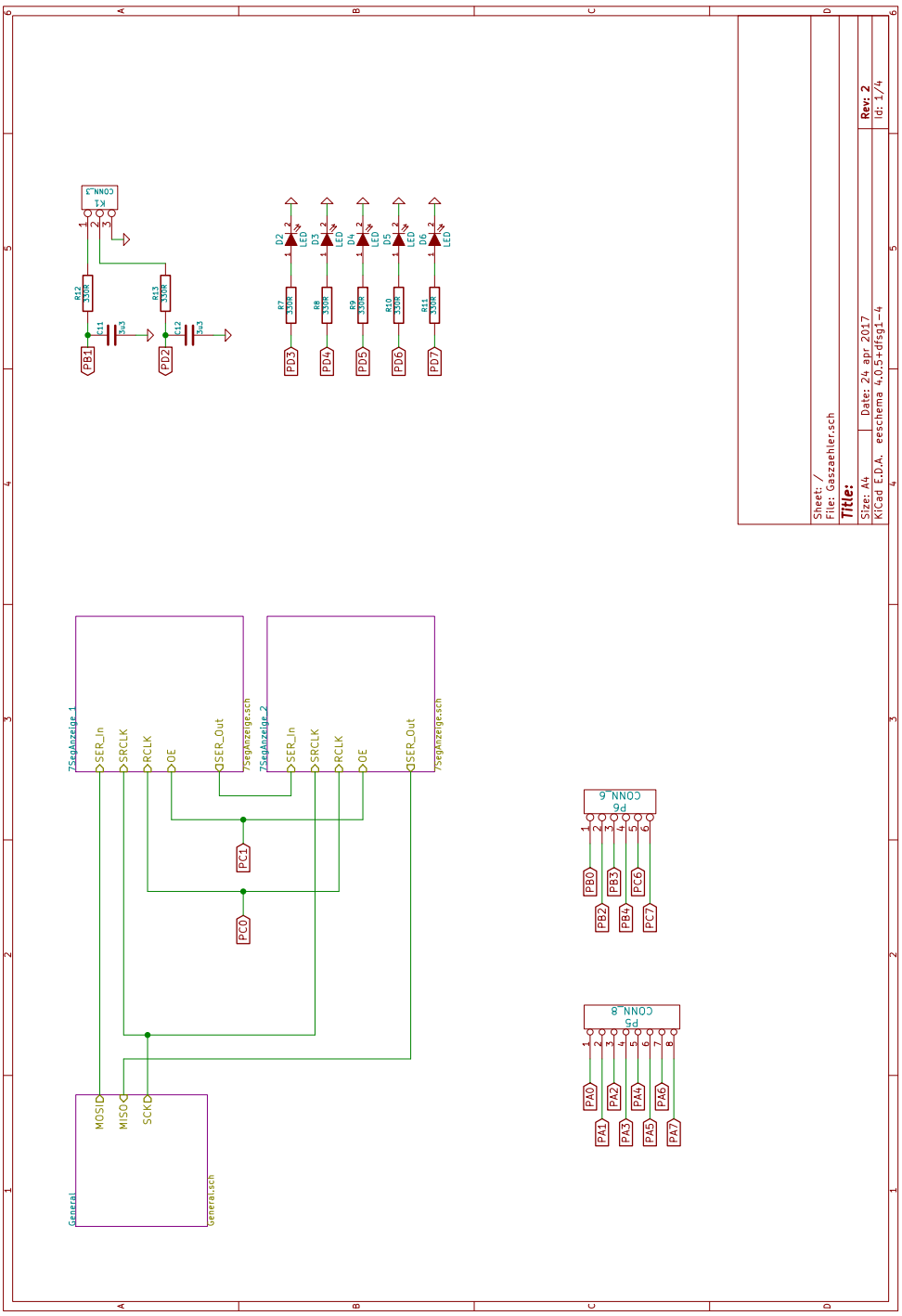
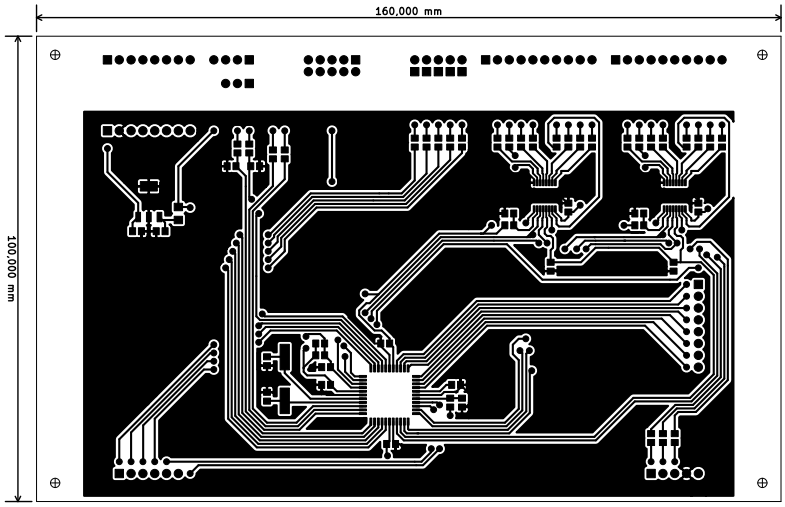
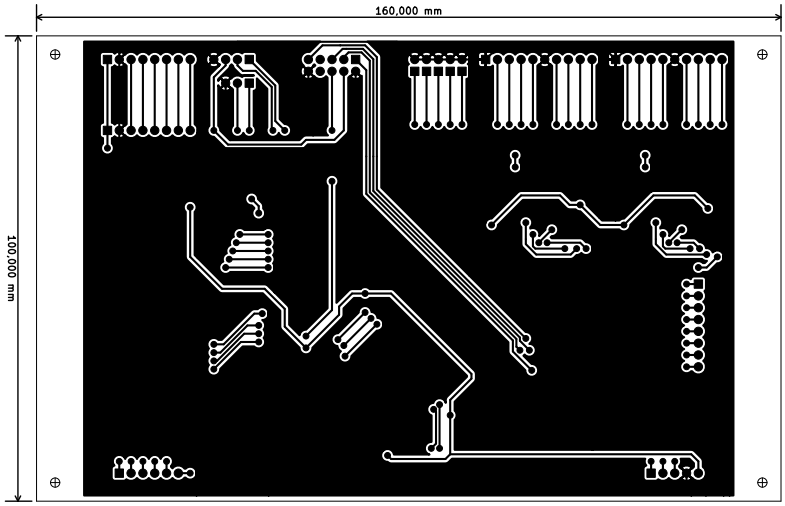


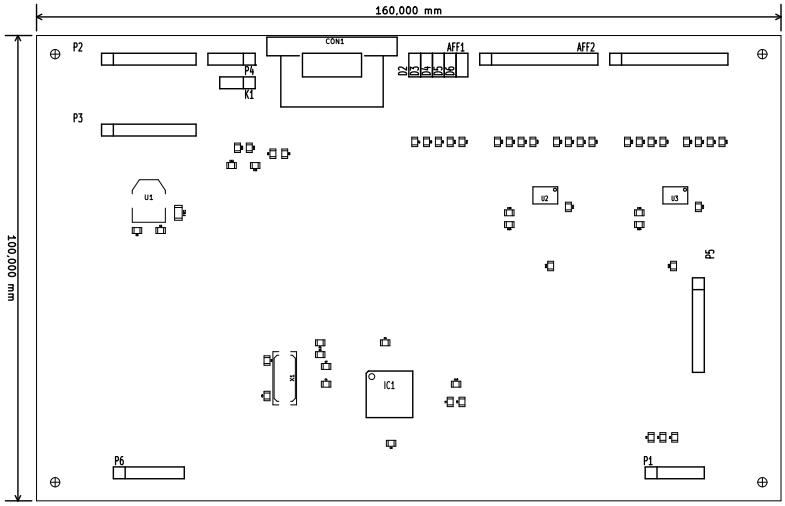

**Figure D.57.:** Schematic of the gascounter NIM insert




**(a)** Top side copper structure

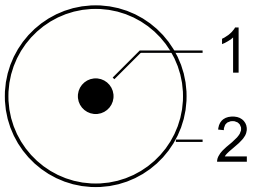



**(b)** Bottom side copper structure




**(c)** Component references

**Figure D.58.:** Circuit board layout of the gascounter NIM insert




| | | |
|---|---|---|
| 1 | Connection to switch | input |
| 2 | Ground | output |

**Figure D.59.:** Pin configuration of the gascounter NIM insert


**(a)** front **(b)** back

**Figure D.60.:** Photos of the outside of the gascounter NIM insert

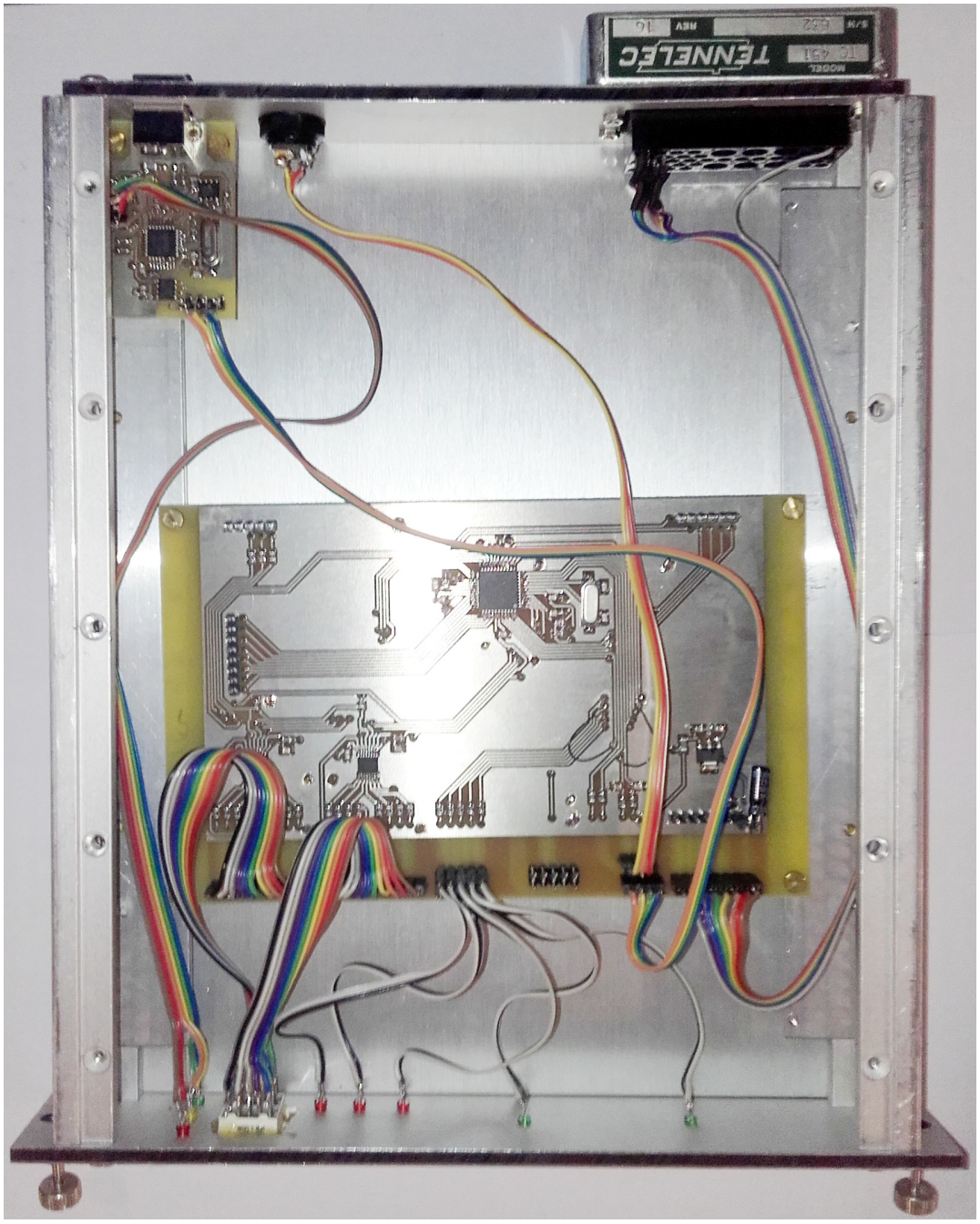


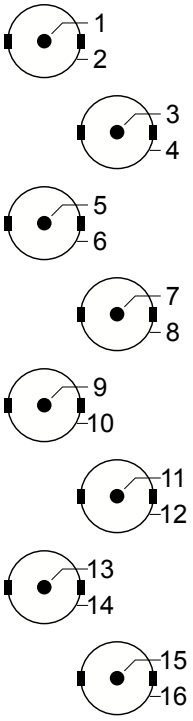

**Figure D.61.:** Photo of the inside of the gascounter NIM insert

## D.9. Sense

The sense NIM insert adapts the eight wires coming from the sense IceCube insert and converts the to BNC connectors. This makes it easy to flexibly attach these signal to different devices, e.g. the transient recorder, the multimeter, or the lock-in amplifier.
The pinout of the connectors is shown in figure D.62 and photos of the front and inside are shown in figure D.63.


| | | |
|---|---|---|
| 1 | (2−) − (2+) reference | output |
| 2 | (3−) − (3+) reference | output |
| 3 | (4−) − (4+) signal | output |
| 4 | (4−) − (4+) reference | output |
| 5 | (1−) − (1+) signal | output |
| 6 | (1−) − (1+) reference | output |
| 7 | (2−) − (2+) signal | output |
| 8 | (3−) − (3+) signal | output |

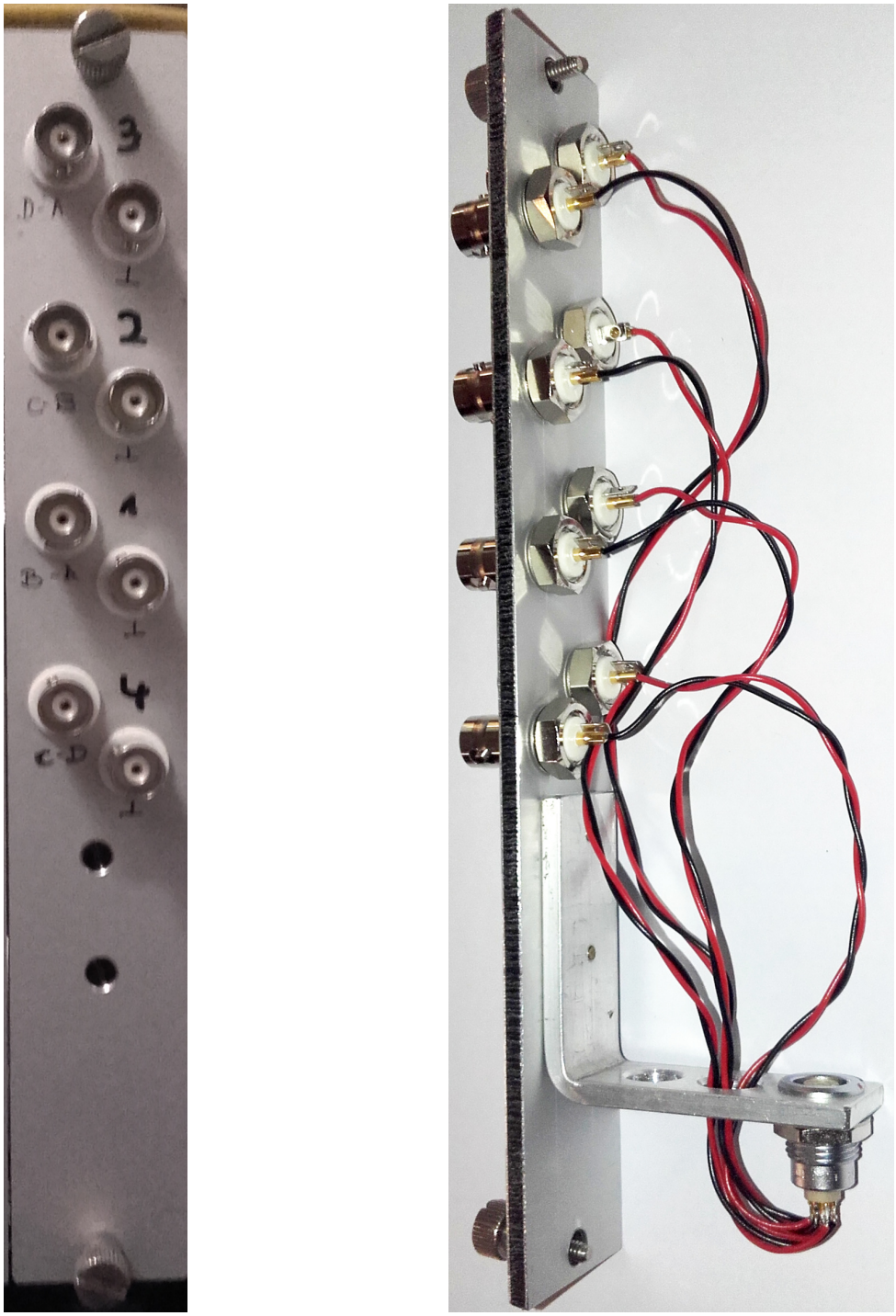
**(a)** back




| | | |
|---|---|---|
| 1 | (3−) − (3+) reference | output |
| 2 | not connected | |
| 3 | (3−) − (3+) reference | output |
| 4 | not connected | |
| 5 | (2−) − (2+) signal | output |
| 6 | not connected | |
| 7 | (2−) − (2+) reference | output |
| 8 | not connected | |
| 9 | (1−) − (1+) signal | output |
| 10 | not connected | |
| 11 | (1−) − (1+) reference | output |
| 12 | not connected | |
| 13 | (4−) − (4+) signal | output |
| 14 | not connected | |
| 15 | (4−) − (4+) signal | output |
| 16 | not connected | |

**(b)** front

**Figure D.62.:** Pin configuration of the sense NIM insert




**(a)** front **(b)** inside

**Figure D.63.:** Photos of the sense NIM insert

## D.10. Camera

The purpose of the camera insert is to have an automated way of taking pictures of the sample. The camera has a three pin 2.5mm phone connector. These connections are shorted to ground via an opto-isolated interface.
The insert has two yellow LEDs on the front indicating the two steps of the shutter. One greed LED indicates that the command to take a picture is being executed. This command can either be started by the switch on the front panel, or by a command send over the USB port. The microcontroller supports the following commands.

