# Development of a parallelizable QCM-D array for the mass spectrometric analysis of proteins

# Master thesis

Master of Process Egnineering
University of Applied Sciences Offenburg
University of Warmia and Mazury Olsztyn

**Siegfried Hohmann, B. Sc.**

2015-04-30

Supervisors:
Prof. Dr. ner. nat. Bernd Spangenberg
Dr. Gerald Brenner-Weiß
Dipl. Ing. Frank Kirschhöfer

## Abstract

Quarz crystal microbalances allow the monitoring of the adsorption process of mass from a liquid to their surface. The adsorbed mass can be analysed regarding to its protein content using mass spectromety. To ensure the protein identification the results of several measurements can be combined. A high content QCM-D array was developed to allow up to ten measurements parallel. The samples can be routed inside the array distributing one sample to several chips. The fluidic parts were prototyped using 3D printing. The assembled array was tight and the sample routing function could be demonstrated. A temperature controller was developed and implemented. The parameters for the PID controller were determined and the controller was shown to be able to keep the temperature constant over long time with high accuracy.

KEYWORDS: QCM-D, Protein, Mass spectrometry, High content measurement, QCM-D Array, 3D printing, Temperature control, Arduino

# Contents

# 1 Introduction

## 1.1 Motivation

The amount of substance adsorbing from a liquid phase to a defined solid surface can be monitored using quarz crystal microbalances. They consist of a cylindrical quarz crystal with electrodes on each side. The inverse piezoelectric effect causes the crystal to be deformed, if a voltage is applyed to it. The resonance frequency depends on the mass of the crystal and changes if mass from the liquid phase adsorbs onto the crystal surface. Additional information about the viscoelastic properties of the adsorbed layer can be obtained by monitoring the decay of the oscillation of the crystal, after its excitation is stopped. This is called quarz crystal microbalance with dissipation monitoring (QCM-D) (O , 1999; Voinova et al., 1999).

If a complex sample is adsorbed onto the chip surface the composition of the adsorbed layer also depends on the chip surface. To determine the composition of the layer on the surface regarding to its protein content, QCM-D coupled with mass spectrometry can be used. The adsorbed proteins are digested on the chip and the resulting peptides are enriched chromatographically. Matrix assisted laser desorption ionization time of flight mass spectrometry (MALDI-ToF MS) allows the detection of the peptide masses with an accuracy allowing to identify the proteins the peptides result from. The sample is mixed with a matrix, that absorbes the energy at the wavelenght of the laser. In a vacuum this allows the desorption of the peptides into the gas phase without their destruction. Adding tetrafluoroacetic acid (TFA) to the sample promotes the ionization of the peptides by the addition of a proton. The ionizized peptides are accelerated into a flight tube and the time, needed to pass it, is determined. The flight time depends on the mass of the peptide and allows to determine it (Perkins et al., 1999; Kirschhöfer et al., 2013).

By comparing the measured masses with the masses of theoretical tryptic digests, the proteins the peptides originate from can be determined. To ensure the correct identification the peptide masses can be further fragmented using a higher laser energy or a collision cell. The comparison of the resulting spectra with theoretical fragmentations of the expected peptide sequences allows to verify their correct identification. To reduce the complexity of the mass spectra sets of at least three chips sampled with indipendent samples are used in this method (Hohmann et al., 2014, 2015).

## 1.2 Aim of the work

Commercially available QCM-D systems only allow up to six parallel measurements. To compare e.g. the adsorption of one sample onto three different chip surfaces three measurements have to be done after another. A high content QCM-D array would allow to perform all measurements parallel. To measure the same sample on three chips with the same surface the possibility to connect them serially would also be usefull.

Aim of the work was to develop a parallelizable QCM-D array, that allows to measure the adsorption of substances from up to ten liquid samples to up to ten QCM-D chips. Also a flexible sample routing allowing the distribution of one sample to more than one chip should be possible.

## 1.3 Structure of the thesis

A 3D model of a parallelizable QCM-D array was developed. The fluidic parts were prototyped using a 3D printer. Copper wires were plated with gold to providing the electrical contacts to the chip. Printed three way valves were developed to route the samples inside the array. The sample routing and tightness of the fluidic was tested using different colored samples. Cooling channels at the bottom of the array allow to keep the sample at a constant temperature. To control the temperature a digital temperature sensor with a PID controller was developed. After the circuit was tested on a breadboard a printed circuit board was developed and manufactured. The parameters of the PID controller were determined using empirical rules applyed to the jump response of the control system. The response of the system to jumps in the setpoint were recorded and the long time stability of the control process was examined.

# 2 Materials and Methods

## 2.1 Chemicals, consumables and devices

A list of the used chemicals and their manufacturers can be found in supplement A. The used consumalbes are are listed in supplement B. Supplement D contains a list of the used devices.

## 2.2 Components

A complete list of all used components can be found in supplement C. In the following sections selected non standard components are described in more detail.