- `*IDN?`
  This command returns the name of the insert, the author, the version, and date of the firmware. The output is three lines long.

- `:PICture`
  This command starts the procedure to take a picture.

- `:FOCus:ON`
  This command shorts the focus signal of the camera.

- `:FOCus:OFF`
  This command opens the focus signal of the camera.

- `:FOCus?`
  This command returns the state of the focus signal. The response is either `ON` or `OFF`.

- `:EXPosure:ON`
  This command shorts the exposure signal of the camera.

- `:EXPosure:OFF`
  This command opens the exposure signal of the camera.

- `:EXPosure?`
  This command returns the state of the exposure signal. The response is either `ON` or `OFF`.

A block diagram of the module is shown in figure D.64. The full schematic of the circuit board is shown in figure D.65 and the corresponding board layout in figure D.66. The pinout of the connectors is shown in figure D.67 and photos of the front, back, and inside are shown in figures D.68 and D.69. The firmware consists of the following files.

**Source-Code D.72:** hardware.h

```
#ifndef HARDWARE_H_
#define HARDWARE_H_

void hardwareInitialization();

#endif /* HARDWARE_H_ */
```

**Source-Code D.73:** hardware.c

```
#include "switch.h"
#include "pinout.h"
#include "hardware.h"

void hardwareInitialization(){
  switchInitialization();
  DDR_LED_Green |= (1<<P_LED_Green);
  DDR_LED_Foc |= (1<<P_LED_Foc);
  DDR_LED_Exp |= (1<<P_LED_Exp);
  DDR_OPT_Foc |= (1<<P_OPT_Foc);
  DDR_OPT_Exp |= (1<<P_OPT_Exp);
}
```

**Source-Code D.74:** main.h

```
#ifndef MAIN_H_
#define MAIN_H_

#include "types.h"

extern bool requestPicture;

bool startExposure();
bool stopExposure();
bool getExposure();
bool startFocus();
bool stopFocus();
bool getFocus();

#endif /* MAIN_H_ */
```

**Source-Code D.75:** main.c

```
#include "serial.h"
#include <avr/interrupt.h>
#include "parser.h"
#include "types.h"
#include <util/delay.h>
#include "main.h"

#include "pinout.h"
#include "switch.h"
#include "hardware.h"

bool requestPicture = false;
bool hardwarePicture = false;

bool startExposure(){
  startFocus();
  _delay_ms(100);
  PORT_OPT_Exp |= (1<<P_OPT_Exp);
  PORT_LED_Exp |= (1<<P_LED_Exp);
  return true;
}
bool stopExposure(){
  PORT_LED_Exp &= ~(1<<P_LED_Exp);
  PORT_OPT_Exp &= ~(1<<P_OPT_Exp);
  stopFocus();
  return true;
}
bool getExposure(){
  if(PORT_OPT_Exp & (1<<P_OPT_Exp)){
    return true;
  }else{
    return false;
  }
}
```

```
bool startFocus(){
  if(getExposure()) return false;
  PORT_OPT_Foc |= (1<<P_OPT_Foc);
  PORT_LED_Foc |= (1<<P_LED_Foc);
  return true;
}
bool stopFocus(){
  if(getExposure()) return false;
  PORT_OPT_Foc &= ~(1<<P_OPT_Foc);
  PORT_LED_Foc &= ~(1<<P_LED_Foc);
  return true;
}
bool getFocus(){
  if(PORT_OPT_Foc & (1<<P_OPT_Foc)){
    return true;
  }else{
    return false;
  }
}
void takePicture(){
  PORT_LED_Green |= (1<<P_LED_Green);
  startExposure();
  _delay_ms(100);
  stopExposure();
  _delay_ms(400);
  PORT_LED_Green &= ~(1<<P_LED_Green);
}

int main(){
  hardwareInitialization();
  initializeSerialPort();
  sei();
  while(1){
    if(requestPicture){
      takePicture();
      sendString("done");
      requestPicture = false;
    }
    switchUpdate();
    hardwarePicture = tasterFlag; //Read flag
    tasterFlag = False; //Clear flag
    if(hardwarePicture){
      takePicture();
      hardwarePicture = false;
    }
  }
}
```

**Source-Code D.76:** parser.h

```
#ifndef PARSER_H_
#define PARSER_H_

#include "types.h"

extern volatile char command[100];

void parseCommand();

#endif /* PARSER_H_ */
```

**Source-Code D.77:** parser.c

```
#include "parser.h"
#include "types.h"
#include "serial.h"
#include "main.h"

volatile char command[100];
```

```
bool compareCommand(char reference[]){
  uint8_t i = 0;
  while(reference[i]){
    if(command[i] != reference[i]) return false;
    i++;
  }
  return true;
}

void respond(bool result){
  if(result){
    sendString("done");
  }else{
    sendString("failed");
  }
}

void parseCommand(){
  if(compareCommand("*IDN?")){
    sendString("Camera␣triggering␣unit\r\nby␣Thomas.Moeller@uni.kn\r\nFirmware␣2␣
        (22.08.2019)");
    return;
  }
  if(compareCommand(":EXP␣ON")){
    respond(startExposure());
    return;
  }
  if(compareCommand(":EXP␣OFF")){
    respond(stopExposure());
    return;
  }
  if(compareCommand(":EXP?")){
    if(getExposure()){
      sendString("ON");
    }else{
      sendString("OFF");
    }
    return;
  }
  if(compareCommand(":FOC␣ON")){
    respond(startFocus());
    return;
  }
  if(compareCommand(":FOC␣OFF")){
    respond(stopFocus());
    return;
  }
  if(compareCommand(":FOC?")){
    if(getFocus()){
      sendString("ON");
    }else{
      sendString("OFF");
    }
    return;
  }
  if(compareCommand(":PIC")){
    if(requestPicture){
      sendString("busy");
      return;
    }
    requestPicture = true;
    return;
  }
  sendString("command␣unknown");
}
```

**Source-Code D.78:** pinout.h

```
#ifndef PINOUT_H_
```

```
#define PINOUT_H_

#include <avr/io.h>

#define PORT_RX PORTD
#define DDR_RX  DDRD
#define PIN_RX  PIND
#define P_RX  PD0

#define PORT_TX PORTD
#define DDR_TX  DDRD
#define PIN_TX  PIND
#define P_TX  PD1


#define PORT_LED_Green  PORTB
#define DDR_LED_Green DDRB
#define PIN_LED_Green PINB
#define P_LED_Green   PB0

#define PORT_LED_Exp  PORTB
#define DDR_LED_Exp   DDRB
#define PIN_LED_Exp   PINB
#define P_LED_Exp   PB1

#define PORT_LED_Foc  PORTB
#define DDR_LED_Foc   DDRB
#define PIN_LED_Foc   PINB
#define P_LED_Foc   PB2

#define PORT_OPT_Exp  PORTB
#define DDR_OPT_Exp   DDRB
#define PIN_OPT_Exp   PINB
#define P_OPT_Exp   PB3

#define PORT_OPT_Foc  PORTB
#define DDR_OPT_Foc   DDRB
#define PIN_OPT_Foc   PINB
#define P_OPT_Foc   PB4


#define PORT_Switch   PORTA
#define DDR_Switch    DDRA
#define PIN_Switch    PINA
#define P_Switch    PA0

#endif /* PINOUT_H_ */
```

**Source-Code D.79:** switch.h

```
/*
 * taster.h
 *
 *  Created on: 07.03.2017
 *      Author: thomas
 */

#ifndef SWITCH_H_
#define SWITCH_H_

#include "types.h"

extern volatile bool tasterFlag;

void switchInitialization();
void switchUpdate();

#endif /* SWITCH_H_ */
```

**Source-Code D.80:** switch.c

```
/*
 * taster.c
 *
 *  Created on: 07.03.2017
 *      Author: thomas
 */

#include "switch.h"

#include "types.h"
#include <util/delay.h>
#include "pinout.h"

volatile bool tasterFlag = False;
volatile bool tasterAlt = False;

void switchInitialization(){
  //activate Pull Up
  PORT_Switch |= (1<<P_Switch);
  _delay_ms(500);
}

void switchUpdate(){
  if (PIN_Switch & (1<<P_Switch)){
    tasterAlt = False;
  }else{
    if(tasterAlt) return;
    tasterFlag = True;
    tasterAlt = True;
  }
}
```

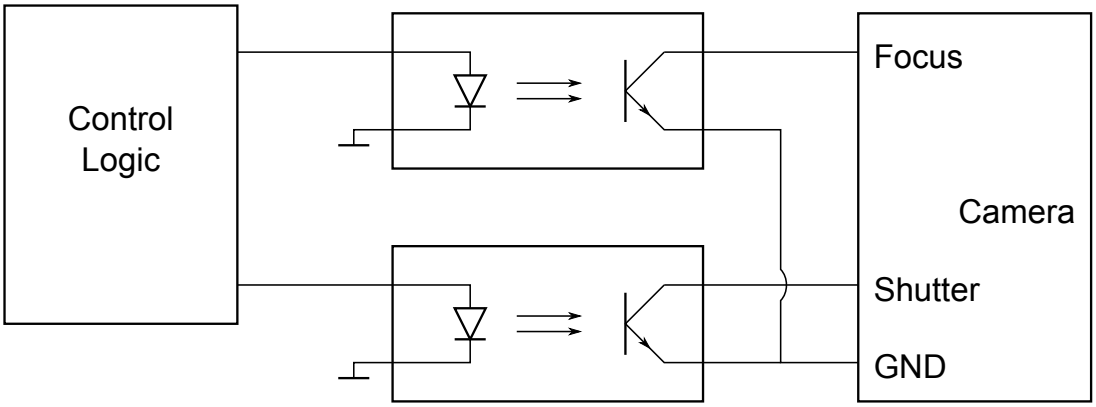



**Figure D.64.:** Simplified block diagram of the camera NIM insert

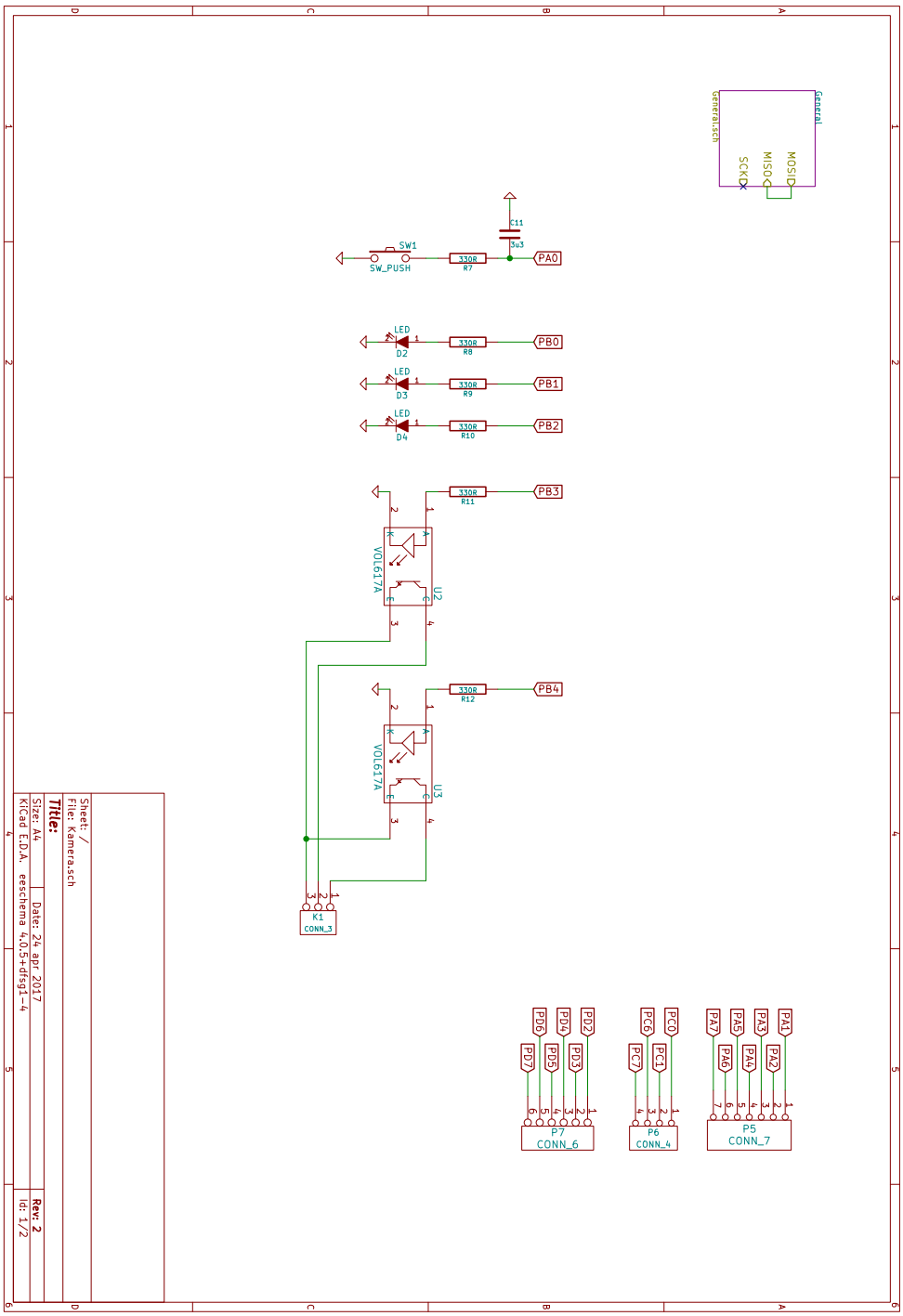
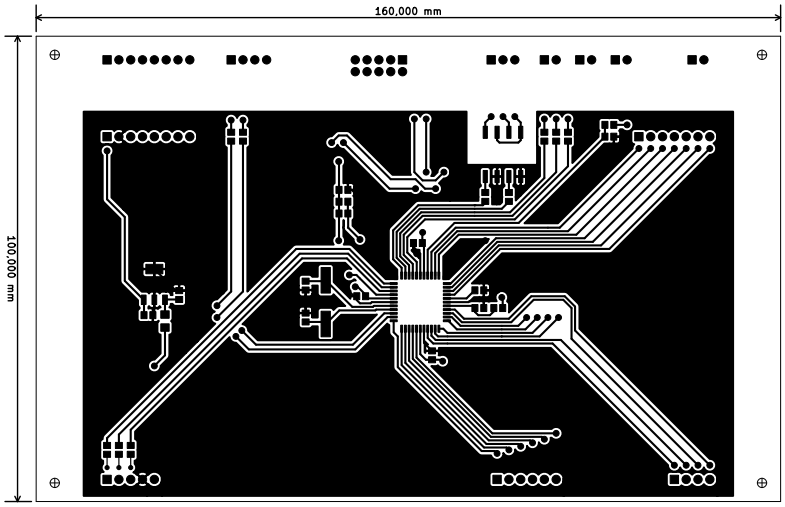
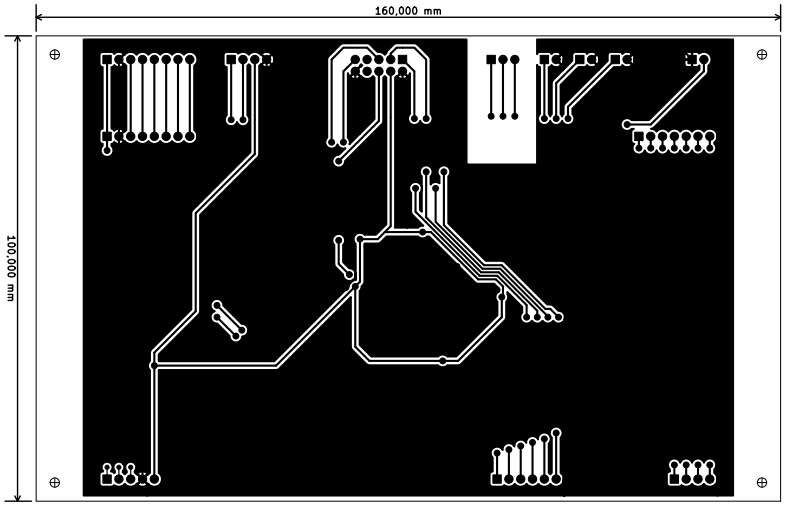


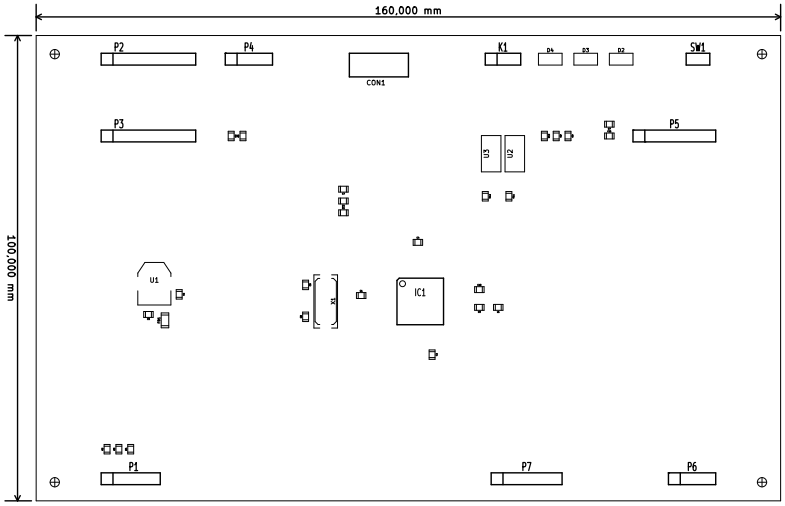

**Figure D.65.:** Schematic of the power camera NIM insert




**(a)** Top side copper structure

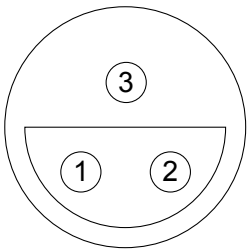



**(b)** Bottom side copper structure




**(c)** Component references

**Figure D.66.:** Circuit board layout of the camera NIM insert




| | | |
|---|---|---|
| 1 | focus | output |
| 2 | shutter | output |
| 3 | ground | output |

**Figure D.67.:** Pin configuration of the camera NIM insert


**(a)** front **(b)** back

**Figure D.68.:** Photos of the outside of the camera NIM insert

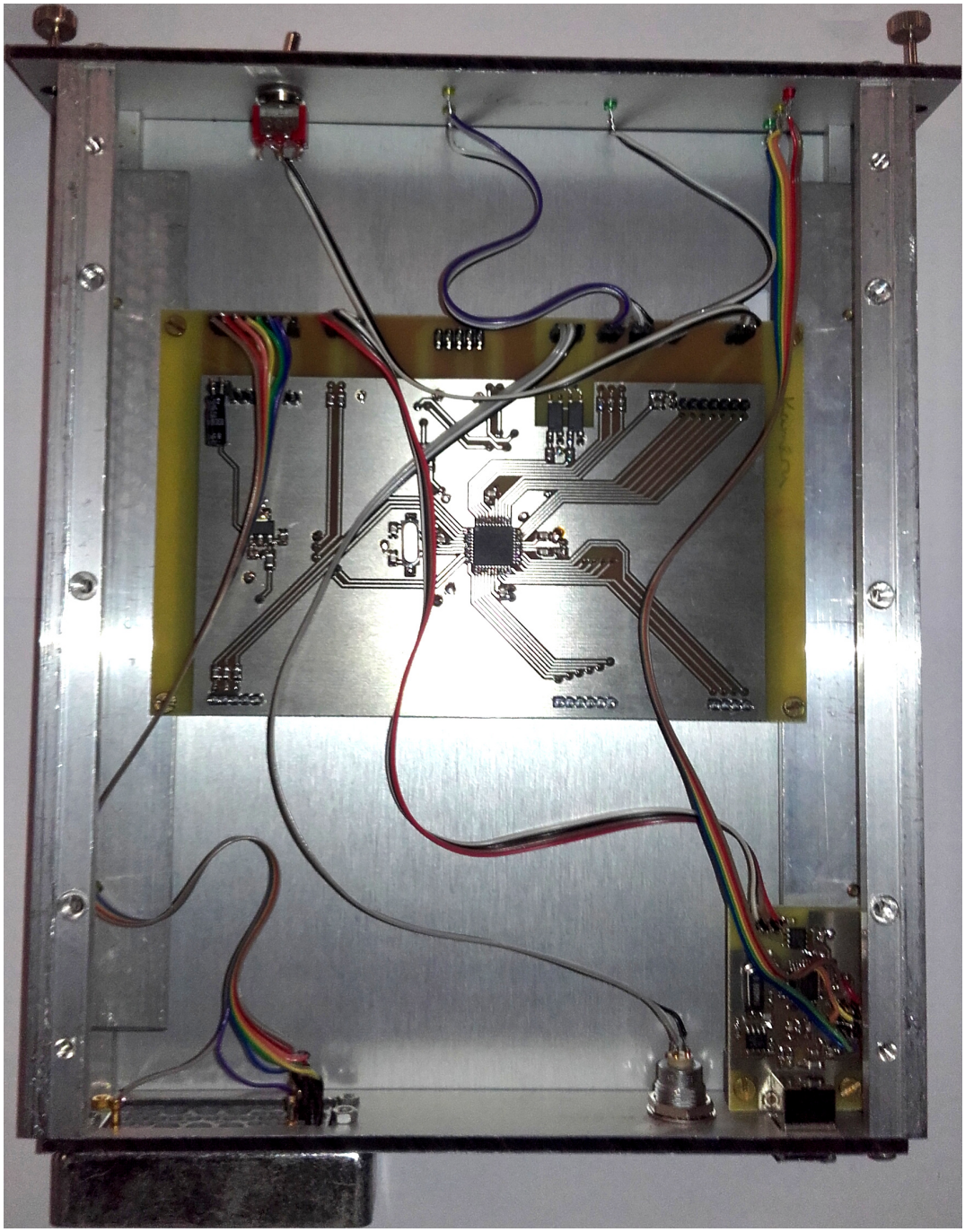


**Figure D.69.:** Photo of the inside of the camera NIM insert

## D.11. Power surveillance

The purpose of the power surveillance insert is to measure the power supplies generated by the NIM crate and to monitor the presence of external power. The latter is interesting if the crate is powered by a UPS.
The insert has two LEDs on the front. One indicates that the internal supplies are within the tolerances, and another indicates the presence of the external power.
To detect the external power, a plugpack power supply is used to drive an optocoupler. Thus it generates no ground loops.
A USB port allows to connect to the microcontroller to read out the individual voltages of the rails as well as the status of the external power. The microcontroller supports the following commands.

- `*IDN?`
  This command returns the name of the insert, the author, the version, and date of the firmware. The output is three lines long.

- `:EXTernal?`
  This command returns `present` if there is external power or `failed` if not.

- `:VOL:06?`
  This command returns the measured voltage of the positive 6 V rail in the format `%06.3fV` in V.

- `:VOL:12?`
  This command returns the measured voltage of the positive 12 V rail in the format `%06.3fV` in V.

- `:VOL:24?`
  This command returns the measured voltage of the positive 24 V rail in the format `%06.3fV` in V.

A block diagram of the module is shown in figure D.70. The full schematic of the circuit board is shown in figure D.71 and the corresponding board layout in figure D.72. The pinout of the connectors is shown in figure D.73 and photos of the front, back, and inside are shown in figures D.74 and D.75. The firmware consists of the following files.

**Source-Code D.81:** adc.h

```
#ifndef ADC_H_
#define ADC_H_

#include "types.h"

extern volatile double voltage06;
extern volatile double voltage12;
extern volatile double voltage24;

void adcInitialize();

#endif /* ADC_H_ */
```

**Source-Code D.82:** adc.c

```
#include "adc.h"
#include "pinout.h"
#include <avr/interrupt.h>

volatile double voltage06;
volatile double voltage12;
volatile double voltage24;

#define Vref 2.642

void adcInitialize(){
  ADMUX |= (1<<REFS1) | (1<<REFS0); //2.56V Referece measured to be 2.642V
  ADMUX &= ~((1<<MUX4) | (1<<MUX3) | (1<<MUX2) | (1<<MUX1) | (1<<MUX0));
  ADMUX |= (PIntern06<<MUX0); //set input to BaseSense
  ADCSRA |= (1<<ADPS2) | (1<<ADPS0); //ADC Freqency = 3686400Hz / 32 = 115200Hz
  ADCSRA |= (1<<ADEN); //ADC enable
  ADCSRA |= (1<<ADIE); //ADC Interrupt enable
  ADCSRA |= (1<<ADSC); //ADC start
}

ISR(ADC_vect){
  uint16_t temp = ADC;
  if((ADMUX & ((1<<MUX4) | (1<<MUX3) | (1<<MUX2) | (1<<MUX1) | (1<<MUX0))) == (PIntern06
      <<MUX0)){
    voltage06 = Vref*temp/1024*3;
    ADMUX &= ~((1<<MUX4) | (1<<MUX3) | (1<<MUX2) | (1<<MUX1) | (1<<MUX0));
    ADMUX |= (PIntern12<<MUX0); //Set input to 12V
  }else if((ADMUX & ((1<<MUX4) | (1<<MUX3) | (1<<MUX2) | (1<<MUX1) | (1<<MUX0))) == (
      PIntern12<<MUX0)){
    voltage12 = Vref*temp/1024*6;
    ADMUX &= ~((1<<MUX4) | (1<<MUX3) | (1<<MUX2) | (1<<MUX1) | (1<<MUX0));
    ADMUX |= (PIntern24<<MUX0); //Set input to 24V
  }else{
    voltage24 = Vref*temp/1024*12;
    ADMUX &= ~((1<<MUX4) | (1<<MUX3) | (1<<MUX2) | (1<<MUX1) | (1<<MUX0));
    ADMUX |= (PIntern06<<MUX0); //Set input to 06V
  }
  ADCSRA |= (1<<ADSC); //start ADC
}
```

**Source-Code D.83:** hardware.h

```
#ifndef HARDWARE_H_
#define HARDWARE_H_

#include "types.h"

extern volatile bool externVoltage;

void hardwareInitialize();
void hardwareUpdate();

#endif /* HARDWARE_H_ */
```

**Source-Code D.84:** hardware.c

```
#include "hardware.h"
#include "pinout.h"
#include "adc.h"
#include "types.h"

volatile bool externVoltage;

void hardwareInitialize(){
  DDRLEDIntern |= (1<<PLEDIntern);
  DDRLEDExtern |= (1<<PLEDExtern);
  PORTExtern |= (1<<PExtern);
```

```
}

void hardwareUpdate(){
  if(~PINExtern & (1<<PExtern)){
    externVoltage = True;
    PORTLEDExtern |= (1<<PLEDExtern);
  }else{
    externVoltage = False;
    PORTLEDExtern &= ~(1<<PLEDExtern);
  }
  bool temp = True;
  if(voltage06 > 6.0+dev06) temp = False;
  if(voltage06 < 6.0-dev06) temp = False;
  if(voltage12 > 12.0+dev12) temp = False;
  if(voltage12 < 12.0-dev12) temp = False;
  if(voltage24 > 24.0+dev24) temp = False;
  if(voltage24 < 24.0-dev24) temp = False;
  if(temp){
    PORTLEDIntern |= (1<<PLEDIntern);
  }else{
    PORTLEDIntern &= ~(1<<PLEDIntern);
  }
}
```

### Source-Code D.85: main.c

```
#include "serial.h"
#include <avr/interrupt.h>
#include "types.h"
#include <util/delay.h>

#include "adc.h"
#include "hardware.h"

int main(){
  initializeSerialPort();
  adcInitialize();
  hardwareInitialize();
  sei();
  while(True){
    hardwareUpdate();
  }
}
```

### Source-Code D.86: parser.h

```
#ifndef PARSER_H_
#define PARSER_H_

#include "types.h"

extern volatile char command[100];

void parseCommand();

#endif /* PARSER_H_ */
```

### Source-Code D.87: parser.c

```
#include "parser.h"
#include "types.h"
#include "serial.h"
#include <stdio.h>
#include "adc.h"
#include "hardware.h"

volatile char command[100];
```

```
bool compareCommand(char reference[]){
  uint8_t i = 0;
  while(reference[i]){
    if(command[i] != reference[i]) return false;
    i++;
  }
  return true;
}

void respond(bool result){
  if(result){
    sendString("done");
  }else{
    sendString("failed");
  }
}

void parseCommand(){
  if(compareCommand("*IDN?")){
    sendString("Power␣monitoring␣unit\r\nby␣Thomas.Moeller@uni.kn\r\nFirmware␣1␣
        (25.05.2017)");
    return;
  }
  if(compareCommand(":EXT?")){
    if(externVoltage){
      sendString("present");
    }else{
      sendString("failed");
    }
    return;
  }
  if(compareCommand(":VOL:06?")){
    char temp[10];
    sprintf(temp,"%06.3fV",voltage06);
    sendString(temp);
    return;
  }
  if(compareCommand(":VOL:12?")){
    char temp[10];
    sprintf(temp,"%06.3fV",voltage12);
    sendString(temp);
    return;
  }
  if(compareCommand(":VOL:24?")){
    char temp[10];
    sprintf(temp,"%06.3fV",voltage24);
    sendString(temp);
    return;
  }
  sendString("command␣unknown");
}
```

**Source-Code D.88:** pinout.h

```
#ifndef PINOUT_H_
#define PINOUT_H_

#include <avr/io.h>

#define PORT_RX PORTD
#define DDR_RX  DDRD
#define PIN_RX  PIND
#define P_RX  PD0

#define PORT_TX PORTD
#define DDR_TX  DDRD
#define PIN_TX  PIND
#define P_TX  PD1
```

```
#define PLEDIntern     PC0
#define PORTLEDIntern PORTC
#define DDRLEDIntern   DDRC
#define PINLEDIntern   PINC

#define PLEDExtern     PC1
#define PORTLEDExtern PORTC
#define DDRLEDExtern   DDRC
#define PINLEDExtern   PINC

#define PIntern06    PA0
#define PORTIntern06  PORTA
#define DDRIntern06    DDRA
#define PINIntern06    PINA

#define PIntern12    PA1
#define PORTIntern12  PORTA
#define DDRIntern12    DDRA
#define PINIntern12    PINA

#define PIntern24    PA2
#define PORTIntern24  PORTA
#define DDRIntern24    DDRA
#define PINIntern24    PINA

#define PExtern      PA3
#define PORTExtern     PORTA
#define DDRExtern    DDRA
#define PINExtern    PINA

#define dev06 0.2
#define dev12 0.2
#define dev24 0.2

#endif /* PINOUT_H_ */
```

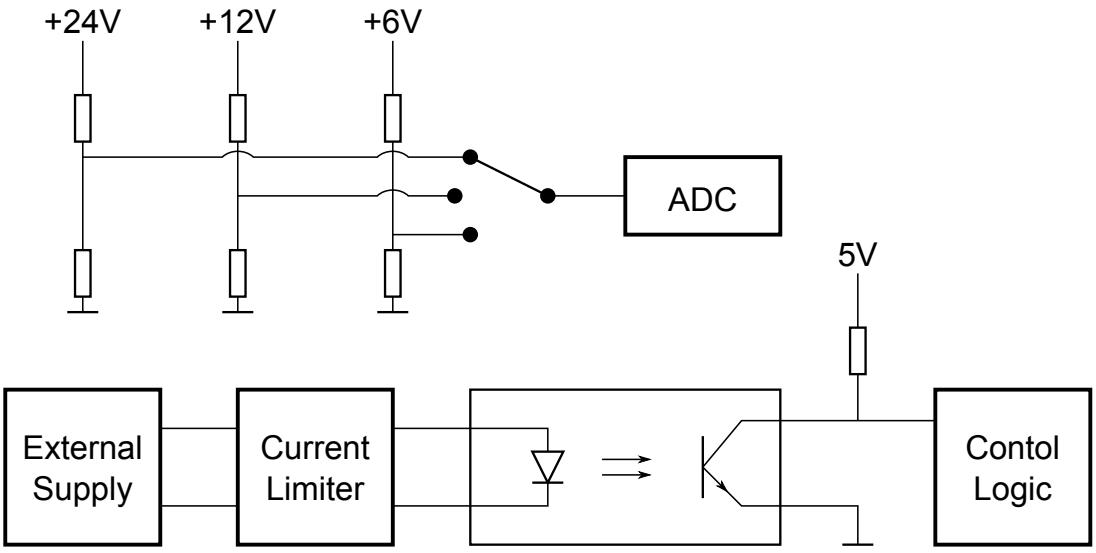



**Figure D.70.:** Simplified block diagram of the power surveillance NIM insert

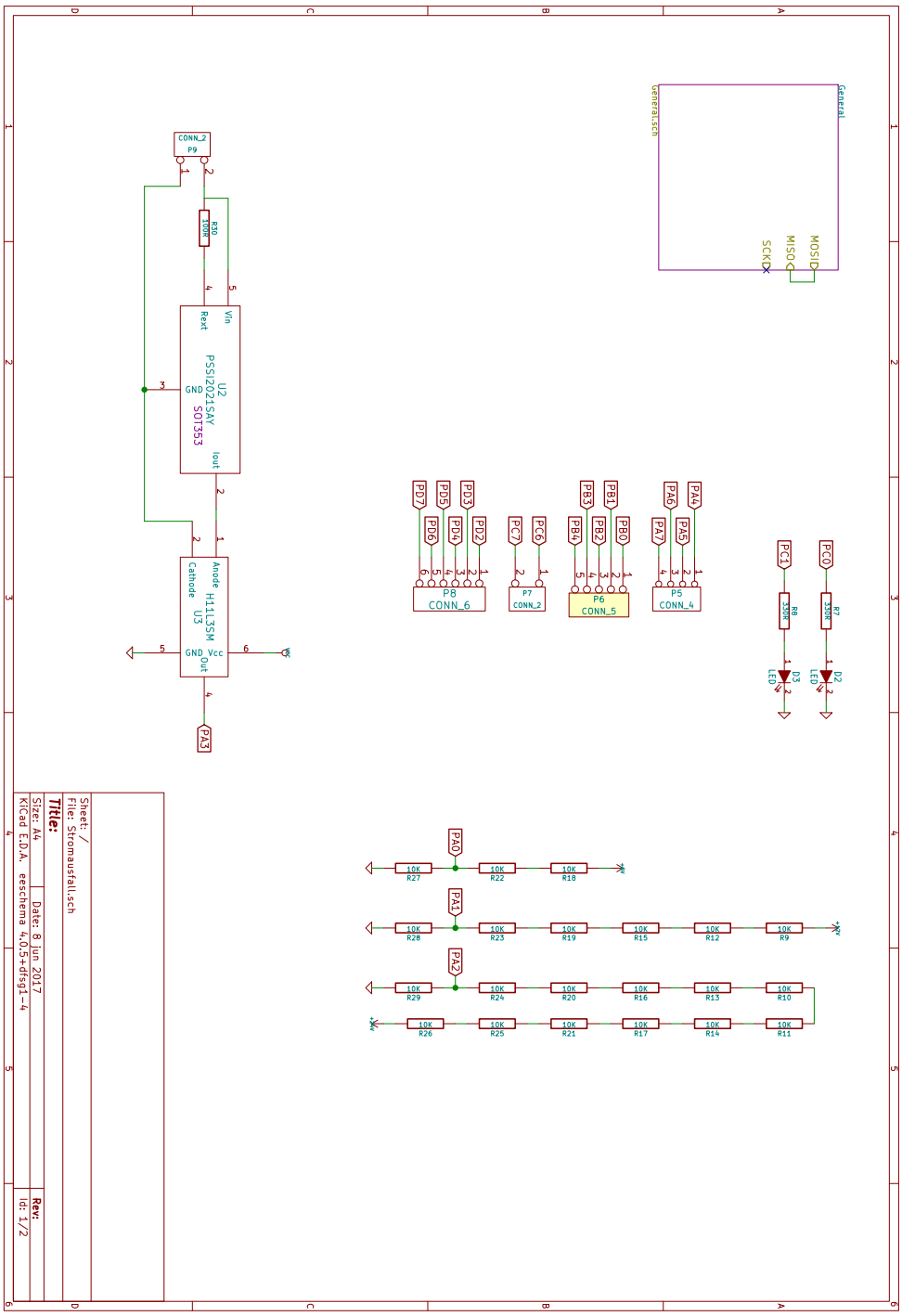
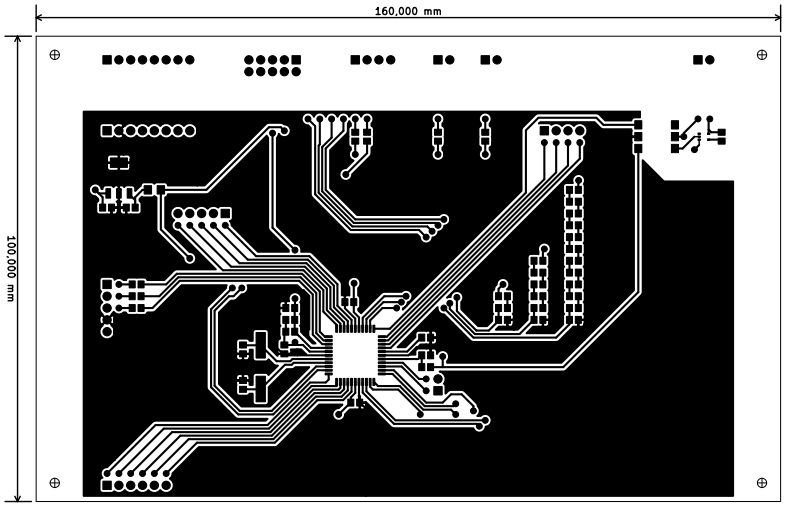
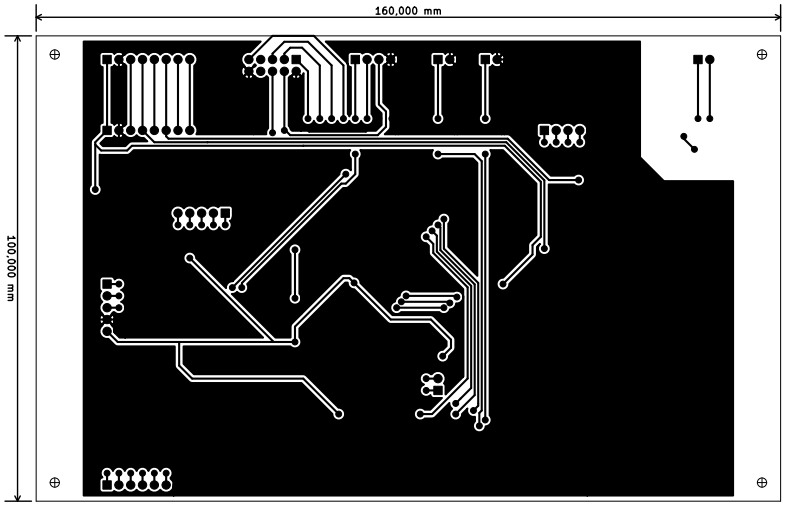