### 2.2.1 QCM-D Sensors

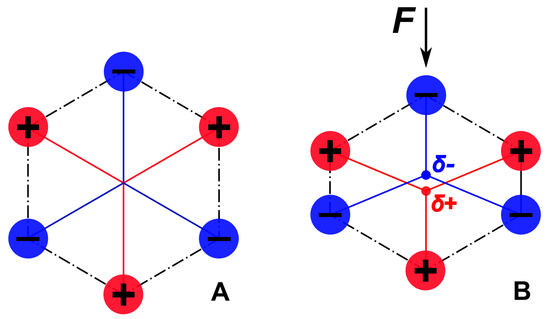


Figure 1: Piezo effect

Ion crystals consist of positive and negative charged ions. These are arranged in a regular three dimensional pattern. The pattern can be reduced to a single minimal cell, that describes the whole crystal stucture, called elementary cell. Each elementary cell is inert to at least one symetric operation, resulting in a net charge of zero at the macroscopic crystal surfaces. An axis in the elementary cell, that is not identical with its rotation about 180 degree around any axis perpendicular to itself, is called a polar axis. Figure 1 shows the elementary cell of a crystal with three polar axes and without a centre of symmetry. Applying a force to a crystal deformes it reversibly unless the force exceeds its structural limits. Applying the force nonperpendicular to one of the polar axes of the crystal in figure 1 **A** the elastic deformation will also cause a dislocation of the mass centers of the positive and negative charged ions (**B**). Summed up over the elementary cells of a macroscopic crystal, that dislocation can be measured as voltage. This is called the piezoelectric effect. The inverse piezoelectric effect describes a deformation of the crystal, when a voltage is applied to its surface (Auld, 1973; Reichl and Ahlers, 1989).

To deform a crystal using the inverse piezoelectric effect and to record the effects of the piezoelectric effect while the crystal is swinging back into its undeformed state is the principle of the quartz crystal microbalance with dissipation monitoring (QCM-D). The resonance frequency of a crystal is dependent to its mass. By measuring the resonance frequency (QCM) or the response to a pulse (QCM-D) of a crystal, changes in its mass can be determined. If one side of the crystal is in contact with a liquid flow, adsorptions onto its surface can be detected. Due to the mass of the adsorbed substances leading to the change in the resonance frequency these types of sensors are called mass sensitive sensors (O , 1999; Voinova et al., 1999).

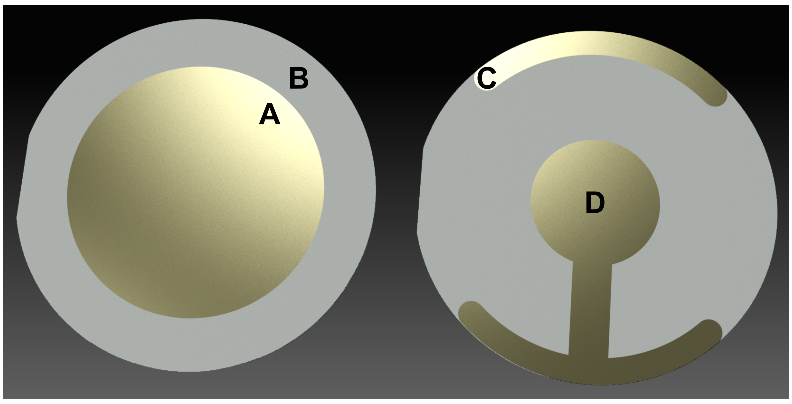



Figure 2: QCM-D chip; A: Sample electrode, B: AT cut quartz substrate, C: Wrap around of the sample electrode, D: Signal electrode

Figure 2 shows a QCM-D sensor based on an AT-cut quarz crystal substrate (**B**). The large circular electrode on one side of the chip (**A**) is in contact with the sample whilest a measurement. To simplify the handling of fluidic probes and electrical connections, the sample electrode is wrapped around to the other side of the chip (**C**). The other electrode (**D**) carries the electrical signal as it is suitable to share ground with the liquids and all sensors. The sample electrode can be modified in a variety of ways allowing it to analyse the adsorption onto different surfaces.

The used QCM-D sensors have a diameter of 14 mm and a thicknes of 0.3 mm. The active surface in contact with the sample is about 80 mm$^2$. Their base frequency is 4.95 MHz.

### 2.2.2 Peltier elements

The diffusion velocity of the main carriers of charge in a metal of semiconductor depends on temperature. If a semiconductor is heated on one side, the charge carriers diffuse to the cold side. In case of a negative doped semiconductor electrons are the main carriers of charge and the cold side becomes negatively charged. Defect electrons, like the main carriers of charge in positive doped semiconductors, lead to a positive charge at the cold side. This is called the Seebeck effect. Flows a current through a semiconductor the reverse effect leads to a temperature gradient in direction of the current flow. This is called the peltier effect.

A peltier element consists of negative and positive charged semiconductors connected serial between two ceramic plates as shown in figure 3. The direction of the current flow through the element determines the direction of the heat transport. This allows to use peltier elements as well for heating as for cooling an object connected to one of its sides. Connecting a heat sink and a fan to the other side helps to increase the heat transport. As the semiconductors heat up by thermal losses the efficiency of the cooling process is lower than that of the heating process (Riffat and Ma, 2003).

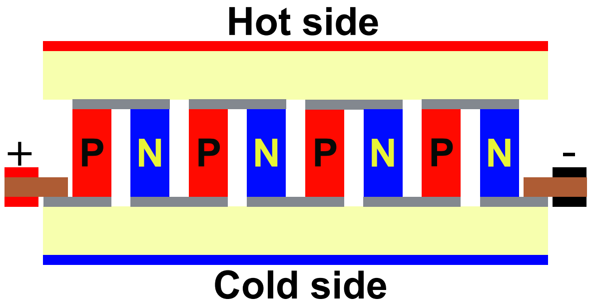



Figure 3: Peltier element

The peltier elements used in this work can be supplyed with a voltage up to 15.4 V drawing a current of 4.6 A. It consists of 127 semiconductor elements between two quadratic ceramic plates with a size of 39.5 x 39.5 mm The possible temperature difference between the hot and the cold side is 60 K and their heat pumpint power is up to 41 W (NTS electronic and components GmbH, 2014).