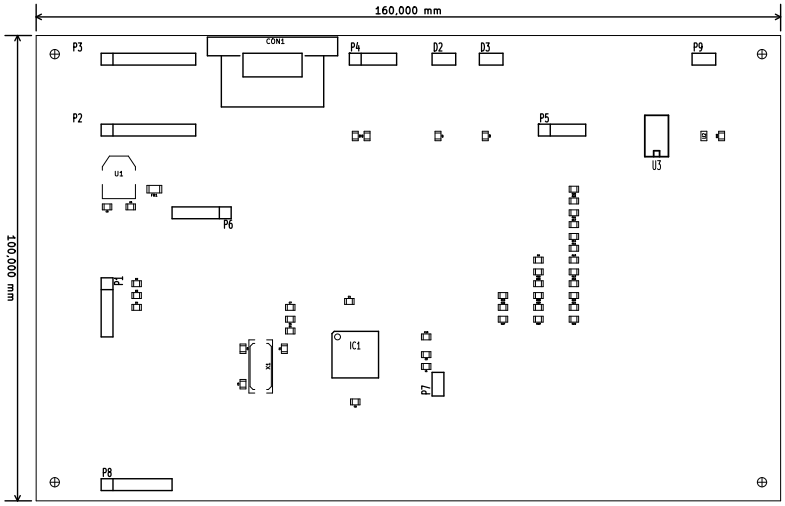

**Figure D.71.:** Schematic of the power surveillance NIM insert




**(a)** Top side copper structure

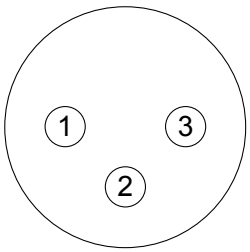



**(b)** Bottom side copper structure



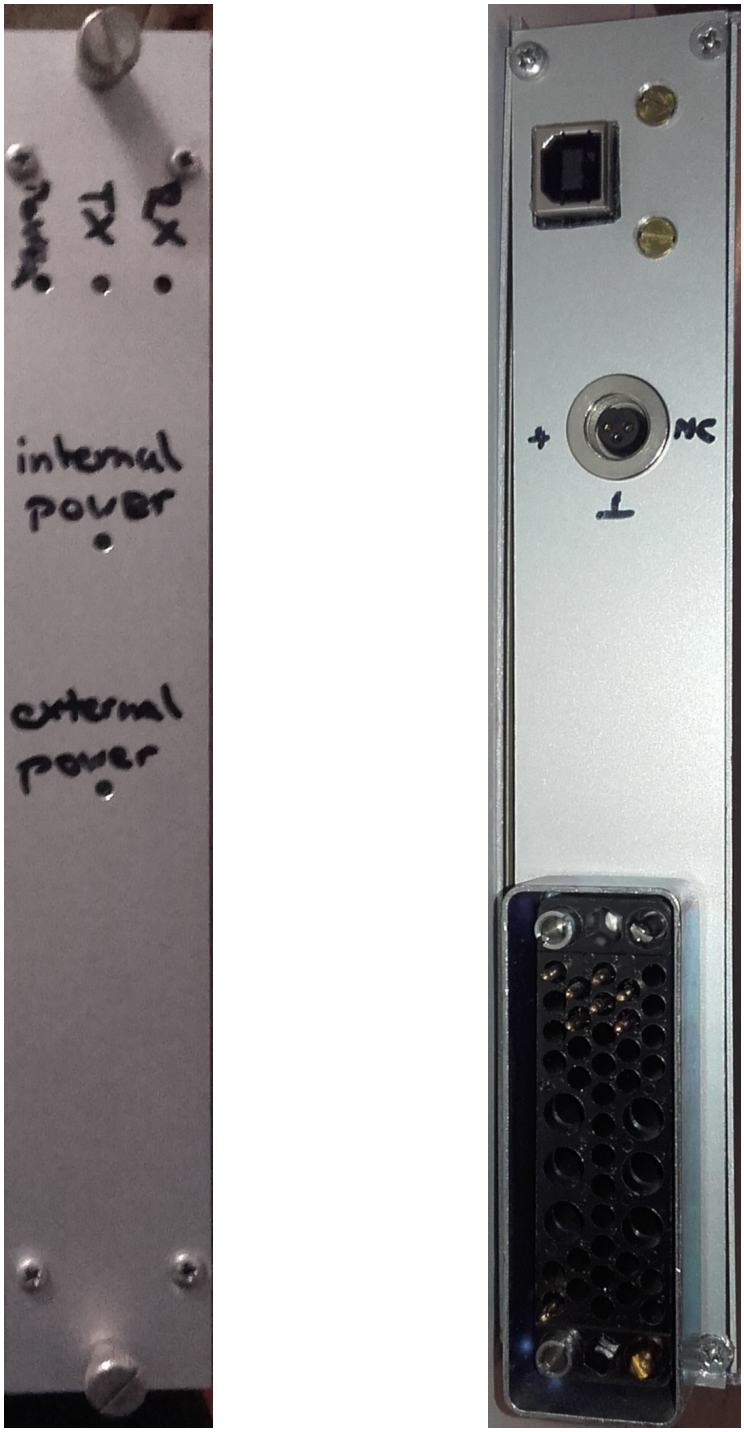

**(c)** Component references

**Figure D.72.:** Circuit board layout of the power surveillance NIM insert




| | | |
|---|---|---|
| 1 | positive supply | input |
| 2 | ground | input |
| 3 | not connected | |

**Figure D.73.:** Pin configuration of the power surveillance NIM insert




**(a)** front **(b)** back

**Figure D.74.:** Photos of the outside of the power surveillance NIM insert

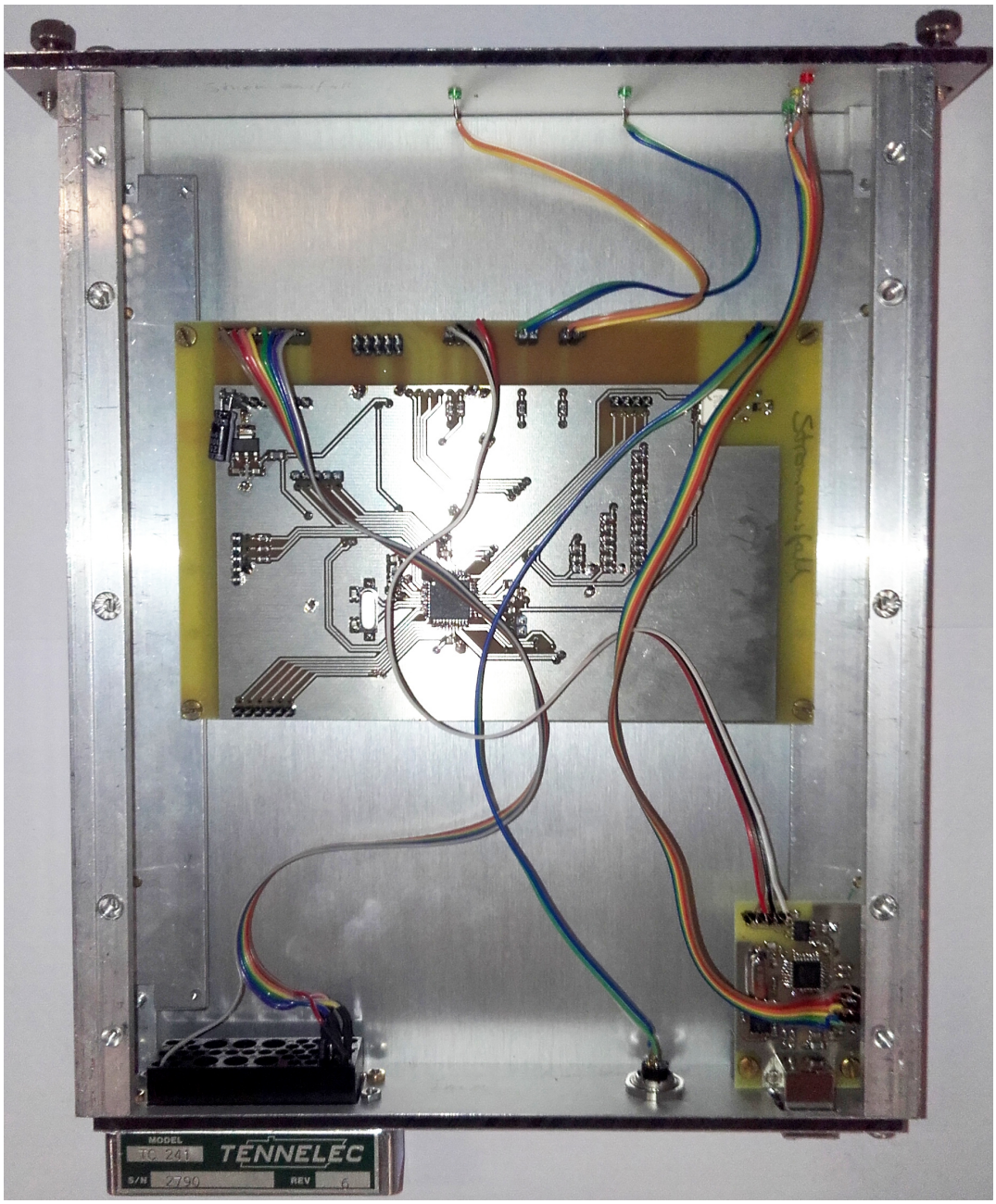



**Figure D.75.:** Photo of the inside of the power surveillance NIM insert

# E. Software reference

The software for running the measurements and controlling the setup is rather comprehensive. Due to a network based software design the complete system uses several programs depending on each other. The benefit of this approach is that if a program crashes, the rest of the functionality of the setup is unaffected. The crashed program can then be restarted and even changed without the need to interrupt the rest. This helps in the debugging process.
Another benefit of the network based approach is that the programs can run on different hardware. In my setup there are three computers: the main laptop at the front running Windows 7 has the IP-address 134.34.142.60, the laptop in the back running Linux has the IP-address 134.34.142.62, and the transient recorder running Windows 7 has the IP-address 134.34.142.61 which is running programs directly or in a virtual machine running Linux. In the following, the computers are referred to by their IP-address.
The programs are running on python version 3 or python version 2.7, since the version 2.7 seemed to by more stable for certain applications.
In the following documentation every python program is described. They are sorted alphabetically and labeled with their folder name.
The description contains the following information (if applicable):

- Server functionality:
  Most of the software can be controlled via network commands to interact with the rest. If the software supports such commands, this entry tells on which port the software is listening.

- Computer:
  If the software is meant to be running on a certain computer, this is specified here. The version of python for running the code is given, too.

- Network connections:
  If the software communicates with other software, all programs which receive orders from this software are listed here.

- Hardware connections:
  If the program interfaces directly with hardware, the hardware and the address is given here.

- The functionality of the software is described.

- Implemented commands:
  If the software has server functionality it will respond to given commands. These commands and their function are listed here. The `ping` command can be sent to every software with server functionality and it will respond `pong`. This command is

used to check if the software is still responsive and since it is a universal command, it is not listed in the following documentation.

## E.1. “Aufwaermen”

- Server functionality: port 20017
- Computer: 134.34.142.60 Windows 7 python 3
- Network connections to:
  - “TemperaturLogger” on 134.34.142.60:20000 appendix E.38 on page 350

This software is designed to speed up the warm-up of the cryostat after the main measurement has been completed. To achieve the speedup, the the program uses the temperature control of the setup to actively heat the sample chamber and the gas outside of the chamber.
To limit the effects of excessive temperature differences within the chamber and between the chamber and the outside gas, several restrictions are implemented. The configuration file `config.conf` contains their parameters. The sample chamber and the vaporizer temperature are controlled separately and checked every six seconds. As long as the heating power is below 95 % the temperature setpoint is increased according to the warm-up rate configuration of $0.2\,\mathrm{K\,min^{-1}}$ unless the following limits are reached: The temperature setpoints must not exceed the end temperature of 295 K for the vaporizer and 280 K for the sample chamber. The difference between the setpoints of vaporizer and sample chamber must be less than 50 K. The ratio between the setpoints must not be bigger than 1.5.
The software is controlled by the program “MeasurementController” (see appendix E.23 on page 341) and implements the following commands:

- `kill`
  This stops the warm-up and any further changes of the setpoints, however, the heaters remain active.
  The command returns `done` and changes the state to `killing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `killed`.

- `pause`
  This pauses the warm-up by keeping the setpoints constant.
  The command returns `done` and changes the state to `pausing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `paused`.

- `halt`
  This is equivalent to `pause`.
  The command returns `done` and changes the state to `halting`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `halted`.

- `continue`
  This resumes the warm-up after `halt` or `pause`.
  The command returns `done` and changes the state to `continuing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `running`.

- `start`
  This starts the warm-up procedure.
  The command returns `done` and changes the state to `starting`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `running`.

- `stuck`
  When the software failed to update the current operation indicating an error, the software state is set to `stuck`. This has no further implications and only serves as an indication to the user.
  The command returns `done` and changes the state to `stuck`. It returns `failed` when the current state does not allow the change.

- `getTimeSinceOperationUpdate`
  This command returns the time in seconds since the last update of the current operation.

- `getOperation`
  This command returns the string of the current operation.

- `getState`
  This command returns the state of the software. Possible values are `continuing`, `finished`, `halted`, `halting`, `killed`, `killing`, `not started`, `paused`, `pausing`, `problem`, `running`, `starting`, or `stuck`.

- `getProgress`
  This command returns the progress of the measurements as a floating point number $\geq 0$ and $\leq 1$.
  When the progress is not defined it returns `no data`.

- `getElapsed`
  This command returns the elapsed time in seconds since the start of the warm-up excluding any pausing and halting times.
  When the time is not defined it returns `no data`.

- `getRemaining`
  This command returns the estimated remaining measurement time in seconds.
  When the time is not defined it returns `no data`.

## E.2. "Aufwaermen_Langsam"

- Server functionality: port 20017

- Computer: 134.34.142.60 Windows 7 python 3
- Network connections to:
  - "TemperaturLogger" on 134.34.142.60:20000 appendix E.38 on page 350

This software is designed to speed up the warm-up of the cryostat after the main measurement has been completed. To achieve the speedup, the the program uses the temperature control of the setup to actively heat the gas outside of the chamber. The heater in the sample chamber is not used which slows the process down, but it maintains a homogeneous temperature inside the sample chamber. This is important when the warm-up is used to measure the temperature dependence of the resistance thermometer on the sample.
To limit the effects of excessive temperature differences between the chamber and the outside gas, several restrictions are implemented. The configuration file `config.conf` contains their parameters. The sample chamber and the vaporizer temperature are checked every six seconds. As long as the heating power of the vaporizer is below 95 % its temperature setpoint is increased according to the warm-up rate configuration of $0.2\,\mathrm{K\,min^{-1}}$ unless the following limits are reached: The temperature setpoint of the vaporizer must not exceed the end temperature of 295 K. The difference between the setpoints of vaporizer and the temperature of the sample chamber must be less than 50 K. The ratio between the setpoint and the temperature must not be bigger than 1.5.
The software is controlled by the program "MeasurementController" (see appendix E.23 on page 341) and implements the following commands:

- `kill`
  This stops the warm-up and any further changes of the setpoints, however, the heaters remain active.
  The command returns `done` and changes the state to `killing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `killed`.

- `pause`
  This pauses the warm-up by keeping the setpoints constant.
  The command returns `done` and changes the state to `pausing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `paused`.

- `halt`
  This is equivalent to `pause`.
  The command returns `done` and changes the state to `halting`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `halted`.

- `continue`
  This resumes the warm-up after `halt` or `pause`.
  The command returns `done` and changes the state to `continuing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `running`.

- `start`
  This starts the warm-up procedure.
  The command returns `done` and changes the state to `starting`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `running`.

- `stuck`
  When the software failed to update the current operation indicating an error, the software state is set to `stuck`. This has no further implications and only serves as an indication to the user.
  The command returns `done` and changes the state to `stuck`. It returns `failed` when the current state does not allow the change.

- `getTimeSinceOperationUpdate`
  This command returns the time in seconds since the last update of the current operation.

- `getOperation`
  This command returns the string of the current operation.

- `getState`
  This command returns the state of the software. Possible values are `continuing`, `finished`, `halted`, `halting`, `killed`, `killing`, `not started`, `paused`, `pausing`, `problem`, `running`, `starting`, or `stuck`.