### 2.2.3 Motor driver

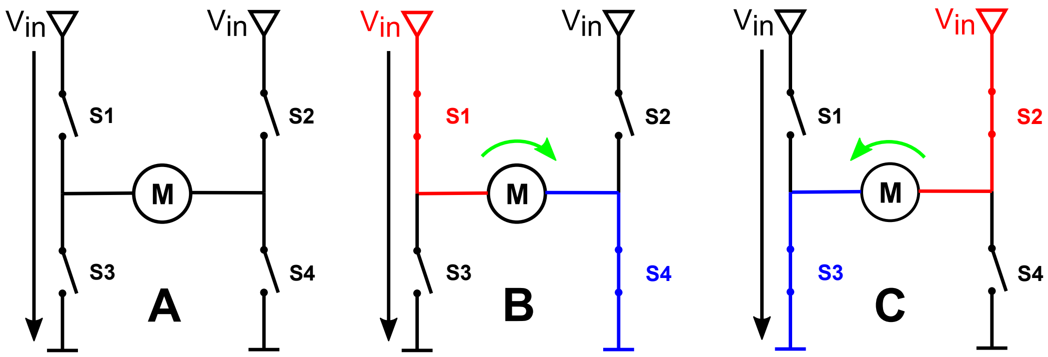



Figure 4: Function of a H bridge

To control the rotation of a motor its speed and its direction is important. The direction can be changed by reversing the polarity of the voltage applyed to the motor. To archive that a H bridge as shown in figure 4 can be used. It consists of four switches S1 to S4. S1 and S2 are connected to the positive supply voltage $V_{in}$, S3 and S4 to ground. The motor is connected with S1 and S3 on its one junction and with S2 and S4 on its other one (**A**). The resulting circuit looks like the capital letter H and is therfore called H bridge. By activating the switches S1 and S4, the left connector of the motor is connected to $V_{in}$ and the right one to ground, resulting in a clockwise rotation (**B**). By activating the switches S2 and S3, the left terminal is connected to ground and the right one to $V_{in}$, resulting in a counterclockwise rotation (**C**). Activating S1 and S2 or S3 and S4 can be used to break the motor as its coils are shorted. Activating S1 and S3 or S2 and S4 should be avoided as it shorts the supply voltage to ground (Tieze and Schenk, 1990).

In this work the integrated motor driver L6203 from SGS-Thomson is used to control the amount and the direction of the current flowing through peltier elements. It containes a H bridge build up by DMOS power transistors as switches combined with a logit circuit. It has two TTL compatible digital inputs. One toggles the transistors used as switches S1 and S3, the other one these of S2 and S4 in a way that either S1 or S3 are activated by the one and either S2 or S4 by the other input. This avoids the possibility of shortening the supply voltage to ground as described above. It allows a supply voltage of up to 48 V, can drive a current of up to 4 A and can be operated at frequencys up to 100 kHz (SGS-THOMSON Microelectronics, 1997).

### 2.2.4 Digital temperature sensor

To measure the array temperature the digital temperature sensor TSIC$^{TM}$ 506 is used. It measures the temperature in a range from -10 °C to +60 °C with an accuracy of 0.1 K and a resolution of 0.034 K and is calibrated by the manufacor. The 11 bit temperature value is transmitted serial and can easily be read by a microprocessor (Thermo Technik GmbH, 2014).

### 2.2.5 Seven segment driver

A seven segment display unit consists of 8 elements, which can be turned on and off seperately and a common connection for the current backflow. The numbers from 0 to 9 and a decimal point can be displayed in a good readability using the seven number and the decimal point segment. To control each segment of a unit, 8 digital signal lines are required without a driver. Using 3 digits it would be even 24 one. The human eye is based on photochemical reactions and their processing making it slow compared to the timescales of a microprocessor. The common connections of the displays are used for a time sharing of the complete information between the digits. If the frequency of the next digit position being allowed to display its content is fast enough, the human eye can see no difference between the digits being all steady on except of a loss of brightness. The state of a display unit fits into one byte. Wired up the state of a display unit is transferred parallel with eight lines. This allows the defined switching between the data for each display unit while iterating over the common connectors. Because digital output lines are mostly rare there are 7 segment drivers managing all the multiplexing and communicating serial to the user. The wanted values for the digits of the display are transferred serially to the driver. The input is read into an internal memory, which is used as value for the digits until changes are recieved. To be able to cascade these drivers to display more numbers than one alone could handle an additional carriage select signal line can be used. Using it, the master device can select by that line, on which display driver to update the values next (Tieze and Schenk, 1990). The MAX7221 7 segment driver from MAXIM was used in this work. It can drive up to eight digits and allows further cascading. The data is transferred serial using the SPI protocol. It allows brightness control by pulse width modulation. Only one resistor is needed to set the current for the segments (MAXIM, 1997).