- `getProgress`
  This command returns the progress of the measurements as a floating point number $\geq 0$ and $\leq 1$.
  When the progress is not defined it returns `no data`.

- `getElapsed`
  This command returns the elapsed time in seconds since the start of the warm-up excluding any pausing and halting times.
  When the time is not defined it returns `no data`.

- `getRemaining`
  This command returns the estimated remaining measurement time in seconds.
  When the time is not defined it returns `no data`.

## E.3. "Aufwaermen_LangsamAutark"

- Computer: 134.34.142.60 Windows 7 python 3

- Network connections to:
  - "TemperaturLogger" on 134.34.142.60:20000 appendix E.38 on page 350

This software performs exactly the same functions as the "Aufwaermen_Langsam" software (see appendix E.2 on page 325), however it does not have server functions. The measurement process is controlled by buttons in the software itself.
This allows to have this software running while another program is performing measurements.

## E.4. "ConductanceCalculator"

- Server functionality: port 20019
- Computer: 134.34.142.60 Windows 7 python 3
- Network connections to:
  - "Multimeter" on 134.34.142.60:20018 appendix E.27 on page 343

This software uses the "Multimeter" software to calculate the conductance of the constriction. It assumes an applied bias voltage of 2 times 10 mV, a series resistance of 20 kΩ. The measured voltage is the voltage drop across the sample amplified by a factor of 100. All configuration details are in the `config.conf` file.
The software implements the following network command:

- `getConductance`
  This command returns the conductance of the sample in units of the conductance quantum $G_0$.
  When the calculation is not possible it returns `no data`.

## E.5. "ConductanceLogger"

- Computer: 134.34.142.60 Windows 7 python 3
- Network connections to:
  - "TemperaturLogger" on 134.34.142.60:20000 appendix E.38 on page 350
- Hardware connections to:
  - Zurich Instruments Lock-In Amplifier

This software checks and logs data every ten seconds. The data is the timestamp, the temperature of the sample chamber and the vaporizer, the conductance as calculated from the lock-in x values, x and y components of the voltage across the sample, and x and y components of the current through the sample.

## E.6. "DataCompression"

- Server functionality: port 20021
- Computer: 134.34.142.61 Windows 7 python 2.7

During the pulsed measurements, the setup saves five different files for the different currents with each file containing a sequence of pulses. This software takes the pulse sequences and averages all pulses together. This is done separately for all five files and their output is combined into a single matlab matrix file. The use of this precompression shortens the time needed for the analysis and significantly reduces the needed amount of data storage. The software implements the following network command:

- `<filename>`
  This command adds the file to the processing queue if the file is not already in the queue. If the queue was empty it will start the compression process.
  It always returns `done`.

## E.7. "DataCompression_IV"

- Server functionality: port 20023
- Computer: 134.34.142.61 (Virtual Machine) Linux python 2.7

This python program is just an interface which handles the network communications and the queue. The compression of the data is handled by a program written in C, because neither matlab nor python is efficient enough to handle the necessary data compression in real time.
The C code takes the files generated by the pulsed thermovoltage measurements which contain a sequence of laser pulses while the bias voltage is much faster sinusoid. The position of each pulse is determined, and 2 ms of data before, during, and long after the pause are fitted with a sinusoid. The amplitudes and offsets of the fits for the current and the voltage with their errors are stored for each pulse in an `h5` file as output. This procedure saves a lot of analysis time and storage space.
The software implements the following network command:

- `<filename>`
  This command adds the file to the processing queue if the file is not already in the queue. If the queue was empty it will start the compression process.
  It always returns `done`.

## E.8. "DebugClient"

This software implements a very simple terminal based network client for my software. It allows the user to manually input network commands to different software and shows the result. It is not intended to be used in the normal setup operation, but only for debugging.

## E.9. “DruckLogger”

- Server functionality: port 20001
- Computer: 134.34.142.62 Linux python 2.7
- Hardware connections to:
  - Pfeiffer TPG261 on `ASRL/dev/ttyUSBDelock_renumbered4::INSTR`

This program reads out the pressure from the pressure sensor and saves it into a log file. The software implements the following network command:

- `getPressure`
  This command returns the pressure in mbar.
  When the calculation is not possible it returns `no data`.

## E.10. “Faulhaber”

- Server functionality: port 20004
- Computer: 134.34.142.62 Linux python 2.7
- Hardware connections to:
  - Faulhaber MCDC 2805 on `ASRL/dev/ttyUSBfaulhaber::INSTR`

This software controls the Faulhaber motor which breaks the sample. It shows the position an speed and allows to move the motor to any position with any speed. The speed and position are limited by settings in the `config.conf` file.
The software implements the following network commands:

- `getPosition`
  This command returns current position of the motor.
  When the communication to the controller is not possible it returns `no data`.
- `getSpeed`
  This command returns current speed of the motor.
  When the communication to the controller is not possible it returns `no data`.
- `startMoveTo␣<Number>[␣<Speed>]`
  This command starts moving the motor to the desired position. If the optional speed is given, the move will be performed with this speed, otherwise the maximum motor speed will be used.
  When an error occurs during this command it returns `failed`, otherwise it returns `done`.
- `completeMoveTo␣<Number>[␣<Speed>]`
  This command moves the motor to the desired position. If the optional speed is given, the move will be performed with this speed, otherwise the maximum motor

speed will be used.
When an error occurs during this command it returns `failed`, otherwise it waits until the motion is finished and then it returns `done`.

- `waitForEndOfMove`
  This command waits until the motor stops. Then it returns `done`.
  When an error occurs during this command it returns `failed`.

- `stop`
  This command stops the motor.
  When an error occurs during this command it returns `failed`, otherwise it returns `done`.

## E.11. "FaulhaberClient"

- Computer: 134.34.142.60 Windows 7 python 2.7
- Network connections to:
  - "Faulhaber" on 134.34.142.60:20018 appendix E.10 on page 330

This program can send commands to the "Faulhaber" software to move the motor to a given position with a given maximum speed. This is necessary since the motor itself is connected to a different computer and it is more convenient to be able to control the motor from this computer.

## E.12. "Gaszaehler"

- Server functionality: port 20010
- Computer: 134.34.142.62 Linux python 2.7
- Hardware connections to:
  - "gascounter" NIM-insert on `ASRL/dev/ttyUSBgascounter::INSTR` appendix D.8 on page 288

This program reads the helium evaporation rate and the absolute reading of the helium gas counter. The values are accessible to other programs and they are logged every 10 s. The software implements the following network commands:

- `getAbsolute`
  This command returns the reading of the gas counter in liters with an accuracy of 10 L.
  When the volume is not available it returns `no data`.

- `getRate`
  This command returns the counting rate of the gas counter in liters per second.
  When the rate is not available it returns `no data`.

## E.13. “HeliumLogger”

- Server functionality: port 20003
- Computer: 134.34.142.62 Linux python 2.7
- Hardware connections to:
  - AMI Model 135 Liquid Helium Level Instrument on `ASRL/dev/ttyUSBDelock_renumbered3::INSTR`

This software reads the helium level from the hardware. With the dimensions of the bath, this level is then converted into a volume. The volume can be accessed via the network and it is logged every 10 s.
The software implements the following network command:

- `getVolume`
  This command returns the liquid helium volume in liters.
  When the volume is not available it returns `no data`.

## E.14. “Hilfskabel”

- Server functionality: port 20013
- Computer: 134.34.142.62 Linux python 2.7
- Hardware connections to:
  - “auxiliary wiring” NIM insert on `ASRL/dev/ttyUSBled::INSTR`
    appendix D.6 on page 260

This software allows to control the LEDs inside the sample chamber. The brightness level is logged every 10 s.
The software implements the following network commands:

- `getBrightness`
  This command returns the brightness of the LEDs.
  When the brightness is not available it returns `no data`.
- `setBrightness␣<Number>`
  This command sets the brightness of the LEDs.
  When the number cannot be understood it returns `failed`, when the command was sent but an error occurred it returns `tried and failed`, and if everything worked it returns `done`.

## E.15. “Kamera”

- Server functionality: port 20005
- Computer: 134.34.142.62 Linux python 2.7
- Hardware connections to:
  - “camera” NIM insert on `ASRL/dev/ttyUSBcamera::INSTR`
    appendix D.10 on page 302

This software is the interface to the NIM insert and allows to take pictures. This can be done by the button in the GUI or by a network command. The timestamp of every picture is saved in a log file.
The software implements the following network command:

- `takePicture`
  This command returns `done`, except if an error occurred. Then it returns `failed`.

## E.16. “Kartenaufzeichnen_V2”

- Server functionality: port 20017
- Computer: 134.34.142.60 Windows 7 python 3
- Network connections to:
  - “ThorlabsMotor” (X-direction) on 134.34.142.60:20006
    appendix E.43 on page 362
  - “ThorlabsMotor” (Y-direction) on 134.34.142.60:20007
    appendix E.43 on page 362
  - “Laser” on 134.34.142.60:20016 appendix E.17 on page 335
  - “TransCOM” on 134.34.142.60:20015 appendix E.45 on page 364
  - “Kamera” on 134.34.142.62:20005 appendix E.15 on page 333
  - “SourceArb” on 134.34.142.60:20014 appendix E.33 on page 347
  - “DataCompression” on 134.34.142.61:20021 appendix E.6 on page 329

This software is used to map the resistance change in the sense leads due to the laser heating at different positions and powers. A primary axis and two diagonally opposing corners are used to define the scanning area. With a given number of steps in the two directions a grid is defined. During the measurement, the laser focus is moved to each grid position. There a picture of the position is taken with the laser at low power. Then the laser is set to provide 4 ms long pulses with constant power every 34 ms. The arbitrary waveform is used for the pulses to compensate the laser power drooping over time due to the heating of the laser diode. Using the transient recorder, the voltage drop across the leads and the bias current is measured for five seconds at different bias voltages (-200, -100, 0, 100, and 200 mV). The described procedure is illustrated by the following pseudo code.

for each position in the grid:
  move laser focus to the position
  take a picture with the laser at low power
  set the laser to pulsing (4ms every 34ms)
  for bias voltage in [−200, −100, 0, 100, 200mV]:
    set the bias voltage
    acquire 5s of data
  compress the data to save space and time

The software is controlled by the program "MeasurementController" (see appendix E.23 on page 341) and implements the following commands:

- `kill`
  This command stops the measurement. This is possible after each data acquisition. The command returns `done` and changes the state to `killing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `killed`.

- `pause`
  This pauses the measurement before moving the laser to the next spot.
  The command returns `done` and changes the state to `pausing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `paused`.

- `halt`
  This interrupts the measurement after the data acquisition before setting the next bias voltage.
  The command returns `done` and changes the state to `halting`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `halted`.

- `continue`
  This resumes the normal operation at the point where the process was interrupted.
  The command returns `done` and changes the state to `continuing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `running`.

- `start`
  This starts the measurement with the given settings.
  The command returns `done` and changes the state to `starting`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `running`.

- `stuck`
  When the software failed to update the current operation indicating an error, the software state is set to `stuck`. This has no further implications and only serves as an indication to the user.
  The command returns `done` and changes the state to `stuck`. It returns `failed` when the current state does not allow the change.

- `getTimeSinceOperationUpdate`
  This command returns the time in seconds since the last update of the current operation.

- `getOperation`
  This command returns the string of the current operation.

- `getState`
  This command returns the state of the software. Possible values are `continuing`, `finished`, `halted`, `halting`, `killed`, `killing`, `not started`, `paused`, `pausing`, `problem`, `running`, `starting`, or `stuck`.

- `getProgress`
  This command returns the progress of the measurements as a floating point number $\geq 0$ and $\leq 1$.
  When the progress is not defined it returns `no data`.

- `getElapsed`
  This command returns the elapsed time in seconds since the start excluding any pausing and halting times.
  When the time is not defined it returns `no data`.

- `getRemaining`
  This command returns the estimated remaining measurement time in seconds.
  When the time is not defined it returns `no data`.

## E.17. "Laser"

- Server functionality: port 20016
- Computer: 134.34.142.60 Windows 7 python 2.7
- Hardware connections to:
  - HP 33200 Arbitrary Waveform Generator on `GPIB::10::INSTR`

This software is the interface to the arbitrary waveform generator driving the modulation input of the laser controller. It supports setting the output to a dc value, a pulse shape or a preprogrammed arbitrary shape.
The software implements the following network commands:

- `output␣[on|off]`
  This command turns the output of the waveform generator on or off. It returns `done` except if an error occurred. Then it returns `failed`.

- `dc␣<Value>`
  This command sets the output to dc at the given value in V. It returns `done` except if an error occurred. Then it returns `failed`.

- `pulse␣<Period>␣<Width>␣<High Level>␣<Low Level>`
  This command sets the output to a pulse waveform with the given parameters in s or V. It returns `done` except if an error occurred. Then it returns `failed`.

- `arb␣<Period>␣<High Level>␣<Low Level>`
  This command sets the output to the preprogrammed arbitrary waveform with the given parameters ins or V. It returns `done` except if an error occurred. Then it returns `failed`.

## E.18. "Laserfokus"

- Server functionality: port 20017
- Computer: 134.34.142.60 Windows 7 python 3
- Network connections to:
  - "ThorlabsMotor" (X-direction) on 134.34.142.60:20006
    appendix E.43 on page 362
  - "ThorlabsMotor" (Y-direction) on 134.34.142.60:20007
    appendix E.43 on page 362
  - "Laser" on 134.34.142.60:20016 appendix E.17 on page 335
  - "Multimeter" on 134.34.142.60:20018 appendix E.27 on page 343

This software is used to measure the reflectivity of the sample at different positions. Since the gold structures have well defined edges, the data can be used to deduce the shape of the laser focus. A primary axis and two diagonally opposing corners are used to define the scanning area. With a given number of steps in the two directions a grid is defined. During the measurement, the laser focus is moved to each grid position. Without the laser, the reflected light is measured by the photodiode connected to the multimeter in order to get the offset. Then the laser power is turned on and reflected light is measured again. The difference between the first and second measurement is the reflected laser.
The software is controlled by the program "MeasurementController" (see appendix E.23 on page 341) and implements the following commands:

- `kill`
  This stops the measurement before moving to the next spot.
  The command returns `done` and changes the state to `killing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `killed`.

- `pause`
  This pauses the measurement before moving to the next spot.
  The command returns `done` and changes the state to `pausing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `paused`.

- `halt`
  This is equivalent to `pause`.
  The command returns `done` and changes the state to `halting`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `halted`.

- `continue`
  This resumes the normal operation at the point where the process was interrupted.
  The command returns `done` and changes the state to `continuing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `running`.

- `start`
  This command starts the measurement with the given settings.
  The command returns `done` and changes the state to `starting`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `running`.

- `stuck`
  When the software failed to update the current operation indicating an error, the software state is set to `stuck`. This has no further implications and only serves as an indication to the user.
  The command returns `done` and changes the state to `stuck`. It returns `failed` when the current state does not allow the change.

- `getTimeSinceOperationUpdate`
  This command returns the time in seconds since the last update of the current operation.

- `getOperation`
  This command returns the string of the current operation.

- `getState`
  This command returns the state of the software. Possible values are `continuing`, `finished`, `halted`, `halting`, `killed`, `killing`, `not started`, `paused`, `pausing`, `problem`, `running`, `starting`, or `stuck`.

- `getProgress`
  This command returns the progress of the measurements as a floating point number $\geq 0$ and $\leq 1$.
  When the progress is not defined it returns `no data`.

- `getElapsed`
  This command returns the elapsed time in seconds since the start excluding any pausing and halting times.
  When the time is not defined it returns `no data`.

- `getRemaining`
  This command returns the estimated remaining measurement time in seconds.
  When the time is not defined it returns `no data`.

## E.19. “Laserkalibrierung”

- Computer: 134.34.142.60 Windows 7 python 3
- Network connections to:
  - “Laser” on 134.34.142.60:20016 appendix E.17 on page 335
  - “Powermeter” on 134.34.142.60:20020 appendix E.30 on page 346

This software sets the laser to a dc value, waits for 1 s for the readings to stabilize, and then reads the power measured by the powermeter. This is done for all voltages from 0 to 7 V in steps of 2 mV and the voltage and corresponding power are stored in a file.

## E.20. “Laserkarte”

- Server functionality: port 20017
- Computer: 134.34.142.60 Windows 7 python 3
- Network connections to:
  - “ThorlabsMotor” (X-direction) on 134.34.142.60:20006 appendix E.43 on page 362
  - “ThorlabsMotor” (Y-direction) on 134.34.142.60:20007 appendix E.43 on page 362
  - “Laser” on 134.34.142.60:20016 appendix E.17 on page 335
  - “LockInAcquisition” on 134.34.142.60:20025 appendix E.22 on page 340
  - “Kamera” on 134.34.142.62:20005 appendix E.15 on page 333
  - “WiderstandsmessungMaeander” on 134.34.142.60:20024 appendix E.54 on page 371

This software is used to map the temperature response of the platinum resistance thermometer to the laser heating at different positions and powers. A primary axis and two diagonally opposing corners are used to define the scanning area. With a given number of steps in the two directions a grid is defined. During the measurement, the laser focus is moved to each grid position. There a picture of the position is taken with the laser at low power. Then the laser excitation voltage is set from 0 to 6 V in steps of 0.5 V and the resistance of the thermometer is measured for each voltage after one second of settling time. This behavior is summarized in the following pseudo code:

```
for each position in the grid:
  move laser focus to the position
  take a picture with the laser at low power
  for excitation voltage in [0, 0.5, ..., 6]:
    set the laser excitation voltage
    wait for one second
    measure the resistance
```

The software is controlled by the program "MeasurementController" (see appendix E.23 on page 341) and implements the following commands:

- `kill`
  This stops the measurement immediately after the resistance measurement.
  The command returns `done` and changes the state to `killing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `killed`.

- `pause`
  This pauses the measurement before moving to the next position.
  The command returns `done` and changes the state to `pausing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `paused`.

- `halt`
  This pauses the measurement after completing the resistance measurement. It may interrupt during the measurement on one point.
  The command returns `done` and changes the state to `halting`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `halted`.

- `continue`
  This resumes the normal operation at the point where the process was interrupted.
  The command returns `done` and changes the state to `continuing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `running`.

- `start`
  This starts the measurement with the given settings.
  The command returns `done` and changes the state to `starting`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `running`.

- `stuck`
  When the software failed to update the current operation indicating an error, the software state is set to `stuck`. This has no further implications and only serves as an indication to the user.
  The command returns `done` and changes the state to `stuck`. It returns `failed` when the current state does not allow the change.

- `getTimeSinceOperationUpdate`
  This command returns the time in seconds since the last update of the current operation.

- `getOperation`
  This command returns the string of the current operation.

- `getState`
  This command returns the state of the software. Possible values are `continuing`, `finished`, `halted`, `halting`, `killed`, `killing`, `not started`, `paused`, `pausing`, `problem`, `running`, `starting`, or `stuck`.

- `getProgress`
  This command returns the progress of the measurements as a floating point number $\geq 0$ and $\leq 1$.
  When the progress is not defined it returns `no data`.

- `getElapsed`
  This command returns the elapsed time in seconds since the start excluding any pausing and halting times.
  When the time is not defined it returns `no data`.

- `getRemaining`
  This command returns the estimated remaining measurement time in seconds.
  When the time is not defined it returns `no data`.

## E.21. "LN2"

- Server functionality: port 20009
- Computer: 134.34.142.62 Linux python 2.7
- Hardware connections to:
  - "liquid nitrogen" NIM insert on `ASRL/dev/ttyUSBln2::INSTR`
    appendix D.7 on page 273

This software is an interface to the liquid nitrogen level meter NIM insert. It shows the current, the empty, and the full frequency as well as the level in %. Its main purpose is to make the level reading accessible to other software.
The software implements the following network command:

- `getLevel`
  This command returns the liquid nitrogen level in %.
  When the data is not valid it returns `no data`.

## E.22. "LockInAcquisition"

- Server functionality: port 20025
- Computer: 134.34.142.60 Windows 7 python 3
- Hardware connections to:
  - Zurich Instruments Lock-In Amplifier

This software can save the data measured by the lock-in amplifier into an h5-file. It grabs the data (time, frequency, phase offset, x, and y) for each channel. In the h5-file there is a group named "channel?" for each channel (0 to 5) and in each group the software creates datasets for the five different data values.
The software implements the following network commands:

- `start␣<Filename>`
  This command starts saving the lock-in data into a file with the given name. It always returns `done`.

- `stop`
  This command stops the data saving and closes the file. It always returns `done`.

## E.23. "MeasurementController"

- Computer: 134.34.142.60 Windows 7 python 3
- Network connections to:
  - whatever server is listening on 134.34.142.60:20017

This software controls most of the measurement programs. It can send the "start", "kill", "pause", "halt", "continue" and "stuck" commands. The "stuck" command gets sent if the software detects that the last update of the current operation of the measurement software is more than 60 s ago.
The program is also configured to write e-mails in the following cases: The measurement gets stuck or changes the state to "problem" or "finished". To test this e-mail function a test mail is sent when the program is started. To prevent spamming in case of the measurement software misbehaving, this software has a check box that enables the sending of the e-mails. This check box is automatically set when a measurement is started and gets cleared when the first e-mail was sent. This way, only one mail is sent if a problem occurs. If the user can fix the problem and the measurement continues, it is necessary to the check box manually to be informed of further problems.

## E.24. "Monitor_P3"

- Network connections to:
  - "TemperaturLogger" on 134.34.142.60:20000 appendix E.38 on page 350
  - "Turbopump" on 134.34.142.62:20002 appendix E.46 on page 366
  - "DruckLogger" on 134.34.142.62:20001 appendix E.9 on page 330
  - "Hilfskabel" on 134.34.142.62:20011 appendix E.14 on page 332
  - "LN2" on 134.34.142.62:20009 appendix E.21 on page 340
  - "HeliumLogger" on 134.34.142.62:20003 appendix E.13 on page 332
  - "Gaszaehler" on 134.34.142.62:20010 appendix E.12 on page 331

  - "Faulhaber" on 134.34.142.62:20004 appendix E.10 on page 330
  - "ThorlabsMotor" on 134.34.142.60:20006 appendix E.43 on page 362
  - "ThorlabsMotor" on 134.34.142.60:20007 appendix E.43 on page 362
  - measurement software on 134.34.142.60:20017

This program shows all relevant data for monitoring the setup. The data is updated in real time. This allows to quickly see the complete state of the experimental setup, which makes it very useful when working on the setup.

## E.25. "Monitor_Shell"

- Network connections to:
  - "TemperaturLogger" on 134.34.142.60:20000 appendix E.38 on page 350
  - "Turbopump" on 134.34.142.62:20002 appendix E.46 on page 366
  - "DruckLogger" on 134.34.142.62:20001 appendix E.9 on page 330
  - "Hilfskabel" on 134.34.142.62:20011 appendix E.14 on page 332
  - "LN2" on 134.34.142.62:20009 appendix E.21 on page 340
  - "HeliumLogger" on 134.34.142.62:20003 appendix E.13 on page 332
  - "Gaszaehler" on 134.34.142.62:20010 appendix E.12 on page 331
  - "Faulhaber" on 134.34.142.62:20004 appendix E.10 on page 330
  - "ThorlabsMotor" on 134.34.142.60:20006 appendix E.43 on page 362
  - "ThorlabsMotor" on 134.34.142.60:20007 appendix E.43 on page 362
  - measurement software on 134.34.142.60:20017

This program shows all relevant data for monitoring the setup. It is designed to run in a command prompt, mainly for checking the status when connecting to the setup via ssh. It does not update the values, it just prints them once. The real time updating is not needed when checking the state remotely.

## E.26. "Motor"

- Server functionality: port 20012
- Computer: 134.34.142.62 Linux python 2.7
- Hardware connections to:
  - "motor surveillance" NIM insert on `ASRL/dev/ttyUSBmotor::INSTR` appendix D.5 on page 249

This software provides an interface to the motor surveillance NIM insert. If shows if the limit switches are currently active, if they have been active, and gives the possibility to reset them. The state of the limit switches is also saved into a log file every 10 s.
The software implements the following network commands:

- `getUpperLimit`
  This command returns if the upper limit switch is currently active. It returns either `True` or `False`. The latter is the normal operating condition. If the data is not available it returns `no data`.

- `getLowerLimit`
  This command returns if the lower limit switch is currently active. It returns either `True` or `False`. The latter is the normal operating condition. If the data is not available it returns `no data`.

- `getUpperLimitTripped`
  This command returns if the upper limit switch was active since the last reset. It returns either `True` or `False`. The latter is the normal operating condition. If the data is not available it returns `no data`.

- `getLowerLimitTripped`
  This command returns if the lower limit switch was active since the last reset. It returns either `True` or `False`. The latter is the normal operating condition. If the data is not available it returns `no data`.

- `resetUpperLimitTripped`
  This command resets the upper limit switch. Normally it returns `done`, if an error occurred it returns `failed`.

- `resetLowerLimitTripped`
  This command resets the lower limit switch. Normally it returns `done`, if an error occurred it returns `failed`.

## E.27. "Multimeter"

- Server functionality: port 20018

- Computer: 134.34.142.60 Windows 7 python 2.7

- Hardware connections to:
  - Agilent 34401A on `GPIB0::4::INSTR`

This software is the interface to the Agilent 34401A multimeter. It can get the current voltage reading and it can change the averaging times which changes the reading speed. The two averaging times are 1 or 100 power line cycles.
The software implements the following network commands:

- `getReading`
  This command returns the current reading. When no data is available it returns `no data`.

- `fast`
  This command sets the multimeter to perform fast readings with 1 power line cycle averaging and waits for the command to be executed. It always returns `no data`.

- `slow`
  This command sets the multimeter to perform fast readings with 100 power line cycles averaging and waits for the command to be executed. It always returns `no data`.

## E.28. "Multiplexer"

- Server functionality: port 20022
- Computer: 134.34.142.62 Linux python 2.7
- Hardware connections to:
  - "multiplexer" NIM insert on `ASRL/dev/ttyUSBmultiplexer::INSTR`
    appendix D.2 on page 200

This software if the interface to the multiplexer NIM insert. It shows the complete switching matrix with markers indicating the connections. There are also buttons for disconnecting everything or just individual inputs or outputs. Since the amount of different combinations that need to be checked for their state is rather large, the displayed connections can have up to several seconds of lag.
The software implements the following network commands:

- `clearAll`
  This command disconnects everything. It always returns `done`.

- `clearInput␣<Number>`
  This command disconnects any outputs from the given input. If the command could not be interpreted it returns `failed`. If the command was sent but an error occurred it returns `tried and failed`. If everything works it returns `done`.

- `clearOutput␣<Number>␣<Signal>`
  This command disconnects any inputs from the given output. The output signal can be either `true`, `sig`, or `signal` meaning the inner conductor of the BNC plug, or it can be `false`, `shi`, or `shield` referring to the shield. If the command could not be interpreted it returns `failed`. If the command was sent but an error occurred it returns `tried and failed`. If everything works it returns `done`.

- `connect␣<InputNumber>␣<OutputNumber>␣<Signal>`
  This command connects the input to the given output. The output signal can be either `true`, `sig`, or `signal` meaning the inner conductor of the BNC plug, or it

can be `false`, `shi`, or `shield` referring to the shield. If the command could not be interpreted it returns `failed`. If the command was sent but an error occurred it returns `tried and failed`. If everything works it returns `done`.

- `isConnected␣<InputNumber>␣<OutputNumber>␣<Signal>`
  This command checks if the input is connected to the given output. The output signal can be either `true`, `sig`, or `signal` meaning the inner conductor of the BNC plug, or it can be `false`, `shi`, or `shield` referring to the shield. If the command could not be interpreted it returns `failed`. If the command was sent but an error occurred it returns `tried and failed`. If everything works it returns either `True` or `False`.

## E.29. "Netzteil"

- Server functionality: port 20011
- Computer: 134.34.142.62 Linux python 2.7
- Hardware connections to:
  - "IceCube supply" NIM insert on `ASRL/dev/ttyUSBpowersupply::INSTR` appendix D.4 on page 233

This software is the interface to the IceCube supply NIM insert. It logs the temperature, the current on both rails, and the raw auxiliary reading on input 1 and 2 every 10 s. The software implements the following network commands:

- `getTemperature`
  This command returns the temperature of the power supply IceCube insert in °C. When the data is not available it returns `no data`.
- `getCurrrentPositive`
  This command returns the current of the positive supply rail in mA. When the data is not available it returns `no data`.
- `getCurrrentNegative`
  This command returns the current of the negative supply rail in mA. When the data is not available it returns `no data`.
- `getAuxReading␣<Number>`
  This command returns the value of the auxiliary input with the given number. The raw data is converted into the result with the parameters given in the configuration file. When the data is not available `no data` is returned and when an error occurred `failed` is returned.

## E.30. “Powermeter”

- Server functionality: port 20020
- Computer: 134.34.142.60 Windows 7 python 2.7
- Hardware connections to:
  - Thorlabs Powermeter PM100D
    on `USB0::0x1313::0x8078::PM002461::INSTR`

This software serves as an interface between the powermeter and the network based software. In reads the values of the powermeter continuously and provides upon request.
The software implements the following network command:

- `getReading`
  This command returns the measured power in W. When the data is not available it returns `no data`.

## E.31. “Probencharakterisierung”

- Computer: 134.34.142.60 Windows 7 python 2.7
- Network connections to:
  - “Multiplexer” on 134.34.142.62:20022 appendix E.28 on page 344
- Hardware connections to:
  - Agilent 34401A multimeter on `GPIB::4::INSTR`
  - Keithley 2400 source meter on `GPIB::2::INSTR`

This software performs four point measurements to determine all resistances of samples for the pulsed measurements. To avoid offset voltages, the measurement is performed with the source turned on and off. The multiplexer is used to switch different lines to the source and the multimeter.

## E.32. “Source”

- Server functionality: port 20014
- Computer: 134.34.142.60 Windows 7 python 2.7
- Hardware connections to:
  - HP 33200 on `GPIB0::5::INSTR`

This software is designed to control the source that supplies the bias voltage to the sample. The dc value can be set and the output can be turned on or off. Additionally, the applied voltage is logged every 10 s.
The software implements the following network commands:

- `setVoltage␣<Value>`
  This command sets the output to the given dc value. Normally it returns `done`. If an error occurred it returns `failed`.

- `enable`
  This command enables the output and returns `done`.

- `disable`
  This command disables the output and returns `done`.

## E.33. "SourceArb"

- Server functionality: port 20014
- Computer: 134.34.142.60 Windows 7 python 2.7
- Hardware connections to:
  - HP 33200 on `GPIB0::5::INSTR`

This software is designed to control the source that supplies the bias voltage to the sample. The dc value can be set and the output can be turned on or off, but since an arbitrary waveform generator is used, more complex waveforms are possible. The software supports sinusoid, square, and ramp with settable frequency, amplitude and offset. To keep older measurement software, this software is backward compatible with the "source" software. The software implements the following network commands:

- `setVoltage␣<Value>`
  This command sets the output to the given dc value. Normally it returns `done`. If an error occurred it returns `failed`.

- `setWaveform␣<Waveform>␣<Frequency>␣<Amplitude>␣<Offset>`
  This command sets the desired waveform. The possible waveforms are `sinusoid`, `square`, and `ramp`. The frequency is in Hz, and the amplitude and the offset are in V.

- `setBurst␣<Frequency>␣<Cycles>␣<Amplitude>␣<Offset>`
  This command sets the waveform to sinusoid and the generator to burst mode. The frequency is in Hz, cycles is the number of cycles, and the amplitude and the offset are in V.

- `enable`
  This command enables the output and returns `done`.

- `disable`
  This command disables the output and returns `done`.

## E.34. “Starter_P3”

- Computer: 134.34.142.62 Linux python 3 or 134.34.142.60 Windows 7 python 3
- Network connections to:
  - “Turbopump” on 134.34.142.62:20002 appendix E.46 on page 366
  - “DruckLogger” on 134.34.142.62:20001 appendix E.9 on page 330
  - “TemperaturLogger” on 134.34.142.60:20000 appendix E.38 on page 350
  - “HeliumLogger” on 134.34.142.62:20003 appendix E.13 on page 332
  - “Faulhaber” on 134.34.142.62:20004 appendix E.10 on page 330
  - “Kamera” on 134.34.142.62:20003 appendix E.15 on page 333
  - “ThorlabsMotor” on 134.34.142.60:20007 appendix E.43 on page 362
  - “Gaszaehler” on 134.34.142.62:20010 appendix E.12 on page 331
  - “Hilfskabel” on 134.34.142.62:20013 appendix E.14 on page 332
  - “LN2” on 134.34.142.62:20009 appendix E.21 on page 340
  - “Motor” on 134.34.142.62:20012 appendix E.26 on page 342
  - “Netzteil” on 134.34.142.62:20011 appendix E.29 on page 345
  - “Stromausfall” on 134.34.142.62:20008 appendix E.35 on page 349
  - “SourceArb” on 134.34.142.60:20014 appendix E.33 on page 347
  - “Laser” on 134.34.142.60:20016 appendix E.17 on page 335
  - “Powermeter” on 134.34.142.60:20020 appendix E.30 on page 346
  - “Multiplexer” on 134.34.142.62:20022 appendix E.28 on page 344
  - “Multimeter” on 134.34.142.60:20018 appendix E.27 on page 343
  - “WiderstandsmessungMaeander” on 134.34.142.60:20024 appendix E.54 on page 371
  - “TransCOM” on 134.34.142.60:20015 appendix E.45 on page 364
  - “Laserkarte” on 134.34.142.60:20017 appendix E.20 on page 338
  - “Aufwaermen” on 134.34.142.60:20017 appendix E.1 on page 324
  - “Aufwaermen_Langsam” on 134.34.142.60:20017 appendix E.2 on page 325
  - “Thermospannungsmessung_statisch” on 134.34.142.60:20017 appendix E.40 on page 354
  - “Widerstandsmessung_Laserwechsel” on 134.34.142.60:20017 appendix E.53 on page 368

This software can start all python programs which are supposed to run on the computer. It also checks that they are still running by sending regular ping requests. The benefit of using this program instead of starting everything from a terminal is that you do not need that many terminal programs. All error messages from all programs are collected in one error file.

## E.35. "Stromausfall"

- Server functionality: port 20008
- Computer: 134.34.142.62 Linux python 2.7
- Hardware connections to:
  - "Power surveillance" NIM insert on `ASRL/dev/ttyUSBpowermonitoring::INSTR` appendix D.11 on page 313

This software logs the voltages and the state of the external power to a file every 10 s. The software implements the following network commands:

- `getVoltage06`
  This command returns measured voltage of the 6 V rail in V. When the data is not available it returns `no data`.
- `getVoltage12`
  This command returns measured voltage of the 12 V rail in V. When the data is not available it returns `no data`.
- `getVoltage24`
  This command returns measured voltage of the 24 V rail in V. When the data is not available it returns `no data`.
- `getExternPower`
  This command returns `True` if the external power is present and `false` if not. When the data is not available it returns `no data`.

## E.36. "TemperatureAdjuster"

- Network connections to:
  - "TemperaturLogger" on 134.34.142.60:20000 appendix E.38 on page 350

This software is designed to adjust the temperature of the vaporizer with the aim of having the sample chamber heater at 50 % heating power. It is not meant to correct short excursions but designed to change the temperature significantly slower than the feedback in the sample chamber temperature control to prevent unnecessary disturbance.
When the sample chamber heating power is above 55 % in sixty consecutive measurements performed every second, the setpoint of the vaporizer is set to the current temperature plus 0.02 K. Then the next 60 measurements are performed. If the power is below 45 %, the temperature is reduced by 0.02 K in similar fashion. The setpoint of the vaporizer temperature is not changed if the heating power is not in the same range in all ten measurements. Additionally the new values for the setpoint are limited to be in the range from 79 K to 295 K in order to prevent damage to the setup in cases of software or hardware failure.

## E.37. "TemperatureController"

- Network connections to:
  - "TemperaturLogger" on 134.34.142.60:20000 appendix E.38 on page 350

This software is able to control the temperature in the sample chamber if the heater is broken. The vaporizer temperature setpoint is changed by the output of a PID controller in order to maintain the desired temperature in the sample chamber. There is no need to use this software under normal operation conditions of the setup.