### 2.2.6 Microcontroller board

The arduino Micro microcontroller board was used for the PID controller. It is based on an ATmega32u4 microcontroller with 16 MHz clock and 2.5 kB SRAM. It provides 20 digital IO pins of which 7 can used for pulse width modulation (PWM) and 12 as analog input channels. It provides 32 kB flash memory for the executable code of which 4 kB are used for the boot loader. 1 kB EEPROM can be used for the nonvolatile storage of data. It has an integrated USB 2.0 port, that is used to program the controller. It can also be used to communicate with a PC to transmit data (Arduino LLC, 2015; Atmel Corporation, 2014).

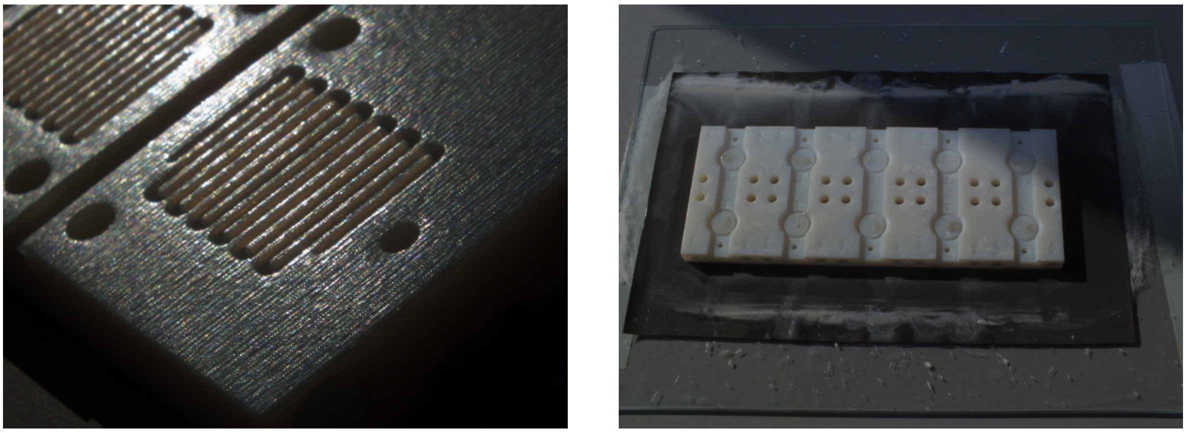
## 2.3 Software

This document was typesetted with LaTeXusing TeXnicCenter v. 1.0. References were included using BIBTeX. Text was edited using Notepad++ v. 6.7.5. Calulations were performed with Matlab R2014a and Octave v. 8.3.2-5. To plot diagrams Microsoft Excel v. 14.0.6129.5000 was used. Technical drawings were made with AutoCAD 2015 SP2. 3D models were produced using Inventor 2015 and AutoCAD 2015 SP2. Arduino IDE v. 1.6.1 was used to program the microcontroller. Code::Blocks v. 13.12 was used for C programming on the PC. KiCAD 2013-07-07 BZR 4022 stable was used to draw circuit diagrams to create component footprints and to develop PCB layouts. Pictures were rendered using Silkypix RAW File Converter v. 3.2.2.0 and edited using Paint Shop Pro v. 5.0 and Gimp v. 2.8.6. Vector graphics were produced and edited using Incscape 0.91.

## 2.4 3D Printing

The fluidic parts and ventiles were produced using the Object Eden260V 3D printer from Stratsys and the polymer Vero White. The printer allows object sizes up to 255 x 252 x 200 mm with a horizontal layer size of 16 µm. The printing head moves with a resolution of 600 dpi at the X and Y axis and with 1600 dpi at the Z axis. To print hollow structures a supporting material is used. It can be removed after the printing process using sodiumhydroxide solution. The printer has eight printing heads for the object and the supporting structure. The basic material is a solution from acrylate monomers and oligomers and is applyed in small droplets, like in an inkjet printer. The liquid is then polymerized using UV light (Stratasys Ltd., 2015).

## 2.5 Modifications of printed parts



(a) Surface roughness of the printed fluidic part

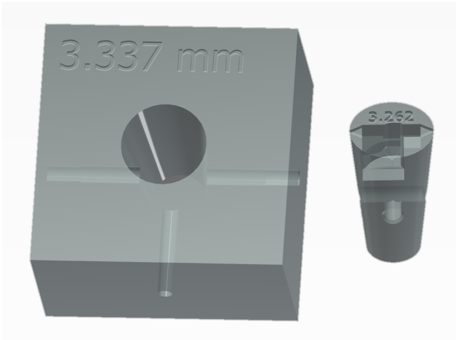
(b) Wet sanding process

Figure 5: Surface roughness caused by the printing process

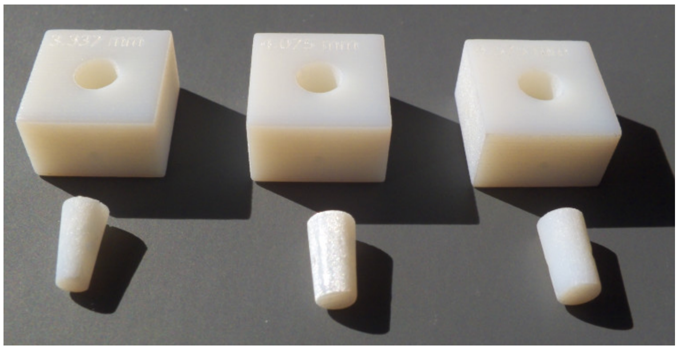
The surface of 3D printed parts shows a rough structure as shown in figure 5 (a). To be able to bond plane glass sheets onto its side containing the cooling channels, the surface was wet sanded. To archive a flat surface sand paper was glued onto a glass sheet and the printed part was moved on its wetted surface in an eight-shaped path.