## E.38. "TemperaturLogger"

- Server functionality: port 20000
- Computer: 134.34.142.60 Windows 7 python 2.7
- Hardware connections to:
  - LakeShore Model 335 Cryogenic Temperature Controller on `GPIB0::12::INSTR`

This software is the interface to the LakeShore temperature controller. It shows and logs the current temperature, the setpoint, the heating power, and the heater range for the vaporizer thermometer and the thermometer in the sample chamber. The heater ranges and the setpoints can be set manually.
Additionally the software also provides plots for the temperatures on different timescales: 20 min duration with 1 s resolution, 10 h duration with 10 s resolution, and 72 h duration with 1 min resolution.
The software implements the following network commands:

- `getTemperatureA`
  This command returns the temperature of the vaporizer in K. When the data is not available it returns `no data`.
- `getTemperatureB`
  This command returns the temperature of the sample chamber in K. When the data is not available it returns `no data`.
- `getSetpointA`
  This command returns the temperature setpoint of the vaporizer in K. When the data is not available it returns `no data`.
- `getSetpointB`
  This command returns the temperature setpoint of the sample chamber in K. When the data is not available it returns `no data`.
- `getPowerA`
  This command returns the heating power of the vaporizer in %. When the data is not available it returns `no data`.

- `getPowerB`
  This command returns the heating power of the sample chamber in %. When the data is not available it returns `no data`.

- `getRangeA`
  This command returns the heater of the vaporizer. The possible values are `Off`, `Low`, `Medium`, and `High`. When the data is not available it returns `no data`.

- `getRangeB`
  This command returns the heater of the sample chamber. The possible values are `Off`, `Low`, `Medium`, and `High`. When the data is not available it returns `no data`.

- `setSetpointA␣<Temperature>`
  This command sets the setpoint of the vaporizer. The temperature is given in K. Normally, the command returns `done`, but if an error occurs it returns `failed`.

- `setSetpointB␣<Temperature>`
  This command sets the setpoint of the sample chamber. The temperature is given in K. Normally, the command returns `done`, but if an error occurs it returns `failed`.

## E.39. "Thermospannungsmessung_Molekuele"

- Server functionality: port 20017
- Computer: 134.34.142.60 Windows 7 python 3
- Network connections to:
  - "ThorlabsMotor" (X-direction) on 134.34.142.60:20006
    appendix E.43 on page 362
  - "ThorlabsMotor" (Y-direction) on 134.34.142.60:20007
    appendix E.43 on page 362
  - "Laser" on 134.34.142.60:20016 appendix E.17 on page 335
  - "LockInAcquisition" on 134.34.142.60:20025 appendix E.22 on page 340
  - "Kamera" on 134.34.142.62:20005 appendix E.15 on page 333
  - "SourceArb" on 134.34.142.60:20014 appendix E.33 on page 347
  - "Faulhaber" on 134.34.142.62:20004 appendix E.10 on page 330
- Hardware connections to:
  - Zurich Instruments Lock-In Amplifier

This software performs the static thermovoltage measurements on molecules using only the lock-in amplifier for the data acquisition. For the dc measurement, the lock-in amplifier is configured for a frequency of 0 Hz. Using only the lock-in amplifier eliminates the problem generated by the input offsets of the transient recorder resulting in wrong conductances. During the procedure, opening and closing traces are recorded with the laser turned on

and off and the laser focus on both sides of the constriction.
Every trace begins with a picture with the laser at low power to document the position of the focus. Then the laser is set to the required power and the software waits for 5 s to have a stable laser power and a stable thermal distribution inside of the sample. The data acquisition with the lock-in amplifier is started subsequently and the motor starts to open or close the constriction. The conductance is continuously monitored with the lock-in data and used to check if the contact has been moved to the limit. The conductance is also used to modify the bias amplitude to have good resolution at low conductances without overloading the IV-converter at closed contacts. When the junction has been moved enough for the conductance to reach the limit, the motor and the data acquisition is stopped, and then another picture of the position of the laser focus is taken at low laser power. This is repeated for every trace until the user stops the measurement.
The behaviour of the software is summarized in the following pseudo code.

```
while True:
  for side in ["sensor","empty"]:
    move the focus to the correct side
    for laser state in ["off","on"]:
      for direction in ["opening","closing"]:
        take picture at low laser power
        if laser == "on":
          turn laser on
        else:
          turn laser off
        wait 5s
        start data acquisition withthe lock−in amplifier
        if direction == "opening":
          move motor in the opening direction
          wait until conductance < 1e−7 G0
        else:
          move motor in the closing direction
          wait until conductance > 3 G0
        stop the data acquisition
        take picture at low laser power
```

The software is controlled by the program “MeasurementController” (see appendix E.23 on page 341) and implements the following commands:

- `kill`
  This command stops the measurement and interrupts the current trace. The motor and the data acquisition are stopped.
  The command returns `done` and changes the state to `killing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `killed`.

- `pause`
  This pauses the measurement at the end of a closing trace. This command is designed to be used when manual interact with the setup is necessary (e.g. refilling

the coolant).
The command returns `done` and changes the state to `pausing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `paused`.

- `halt`
  This command interrupts the currently active measurement. The motor and the data acquisition are stopped. The trace can be continued in the same direction with a new output file when the measurement is resumed.
  The command returns `done` and changes the state to `halting`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `halted`.

- `continue`
  This resumes the normal operation at the point where the process was interrupted. The command returns `done` and changes the state to `continuing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `running`.

- `start`
  This starts the measurement with the settings shown in the user interface.
  The command returns `done` and changes the state to `starting`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `running`.

- `stuck`
  When the software failed to update the current operation indicating an error, the software state is set to `stuck`. This has no further implications and only serves as an indication to the user.
  The command returns `done` and changes the state to `stuck`. It returns `failed` when the current state does not allow the change.

- `getTimeSinceOperationUpdate`
  This command returns the time in seconds since the last update of the current operation.

- `getOperation`
  This command returns the string of the current operation.

- `getState`
  This command returns the state of the software. Possible values are `continuing`, `finished`, `halted`, `halting`, `killed`, `killing`, `not started`, `paused`, `pausing`, `problem`, `running`, `starting`, or `stuck`.

- `getProgress`
  This measurement has no end, therefore the progress is not defined and it returns `no data`.

- `getElapsed`
  This command returns the elapsed time in seconds since the start excluding any pausing and halting times.
  When the time is not defined it returns `no data`.

- `getRemaining`
  Since the measurement has no end, the remaining time is not defined and it returns `no data`.

## E.40. "Thermospannungsmessung_statisch"

- Server functionality: port 20017
- Computer: 134.34.142.60 Windows 7 python 3
- Network connections to:
  - "ThorlabsMotor" (X-direction) on 134.34.142.60:20006 appendix E.43 on page 362
  - "ThorlabsMotor" (Y-direction) on 134.34.142.60:20007 appendix E.43 on page 362
  - "Laser" on 134.34.142.60:20016 appendix E.17 on page 335
  - "TransCOM" on 134.34.142.60:20015 appendix E.45 on page 364
  - "Kamera" on 134.34.142.62:20005 appendix E.15 on page 333
  - "SourceArb" on 134.34.142.60:20014 appendix E.33 on page 347
  - "Faulhaber" on 134.34.142.62:20004 appendix E.10 on page 330
  - "WiderstandsmessungMaeander" on 134.34.142.60:20024 appendix E.54 on page 371
- Hardware connections to:
  - Zurich Instruments Lock-In Amplifier

This software performs the static thermovoltage measurements using the transient recorder for the data acquisition. During the procedure, opening and closing traces are recorded with the laser turned on and off and the laser focus on both sides of the constriction.
Every trace begins with a picture with the laser at low power to document the position of the focus. Then the laser is set to the required power and the software waits for 5 s to have a stable laser power and a stable thermal distribution inside of the sample. The resistance of the thermometer is measured once and then the measurement is set to the continuous mode to allows to extract the resistance values from the transient recorder data. The data acquisition with the transient recorder is started subsequently and the motor starts to open or close the constriction. The conductance is continuously monitored with the lock-in data and used to check if the contact has been moved to the limit and also the speed of the motor is adjusted to the conductance in order to reduce the "useless" data

which is recorded. The values of the lock-in amplifier are not saved directly, since this would result in two different time axes. Hence the relevant signals of the lock-in amplifier are presented at the outputs and these analog signals are recorded by the transient recorder. When the junction has been moved enough for the conductance to reach the limit, the motor and the data acquisition is stopped, the resistance of the thermometer is measured again, and then another picture of the position of the laser focus is taken at low laser power. This is repeated for every trace until the user stops the measurement. The behaviour of the software is summarized in the following pseudo code.

```
while True:
  for side in ["sensor","empty"]:
    move the focus to the correct side
    for laser state in ["off","on"]:
      for direction in ["opening","closing"]:
        take picture at low laser power
        if laser == "on":
          turn laser on
        else:
          turn laser off
        wait 5s
        measure the thermometer resistance in single mode
        set resistance measurement mode to continuous
        start data acquisition in the transient recorder
        if direction == "opening":
          move motor in the opening direction
          wait until conductance < 0.2 G0
        else:
          move motor in the closing direction
          wait until conductance > 20 G0
        stop the data acquisition
        measure the thermometer resistance in single mode
        take picture at low laser power
```

The software is controlled by the program "MeasurementController" (see appendix E.23 on page 341) and implements the following commands:

- `kill`
  This command stops the measurement and interrupts the current trace. The motor and the data acquisition are stopped.
  The command returns `done` and changes the state to `killing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `killed`.

- `pause`
  This pauses the measurement at the end of a closing trace. This command is designed to be used when manual interact with the setup is necessary (e.g. refilling the coolant).
  The command returns `done` and changes the state to `pausing`. It returns `failed`

when the current state does not allow the change. When executed, the state changes to `paused`.

- `halt`
  This command interrupts the currently active measurement. The motor and the data acquisition are stopped. The trace can be continued in the same direction with a new output file when the measurement is resumed.
  The command returns `done` and changes the state to `halting`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `halted`.

- `continue`
  This resumes the normal operation at the point where the process was interrupted. The command returns `done` and changes the state to `continuing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `running`.

- `start`
  This starts the measurement with the settings shown in the user interface.
  The command returns `done` and changes the state to `starting`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `running`.

- `stuck`
  When the software failed to update the current operation indicating an error, the software state is set to `stuck`. This has no further implications and only serves as an indication to the user.
  The command returns `done` and changes the state to `stuck`. It returns `failed` when the current state does not allow the change.

- `getTimeSinceOperationUpdate`
  This command returns the time in seconds since the last update of the current operation.

- `getOperation`
  This command returns the string of the current operation.

- `getState`
  This command returns the state of the software. Possible values are `continuing`, `finished`, `halted`, `halting`, `killed`, `killing`, `not started`, `paused`, `pausing`, `problem`, `running`, `starting`, or `stuck`.

- `getProgress`
  This measurement has no end, therefore the progress is not defined and it returns `no data`.

- `getElapsed`
  This command returns the elapsed time in seconds since the start excluding any pausing and halting times.
  When the time is not defined it returns `no data`.

- `getRemaining`
  Since the measurement has no end, the remaining time is not defined and it returns `no data`.

## E.41. "Thermospannungsmessung_statisch_V2"

- Server functionality: port 20017
- Computer: 134.34.142.60 Windows 7 python 3
- Network connections to:
  - "ThorlabsMotor" (X-direction) on 134.34.142.60:20006
    appendix E.43 on page 362
  - "ThorlabsMotor" (Y-direction) on 134.34.142.60:20007
    appendix E.43 on page 362
  - "Laser" on 134.34.142.60:20016 appendix E.17 on page 335
  - "LockInAcquisition" on 134.34.142.60:20025 appendix E.22 on page 340
  - "Kamera" on 134.34.142.62:20005 appendix E.15 on page 333
  - "SourceArb" on 134.34.142.60:20014 appendix E.33 on page 347
  - "Faulhaber" on 134.34.142.62:20004 appendix E.10 on page 330
  - "WiderstandsmessungMaeander" on 134.34.142.60:20024
    appendix E.54 on page 371
- Hardware connections to:
  - Zurich Instruments Lock-In Amplifier

This software performs the static thermovoltage measurements using only the lock-in amplifier for the data acquisition. For the dc measurement, the lock-in amplifier is configured for a frequency of 0 Hz. Using only the lock-in amplifier eliminates the problem generated by the input offsets of the transient recorder resulting in wrong conductances. During the procedure, opening and closing traces are recorded with the laser turned on and off and the laser focus on both sides of the constriction.
Every trace begins with a picture with the laser at low power to document the position of the focus. Then the laser is set to the required power and the software waits for 5 s to have a stable laser power and a stable thermal distribution inside of the sample. The resistance of the thermometer is measured once and then the measurement is set to the continuous mode. The data acquisition with the lock-in amplifier is started subsequently and the motor starts to open or close the constriction. The conductance is continuously monitored with the lock-in data and used to check if the contact has been moved to the limit and also the speed of the motor is adjusted to the conductance in order to reduce the "useless" data which is recorded. When the junction has been moved enough for the conductance to reach the limit, the motor and the data acquisition is stopped, the resistance of the thermometer is measured again, and then another picture of the position of the laser focus is taken at low laser power. This is repeated for every trace until the

user stops the measurement.
The behaviour of the software is summarized in the following pseudo code.

```
while True:
  for side in ["sensor","empty"]:
    move the focus to the correct side
    for laser state in ["off","on"]:
      for direction in ["opening","closing"]:
        take picture at low laser power
        if laser == "on":
          turn laser on
        else:
          turn laser off
        wait 5s
        measure the thermometer resistance in single mode
        set resistance measurement mode to continuous
        start data acquisition with the lock−in amplifier
        if direction == "opening":
          move motor in the opening direction
          wait until conductance < 0.2 G0
        else:
          move motor in the closing direction
          wait until conductance > 20 G0
        stop the data acquisition
        measure the thermometer resistance in single mode
        take picture at low laser power
```

The software is controlled by the program "MeasurementController" (see appendix E.23 on page 341) and implements the following commands:

- `kill`
  This command stops the measurement and interrupts the current trace. The motor and the data acquisition are stopped.
  The command returns `done` and changes the state to `killing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `killed`.

- `pause`
  This pauses the measurement at the end of a closing trace. This command is designed to be used when manual interact with the setup is necessary (e.g. refilling the coolant).
  The command returns `done` and changes the state to `pausing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `paused`.

- `halt`
  This command interrupts the currently active measurement. The motor and the data acquisition are stopped. The trace can be continued in the same direction

with a new output file when the measurement is resumed.
The command returns `done` and changes the state to `halting`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `halted`.

- `continue`
  This resumes the normal operation at the point where the process was interrupted.
  The command returns `done` and changes the state to `continuing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `running`.

- `start`
  This starts the measurement with the settings shown in the user interface.
  The command returns `done` and changes the state to `starting`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `running`.

- `stuck`
  When the software failed to update the current operation indicating an error, the software state is set to `stuck`. This has no further implications and only serves as an indication to the user.
  The command returns `done` and changes the state to `stuck`. It returns `failed` when the current state does not allow the change.

- `getTimeSinceOperationUpdate`
  This command returns the time in seconds since the last update of the current operation.

- `getOperation`
  This command returns the string of the current operation.

- `getState`
  This command returns the state of the software. Possible values are `continuing`, `finished`, `halted`, `halting`, `killed`, `killing`, `not started`, `paused`, `pausing`, `problem`, `running`, `starting`, or `stuck`.

- `getProgress`
  This measurement has no end, therefore the progress is not defined and it returns `no data`.

- `getElapsed`
  This command returns the elapsed time in seconds since the start excluding any pausing and halting times.
  When the time is not defined it returns `no data`.

- `getRemaining`
  Since the measurement has no end, the remaining time is not defined and it returns `no data`.

## E.42. “Thermospannungsmessung_V3”

- Server functionality: port 20017
- Computer: 134.34.142.60 Windows 7 python 3
- Network connections to:
  - “Laser” on 134.34.142.60:20016 appendix E.17 on page 335
  - “TransCOM” on 134.34.142.60:20015 appendix E.45 on page 364
  - “Kamera” on 134.34.142.62:20005 appendix E.15 on page 333
  - “SourceArb” on 134.34.142.60:20014 appendix E.33 on page 347
  - “Multimeter” on 134.34.142.60:20018 appendix E.27 on page 343
  - “Faulhaber” on 134.34.142.62:20004 appendix E.10 on page 330
  - “DataCompression_IV” on 134.34.142.61:20023 appendix E.7 on page 329

This software performs the thermovoltage measurement on the samples for the pulsed heating. With an applied dc voltage of 20 mV, the conductance is calculated from the voltage drop across the junction measured by the multimeter across the sense leads. Using this conductance, the junction is moved until the conductance is within the measurement range (0.2 to 20 $G_0$). Depending on how far the conductance is away from this range, the motor speed is adjusted in three steps. Once the conductance is where it is supposed to, the motor is stopped.
After taking a picture of the position of the laser focus, the laser is set to a pulse sequence with a 4 ms pulse every 34 ms. Since the laser power droops during the pulse, the arbitrary waveform is used to compensate this. Synchronized with the laser pulses, the bias voltage is set to a sinusoidal burst that starts at the beginning of every laser pulse. Then 45 s of data are recorded. To save space the data is directly compressed after the acquisition in real time. The compression runs as a background process, thus the measurement can continue. The motor is moved until the conductance differs by more that 2 % from the last measured contact. If the conductance leaves the measurement range, the direction of the motor is changed.
The behaviour of the program is summarized by the following pseudo code.

```
while True:
  apply dc bias of 20 mV
  move motor at appropriate speed
  if conductance outside measurement range:
    change direction
  else
    if conductance differs from last measurement by > 2%:
      stop motor
      take a picture with low laser power
      apply sinusoidal bias voltage
      set laser to 4ms pulses every 34ms
      wait for 5s
```

acquire 45s of data
start the datacompression in the background

The software is controlled by the program "MeasurementController" (see appendix E.23 on page 341) and implements the following commands:

- `kill`
  This command stops the measurement directly, however it may still take some time, if the measurement is currently acquiring data, since this process is not interrupted. The command returns `done` and changes the state to `killing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `killed`.

- `pause`
  This command pauses the measurement during the motor motion or after the data acquisition.
  The command returns `done` and changes the state to `pausing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `paused`.

- `halt`
  This is equivalent to `pause`, except it is also possible to halt the measurement between taking the picture and starting the data acquisition.
  The command returns `done` and changes the state to `halting`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `halted`.

- `continue`
  This resumes the normal operation at the point where the process was interrupted. The command returns `done` and changes the state to `continuing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `running`.

- `start`
  This starts the measurement with the setting given in the user interface.
  The command returns `done` and changes the state to `starting`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `running`.