In two steps threads for the fittings were cutted into the printed material. Therefore a hollow structure in the diameter of the core drill was printed. To correct tolerance errors in the position of the threads in the cooling plate, a round file and a scalpel were used. To correct the deepnes of the structures to insert the temperature sensor a sperical cutter mounted onto a Dremel rotary tool was used.

## 2.6 Valve development using test bodys




(a) 3D model of the test body and the ventile



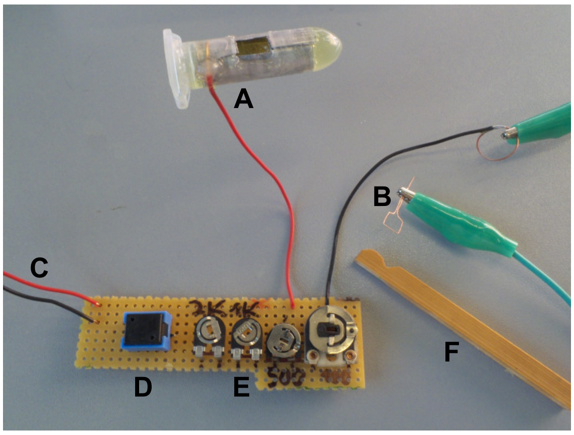
(b) Test bodys with different conical angels

Figure 6: 3D model of the test body, the ventiles and test bodys with different conical angles

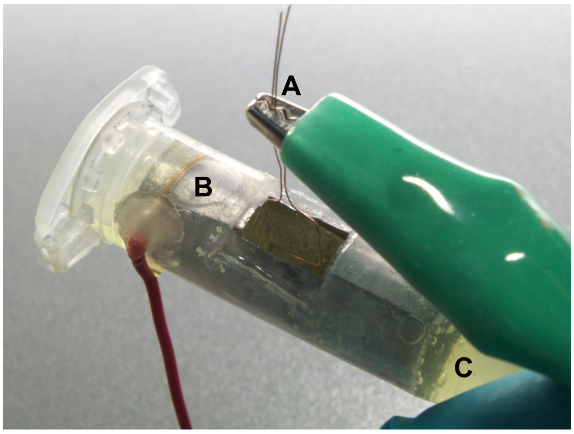
The designed three way valves are conically shaped and can be removed by a hook, that is inserted and turned about 90°. To determine the best angle of the conical shape three versions with different angles were printed together with corresponding test bodys. The test bodys contain three channels and a hollow shape to put in the ventile as shown in figure 6 (a). The ventiles were pressed into the test bodys and a needle with a syringe was put into the channels of the test body. Water was pressed through every channel of the ventile to test its function. To determine if they close tight against the environment the test body with the ventiles were submerged in water and air was pressed into the channel while the ventile was in a position blocking it.

## 2.7 Galvanization

The galvanization setup developed to plate the wires, allowing the electrical connection to the QCM-D chips by being wrapped around the sealing rings is shown in Fig. 7 (a). The galvanization chamber (**A**) is filled with a manufacturers electrolyte solution containing gold ions. The used wire is shaped by a wire shaping tool (**F**) and then submerged into the electrolyte in the galvanization chamber as shown in 7 (b). The circuit board is connected to a 3 V DC power supply by the wires (**C**). The potentiometers (**E**) allow the adjustment of the current while the plating process in a wide range of intensitys and current per area. The pushbutton (**B**) can be used to define the plating duration more exactly than disassembling a hole plating setup.




(a) Galvanization setup; A: Galvanization chamber, B: Target, C: Power supply, D: Pushbutton, E: Current control, F: Wire shaping tool




(b) Galvanization chamber; A: Wire as cathode, B: Round shaped metal sheet anode, C: Electrolyte solution

Figure 7: Galvanization circuit and chamber

## 2.8 UV bonding

After the sanding process the printed fluidic part was sonicated in water several times. Remaining water was removed using nitrogen and the part was allowed to dry for an hour. The UV bonder was allowed to reach the room temperature in 30 minutes. It was applyed to the printed part using a 1 mL syringe with a needle. The needle allowed to apply the bonder to the dividers of the cooling channels. The printed fluidic part with the applyed bonder can be seen in figure 8 (a). Glass sheets were cut to cover 2 or 1 of the cells. The sheets were sanded to remove sharp edges and fixed together using tranparent tape. Then they were put onto the prepared fluidic part and exposed to the readiation of a UV lamp in the wavelenght range the bonder adsorbs UV light best for about half an hour as seen in figure 8 (b). Remaining bonder on the sides was removed using acetone (DELO Industrie Klebstoffe, 2014).

(a) Application of the UV bonder

(b) UV gluing

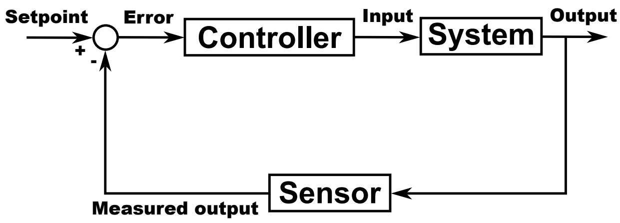
Figure 8: Bonding between the wet sanded fluidic part and glass plates using UV light

## 2.9 Printed circuit board etching

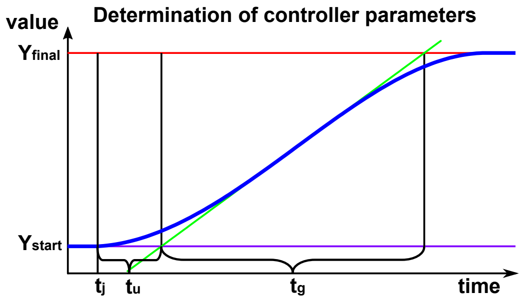
The designed layout of the printed circuit board for the temperature controller was printed onto transparency film, using an inkjet printer. The foil was fixed onto the copper side of a circuit board coated with photoresist and exposed to sunlight and a UV lamp. The photoresist exposed to the light was removed using sodiumhydroxide solution with a concentraion of 10 g/L. The board was etched using sodium persulfate solution in a concentration of 120 g/L at a temperature of about 60°C. The glass plate used to keep the foil plane adsorbed most of the UV light leading to bad results. Finally the PCB was produced at the Institute for Data Processing and Electronics.

## 2.10 Determination of controller parameters

The principle of a control loop is shown in figure 9 (a). Changes in the system input lead to changes in the system output. The system input could e.g. be the position of a valve. Changes in its position cause changes in the flow though the valve, which is the system output. The system output is measured by a sensor. The difference between the measured system output and the wanted setpoint is built and fed as input into the controller. Depending on the error and its behaviour over the time the controller output changes. The controller output is coupled to the system input, closing the control loop.




(a) Principle of a control loop




(b) Typical jump response of a control system

Figure 9: Principle of a control loop and typical jump response

The most commonly used controller type is the PID controller. PID is an acronym for proportional, integral and derivative. Its behaviour is determined by three values: (i) its proportional gain $K_p$, its integral constant $K_i$ and its derivative constant $K_d$. The proportional gain determines the amout of the controller output, that directly depends on the input. The integral constant determines the amout of the error, that is summed up over the time and added to the controller output. The derivative constant determines the amount the controller reacts to the speed of changes in the error (Unbehauen, 1989a).

The controller parameters can be determined by a mathematical model of the control system. This is mostly done for easy control systems. If the number of influencing energy reservoirs and their interaction is to complex for a mathematical modelling, empirical rules can be used to determine the parameters. Figure 9 (b) shows the typical respond of a control system to a jump in its input at the time $t_j$ over the time. $Y_{start}$ is the value of the system to be controled in its stable state at the time of the input jump and $Y_{final}$ is the final state the value approaches to after the input jump. The difference between the time of the input jump and the section between the first inflexion tangent and the start value is called $t_u$. The difference between $t_u$ and the intersection of the tangent with the final value $Y_{final}$ is called $t_g$ .

The empirical rules by Chien, Hrones and Reswick shown in table 1 were used to determine the controller parameters. The jump response of the system was recorded three times for heating and for cooling. For heating an input jump of 50 PWM was used, because it is very efficient. For cooling its complete intensity of 255 PWM was used.

The jump responses were recorded via the USB interface and the comma seperated files were imported into Matlab to determine the inflexion point. Listing 1 shows the performed calculations. The variable $maxC1$ contains the start temperature, $minC1$ the final temperature in steady state, $jumpTimeC$ the time point at which the input jump was applyed and $jumpC$ the amount the input was changed at $jumpTimeC$. The system gain is the ratio between the

| Parameter | Control response | Disturbance response |
|---|---|---|
| $T_n$ | $2.4 \cdot T_u$ | $T_g$ |
| $T_v$ | $0.42 \cdot T_u$ | $0.5 \cdot T_u$ |
| $K_p$ | $0.95 \cdot \frac{T_g}{T_u \cdot K_s}$ | $0.6 \cdot \frac{T_g}{T_u \cdot K_s}$ |
| $K_i$ | $\frac{K_p}{T_v}$ | $\frac{K_p}{T_v}$ |
| $K_d$ | $K_p \cdot T_v$ | $K_p \cdot T_v$ |

Table 1: Rules by Chien, Hrones and Reswick for the determination of PID controller parameters without overswing

amount the temperature changed $diffC1$ and the amount the input was changed. To determine the inflexion point a polynom $polyC1$ with order 10 was fitted to the measured data imported into the variables $TimeC1$ and $TempC1$. Its first and second derivative $polyC1d$ and $polyC1dd$ were determined. The zero points $zeroC1dd$ of the second derivative were determined and the real one at the fitting time point $xC1$ was selected manually. The gain of the inflexion tangent $mC1$ is the value of the first derivative of the polynom at the time $xC1$. The time points $TuC1$ and $TgC1$ result from the intersections between the tangent and the start and final temperature as shown in figure 32. The rules by Chien, Hrones and Reswick for control response without overshoot were finally applyed to determine the controller gain $KpC1$, the integral constant $KiC1$ and the derivative constant $KdC1$ (Unbehauen, 1989a,b; Korsane et al., 2014). The jump response was recorded for heating and cooling three times and the arithmetic avarage of the resulting constants were used for the controller.

```
KsC1 = diffC1 / jumpC
polyC1 = polyfit(TimeC1,TempC1,10)
polyC1d = polyder(polyC1)
polyC1dd = polyder(polyC1d)
zeroC1dd = roots(polyC1dd)
xC1 = zeroC1dd(7)
yC1 = polyval(polyC1,xC1)
mC1 = polyval(polyC1d,zeroC1dd(7))
TuC1 = ((maxC1 - yC1) / mC1) + xC1 - jumpTimeC
TgC1 = ((minC1 - yC1) / mC1) + xC1 - jumpTimeC - TuC1
TnC1 = TgC1
TvC1 = 0.5 * TuC1
KpC1 = (0.6 * TgC1) / (TuC1 * KsC1)
KiC1 = KpC1 / TnC1
KdC1 = KpC1 * TvC1
```