- `stuck`
  When the software failed to update the current operation indicating an error, the software state is set to `stuck`. This has no further implications and only serves as an indication to the user.
  The command returns `done` and changes the state to `stuck`. It returns `failed` when the current state does not allow the change.

- `getTimeSinceOperationUpdate`
  This command returns the time in seconds since the last update of the current operation.

- `getOperation`
  This command returns the string of the current operation.
- `getState`
  This command returns the state of the software. Possible values are `continuing`, `finished`, `halted`, `halting`, `killed`, `killing`, `not started`, `paused`, `pausing`, `problem`, `running`, `starting`, or `stuck`.
- `getProgress`
  This measurement has no end, therefore the progress is not defined and it returns `no data`.
- `getElapsed`
  This command returns the elapsed time in seconds since the start excluding any pausing and halting times.
  When the time is not defined it returns `no data`.
- `getRemaining`
  Since the measurement has no end, the remaining time is not defined and it returns `no data`.

## E.43. "ThorlabsMotor"

- Server functionality: port 20006 (X-direction) and 20007 (Y-direction)
- Computer: 134.34.142.60 Windows 7 python 2.7
- Hardware connections to:
  - Thorlabs TDC001 SN:83850509 for X direction
  - Thorlabs TDC001 SN:83827673 for Y direction

This software controls the two Thorlabs motors which move the laser focus. It shows and logs their position. The software also provides the possibility to move them to a new position and to start the homing sequence.
The two motors are handled by the same user interface, but internally the motors are separated. This is also why the software uses two different ports for the server function. The software implements the following network commands:

- `getPosition`
  This command returns the position of the motor in mm. When the data is not available it returns `no data`.
- `moveTo␣<Position>`
  This command moves the motor to the position given in mm. It returns `done` when the motion is finished, of `failed` in case of an error.
- `home`
  This command initiates the homing sequence. It returns `done` when the homing procedure has finished.

## E.44. "Training"

- Server functionality: port 20017
- Computer: 134.34.142.60 Windows 7 python 3
- Network connections to:
  - "Faulhaber" on 134.34.142.62:20004 appendix E.10 on page 330
- Hardware connections to:
  - Zurich Instruments Lock-In Amplifier

This software is designed to open and close the constriction repeatedly in order to generate a "nice" arrangement of the atoms at the breaking point. The software behaves identical to the real measurement, but it does not operate the laser and it does not save any data.
When the measurement is running, the constriction is opened until the conductance is below $0.2\,G_0$. Then the software changes directions and closes the junction until a conductance of $20\,G_0$ is reached. This process is repeated until the user stops the program.
The software is controlled by the program "MeasurementController" (see appendix E.23 on page 341) and implements the following commands:

- `kill`
  This stops the programs instantly.
  The command returns `done` and changes the state to `killing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `killed`.

- `pause`
  The software continues the bending of the sample until the end of the closing process. There it will wait.
  The command returns `done` and changes the state to `pausing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `paused`.

- `halt`
  The software interrupts the bending process instantly. The process can later be resumed and the bending continues in the same direction.
  The command returns `done` and changes the state to `halting`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `halted`.

- `continue`
  This resumes the normal operation at the point where the process was interrupted.
  The command returns `done` and changes the state to `continuing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `running`.

- `start`
  This starts the measurement.
  The command returns `done` and changes the state to `starting`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `running`.

- `stuck`
  When the software failed to update the current operation indicating an error, the software state is set to `stuck`. This has no further implications and only serves as an indication to the user.
  The command returns `done` and changes the state to `stuck`. It returns `failed` when the current state does not allow the change.

- `getTimeSinceOperationUpdate`
  This command returns the time in seconds since the last update of the current operation.

- `getOperation`
  This command returns the string of the current operation.

- `getState`
  This command returns the state of the software. Possible values are `continuing`, `finished`, `halted`, `halting`, `killed`, `killing`, `not started`, `paused`, `pausing`, `problem`, `running`, `starting`, or `stuck`.

- `getProgress`
  This measurement has no end, therefore the progress is not defined and it returns `no data`.

- `getElapsed`
  This command returns the elapsed time in seconds since the start excluding any pausing and halting times.
  When the time is not defined it returns `no data`.

- `getRemaining`
  Since the measurement has no end, the remaining time is not defined and it returns `no data`.

## E.45. "TransCOM"

- Server functionality: port 20015

- Computer: 134.34.142.60 Windows 7 python 2.7

- Hardware connections to:
  - Transient recorder API on `134.34.142.61`

This software is only an interface between the transient recorder API and the network based commands of my software. It allows the measurement software to control the transient recorder.
The software implements the following network commands:

- `saveRec␣<Filename>␣<Channels>`
  This command saves the desired channel data into a file with the given name. It returns `no data` while the file is being saved. This will trigger follow up requests. They are also answered by `no data` until the file is saved. Then it returns `done`.

- `getScalarTableADouble␣<Column>␣<Row>`
  This command returns the value in the scalar table A at the given position. It returns `failed` if an error occurs.

- `setSampleRate␣<Value>`
  This command sets the sample rate to the given value in Hz. It returns `done` except if an error occurs. In this case it returns `failed`.

- `setSampleRate␣<Value>`
  This command sets the block size to the given value. It returns `done` except if an error occurs. In this case it returns `failed`.

- `setTriggerDelay␣<Value>`
  This command sets the trigger delay to the given value. It returns `done` except if an error occurs. In this case it returns `failed`.

- `run`
  This command arms the transient recorder. It always returns `done`.

- `stop`
  This command stops the transient recorder. It always returns `done`.

- `trigger`
  This command triggers the armed transient recorder. It always returns `done`.

- `waitForRecording`
  This command waits until the recording is finished. Only then it returns `done`.

- `setToScopeMode`
  This command changes the transient recorder to the scope mode. It always returns `done`.

- `setToContinuousMode`
  This command changes the transient recorder to the continuous mode. It always returns `done`.

## E.46. “Turbopump”

- Server functionality: port 20002
- Computer: 134.34.142.62 Linux python 2.7
- Hardware connections to:
  - Alcatel ATC250 on `ASRL/dev/ttyUSBDelock_renumbered2::INSTR`

This program is used to show and log the operation parameters of the turbopump. The program shows the rotation speed, the temperature, the current mode of operation, and any faults. It has the possibility to start and stop the pump. However, these features are rarely used since the switches on the controller are easily accessible.
The software implements the following network command:

- `getSpeed`
  This command returns the rotation speed of the pump in rpm. When the data is not available it returns `no data`.

## E.47. “Verkabelungscharakterisierung”

- Computer: 134.34.142.60 Windows 7 python 2.7
- Network connections to:
  - “Multiplexer” on 134.34.142.62:20022 appendix E.28 on page 344
- Hardware connections to:
  - Agilent 34401A multimeter on `GPIB::4::INSTR`
  - Keithley 2400 source meter on `GPIB::2::INSTR`

This software measures all possible sourcing and sensing combinations on a test sample, and checks if the results agree with calculated values. This is performed to check the complete setup wiring for irregularities like broken wires or ground contacts.
The multiplexer switches the current source to the different combinations, and the multimeter is switched to the different sensing wires to perform four point resistance measurement with the current supply on and off. These differential measurements allow to calculate the correct resistance even in the presence of offset voltages. Also the offset voltage itself is measured.

## E.48. “Widerstandskalibrierung”

- Computer: 134.34.142.60 Windows 7 python 2.7
- Network connections to:
  - “TemperaturLogger” on 134.34.142.60:20000 appendix E.38 on page 350

  - "TransCOM" on 134.34.142.60:20015 appendix E.45 on page 364
  - "Source" on 134.34.142.60:20014 appendix E.32 on page 346

This software measures the resistances of temperature sensing leads in the samples for the pulsed measurements at different temperatures.
Starting at 86 K, the software increases the setpoints for the sample chamber and the vaporizer in steps of 2 K up to 120 K. At every step, once the temperature has settled, bias voltages of -100, -50, 0, 50, and 100 mV are applied, and the transient recorder acquires data for every voltage.

## E.49. "WiderstandskalibrierungSpezialprobe"

- Computer: 134.34.142.60 Windows 7 python 2.7
- Network connections to:
  - "TemperaturLogger" on 134.34.142.60:20000 appendix E.38 on page 350
  - "Multiplexer" on 134.34.142.62:20022 appendix E.28 on page 344
- Hardware connections to:
  - Agilent 34401A multimeter on `GPIB::4::INSTR`
  - Keithley 2400 source meter on `GPIB::2::INSTR`

This software measures the resistance of the central part of the sample with a four point measurement. To elimintate any influence of thermovoltages, the voltage is measured at source currents of -10, -5, 0, 5, and 10 µA. The temperature of the vaporizer and the sample chamber are logged simultaneously to have the temperature dependence of the resistance. The software is designed to run during the cool down or the warm up phase and was originally meant to measure the resistance of a thin platinum wire on a specific kind of test samples.

## E.50. "WiderstandskalibrierungStatisch"

- Computer: 134.34.142.60 Windows 7 python 2.7
- Network connections to:
  - "TemperaturLogger" on 134.34.142.60:20000 appendix E.38 on page 350
  - "Multiplexer" on 134.34.142.62:20022 appendix E.28 on page 344
  - "WiderstandsmessungMaeander" on 134.34.142.60:20024 appendix E.54 on page 371

This software uses the multiplexer to configure the electronics for a four point measurement on the platinum resistance thermometer on the samples for the static thermovoltage measurements. Then it logs the meander resistance and the corresponding temperatures every 2 s. This program is meant to run during the cool down or warm up procedure, so

that the resistance can be measured as function of the temperature. In the analysis, this data can then be inverted and the temperature of the sample can be derived from the resistance.

## E.51. "WiderstandskalibrierungStatisch_V2"

- Computer: 134.34.142.60 Windows 7 python 2.7
- Network connections to:
  - "TemperaturLogger" on 134.34.142.60:20000 appendix E.38 on page 350
  - "WiderstandsmessungMaeander" on 134.34.142.60:20024 appendix E.54 on page 371

This software relies on the multiplexer being configured for a four point measurement on the platinum resistance thermometer on the samples for the static thermovoltage measurements. The multiplexer connections are not changed to avoid any danger to the sample. The software logs the meander resistance and the corresponding temperatures every 2 s. This program is meant to run during the cool down or warm up procedure, so that the resistance can be measured as function of the temperature. In the analysis, this data can then be inverted and the temperature of the sample can be derived from the resistance.

## E.52. "WiderstandsmessungLaser"

- Computer: 134.34.142.60 Windows 7 python 2.7
- Network connections to:
  - "TemperaturLogger" on 134.34.142.60:20000 appendix E.38 on page 350
  - "Multiplexer" on 134.34.142.62:20022 appendix E.28 on page 344
  - "WiderstandsmessungMaeander" on 134.34.142.60:20024 appendix E.54 on page 371
  - "Laser" on 134.34.142.60:20016 appendix E.17 on page 335

This software uses the multiplexer to configure the electronics for a four point measurement on the platinum resistance thermometer on the samples for the static thermovoltage measurements. The resistance is measured as a function of temperature at different laser modulation voltages of 0, 1, 2, 3, and 4 V. After setting the new voltage, the software waits for 5 s to reach sample conditions for the laser and the sample. To measure the temperature dependence, this software is run during the cool down or warm up phases.

## E.53. "Widerstandsmessung_Laserwechsel"

- Server functionality: port 20017
- Computer: 134.34.142.60 Windows 7 python 3

- Network connections to:
  - "ThorlabsMotor" (X-direction) on 134.34.142.60:20006 appendix E.43 on page 362
  - "ThorlabsMotor" (Y-direction) on 134.34.142.60:20007 appendix E.43 on page 362
  - "TemperaturLogger" on 134.34.142.60:20000 appendix E.38 on page 350
  - "Laser" on 134.34.142.60:20016 appendix E.17 on page 335
  - "Kamera" on 134.34.142.62:20005 appendix E.15 on page 333
  - "TemperaturLogger" on 134.34.142.60:20000 appendix E.38 on page 350
  - "WiderstandsmessungMaeander" on 134.34.142.60:20024 appendix E.54 on page 371

This software measures the resistance of the platinum resistance thermometer integrated in the samples without the laser or with the laser illuminating either side of the constriction. The measurement is supposed to be performed during the cool down of the cryostat in order to measure the temperature dependence of the resistance.
The program behaves as indicated by the following pseudo code:

```
for side in [sensor,empty]:
  move laser focus to the side
  for laserpower in [off,on]:
    this is the waiting point for pause or halt
    take picture with low laser power
    set laserpower
    wait 10 seconds
    measure and store resistance
    store temperatures of the setup
    take picture with low laser power
```

The software is controlled by the program "MeasurementController" (see appendix E.23 on page 341) and implements the following commands:

- `kill`
  This aborts the measurement. The currently measured point is completed including the end picture.
  The command returns `done` and changes the state to `killing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `killed`.

- `pause`
  This pauses the measurement before the next measurement.
  The command returns `done` and changes the state to `pausing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `paused`.

- `halt`
  This is equivalent to `pause`.
  The command returns `done` and changes the state to `halting`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `halted`.

- `continue`
  This resumes the normal operation at the point where the process was interrupted.
  The command returns `done` and changes the state to `continuing`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `running`.

- `start`
  This starts the measurement.
  The command returns `done` and changes the state to `starting`. It returns `failed` when the current state does not allow the change. When executed, the state changes to `running`.

- `stuck`
  When the software failed to update the current operation indicating an error, the software state is set to `stuck`. This has no further implications and only serves as an indication to the user.
  The command returns `done` and changes the state to `stuck`. It returns `failed` when the current state does not allow the change.

- `getTimeSinceOperationUpdate`
  This command returns the time in seconds since the last update of the current operation.

- `getOperation`
  This command returns the string of the current operation.

- `getState`
  This command returns the state of the software. Possible values are `continuing`, `finished`, `halted`, `halting`, `killed`, `killing`, `not started`, `paused`, `pausing`, `problem`, `running`, `starting`, or `stuck`.

- `getProgress`
  This measurement has no end, therefore the progress is not defined and it returns `no data`.

- `getElapsed`
  This command returns the elapsed time in seconds since the start excluding any pausing and halting times.
  When the time is not defined it returns `no data`.

- `getRemaining`
  Since the measurement has no end, the remaining time is not defined and it returns `no data`.

## E.54. "WiderstandsmessungMaeander"

- Server functionality: port 20024
- Computer: 134.34.142.60 Windows 7 python 2.7
- Hardware connections to:
  - Keithley 2400 source meter on `GPIB::2::INSTR`

This software is an interface to the Keithley 2400 source meter. The software performs offset corrected four point resistance measurements with an excitation current of 5 µA and a voltage limit of 1 V.
The software implements the following network commands:

- `getReading`
  This command performs a resistance measurement. This sets the source meter to the single measurement mode. It returns the value in Ω or `no data` if the data is not available.
- `getContinuous`
  This command checks if the source meter is in continuous measurement mode. If so, it returns `True`, otherwise `False`.
- `setContinuous␣<true|on|enable>`
  This commands sets the source meter to the continuous mode. It returns `done`.

# F. Naming conventions

## Measurements

The individual measurements are identified by a unique timestamp. This timestamp has the format "YYYYMMDDhhmmss" with "YYYY" being the year, "MM" the month, "DD" the day, "hh" the hour in 24-h format, "mm" the minute, and "ss" the second. The date is given in local time, thus problems might occur during the transitions to daylight-saving time. The format is designed such that the alphabetic sorting will produce the same results as the sorting by time. To indicate which type of measurement was performed, the measurement data stays in the folder of the program that recorded the data. The exceptions to this are the major data acquisition programs where the whole dataset is gathered by multiple programs. All this data is combined into one folder per measurement. The folders are labeled "YYYYMMDDhhmmss-type" where type is set according to the following list

- rcal:
  The data in this type of measurement was recorded by the "Widerstandskalibrierung" software (see appendix E.48 on page 366). The data contains the resistance of the sense leads of a sample designed for the pulsed measurements as function of the sample chamber temperature.

- rmap:
  The data was recorded by the "Kartenaufzeichnen_V2" software (see appendix E.16 on page 333). The data contains the spatially resolved resistance change of the sense leads of the samples used in the pulsed measurements.

- rmap-special:
  The data was recorded by the "Kartenaufzeichnen_V2" software (see appendix E.16 on page 333). The data contains the spatially resolved resistance change of the narrow gold structure present in the test samples used in these measurements.

- tv:
  The data was recorded by the "Thermospannungsmessung_V3" software (see appendix E.42 on page 360). The data contains the thermopower for each heating pulse at different conductances.

- training:
  The data was recorded by the "Training" software (see appendix E.44 on page 363).

- lmap:
  The data was recorded by the "Widerstandsmessung_Laserwechsel" software (see appendix E.53 on page 368). The data contains the resistance of the thermometer

integrated into the sample as function of the position of the heating laser spot at different laser powers.

- rlaser:
  The data was recorded by the "Widerstandsmessung_Laserwechsel" software (see appendix E.53 on page 368). The data contains the resistance of the thermometer integrated into the sample as function of the sample chamber temperature with and without the laser heating on either side of the constriction.

- tvstat:
  The data was recorded by the "Thermospannungsmessung_Statisch" software (see appendix E.40 on page 354). The data contains the thermovoltage of a gold junction as function of the conductance with and without the laser heating on either side of the constriction.

- tvstat2:
  The data was recorded by the "Thermospannungsmessung_Statisch_V2" software (see appendix E.41 on page 357). The data contains the thermovoltage of a gold junction as function of the conductance with and without the laser heating on either side of the constriction.

- tvmol:
  The data was recorded by the "Thermospannungsmessung_Molekuele" software (see appendix E.39 on page 351). The data contains the thermovoltage of a molecular junction as function of the conductance with and without the laser heating on either side of the constriction.

## Simulations

The simulations are labeled according to the scheme "VX_YY" with "X" being a capital letter starting at "A" counting the major versions, and "YY" a two digit number for the minor versions. Particular results might also include a timestamp of the beginning of the calculation.

## Samples

The samples are (mostly) named according to the following rules:

- B????K: The naming of the samples created by Bastian Kopp. The samples are made for thermovoltage measurements on atomic gold contacts using the additional sensing wires as resistive thermometers.
- T????M: My version of the samples in the aforementioned geometry.
- T????S: Samples designed for static thermovoltage measurements on atomic gold contacts with the additional resistive thermometer on top of one side of the constriction.
- T????C: Samples designed for static thermovoltage measurements on molecular contacts.
- T????T: Test samples for various purposes.
- T????P: Samples with plasmonic gratings.