Listing 1: Matlab code used to determine the controller parameters

## 2.11 Calibration of the current measurement

The shunt resistors were disconnected from ground to calibrate the current measurement . A defined current in the range from 0.5 A to 3.5 A was driven through the resistors using the current limitation function of the laboratory power supply. The current was additionally measured with a multimeter. The readings from the analog to digital converters (ADCs) were recorded. Assuming a linear behavior between the current and the measurend voltage at the analog input a linear regression was performed. The gain was used to calculate the current flowing through the resistor and thereby through the peltier elements (Felderhoff, 1990).

## 2.12 Calibration of the room temperature sensor

A negative temperature coefficient resistor (NTC) was used to estimate the room temperature and thereby adjust the controller parameters to weather the main tast is heating or cooling. To calibrate the sensor it was fixed on the cooling plate next to the digital temperature sensor. The temperature was set to values between 20 °C and 40 °C in steps of 1 K and the voltage resulting in the voltage divider between the NTC and a resistor was determined. As the readings showed a highly linear respond to the temperature the slope of the linear regression through the measured points was determined and used to calculate the room temperature from the readings of the ADC (Herold, 1993; Schrüfer, 1984).

# 3 Results

## 3.1 QCM-D Array

### 3.1.1 Concept

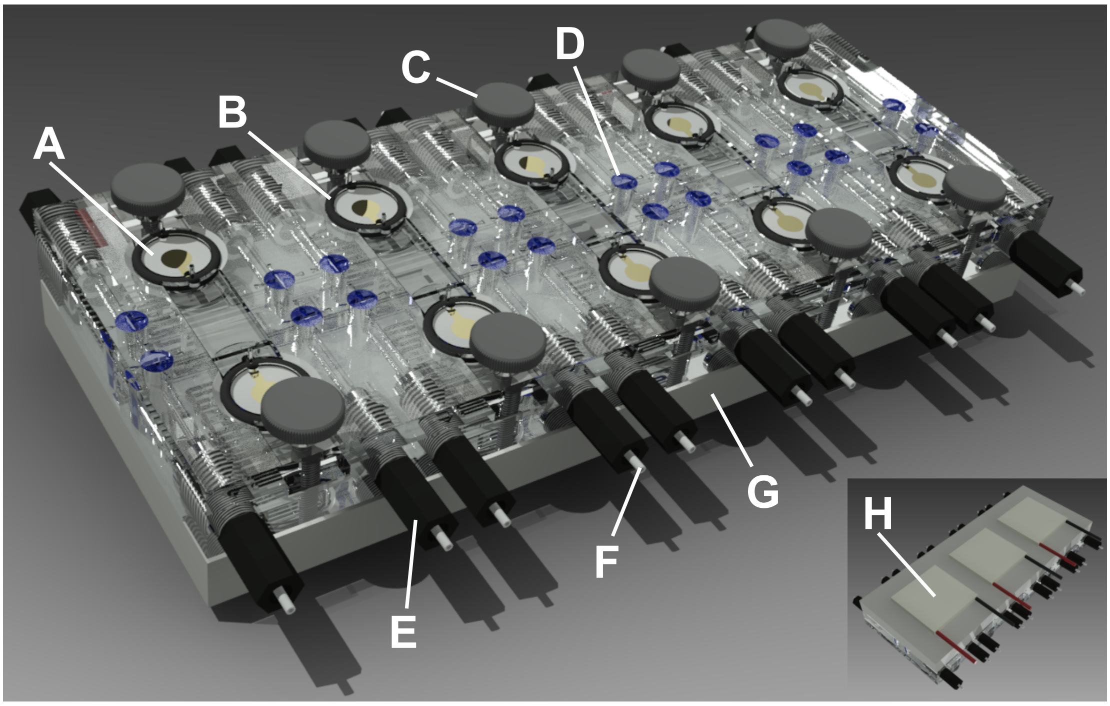



Figure 10: 3D Model of the QCM-D Array; A: QCM-D chip, B: Sealing ring, C: Fixing screw, D: Ventile, E: Tube fitting, F: Teflon tubing, G: Steel plate, H: Peltier element

The 3D model of the complete fluidic array is shown in figure 10. The array consists of a lower part containing the fluidic structures and an upper part providing the electrical connections and the fixation of the QCM-D chips. The array can be loaded with up to ten QCM-D chips (**A**). The chips are fixed between two sealing rings (**B**). One fixing screw (**C**) per chip allows a position independent defined pressure over the chip. Samples enter the array through teflon tubings (**F**) with fittings (**E**). Below the fluidic part a steel plate (**G**) with peltier elements (**H**) allows to control the temperature of the samples and the array.

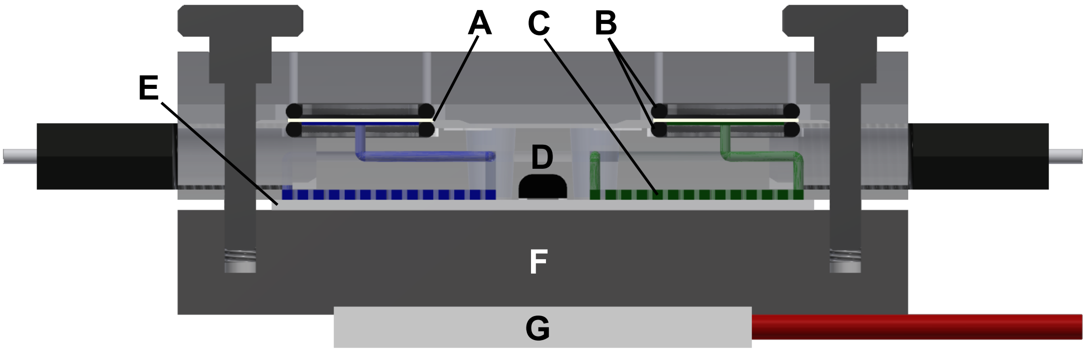



Figure 11: Section through the fluidic array; A: QCM-D chip, B: Sealing rings, C: Cooling channels, D: Temperature sensor, E: Glass sheets, F: Steel plate, G: Peltier element

Figure 11 shows a section through the middle of the array. The QCM-D chips (**A**) are fixed between two sealing rings (**B**). Before the sample reaches the chip, it flows through cooling channels at the bottom of the array (**C**). The temperature sensor (**D**) is located at the same level as the cooling channels, allowing to control the temperature of the sample flowing through the channels. The cooling channels are sealed with glass sheets (**E**) and fixed to the steel plate (**F**) with screws.

On one side the steel plate has ten threads for the fixing screws and three indentations for the peltier elements on the other side. Figure 19 shows the technical drawing given to the workshop for the production of the plate. Steel was choosen as material for the plate because of its high thermal capacity and its quite low thermal conductance. Caused by the low thermal conductivity the plate needs about half an hour to reach its target temperature, but once reached it also holds its temperature some time. This helps to keep the temperature of the samples and the chips constant against external disturbances.

The internal hollow structure of two cells of the array are shown in figure 12. The sample enters the array through the first fitting (**A**) and flows through the first ventile (**B**) into the cooling channels (**C**). The channels are used to bring the sample to the measurement temperature by contact with the temperature controlled glass plate that covers it below. The sample flows from the cooling channels over the QCM-D chip (**D**) to the next ventile (**E**). There it can either be routed to the first exit fitting (**F**) to leave the array or to the next cell (**G**). At the edges of the array the sample can be routed to the other row (**H**) allowing every possible sample distribution between the ten cells.

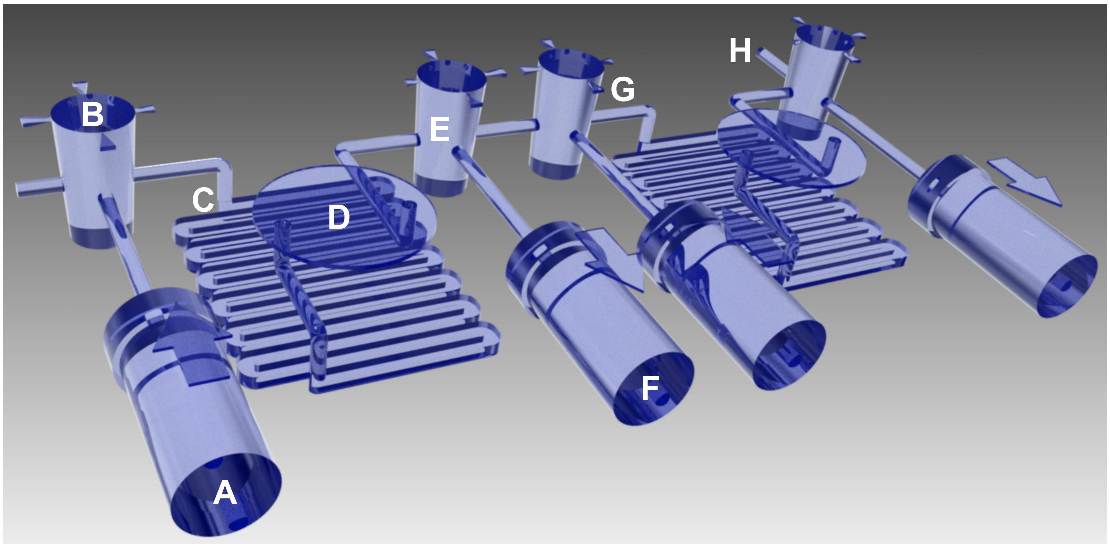



Figure 12: Structure of the channels inside the fluidic array; A: Sample inflow through first fitting, B: Input ventile, C: Cooling channels, D: Sample volume below the QCM-D chip, E: Exit ventile, F: Exit through fitting, G: Exit to the input ventile of the next chamber, H: Routing of the sample over the edge

### 3.1.2 Ventiles

The three way ventiles used to route the sample are shown in figure 13. The best results for removability and tightness were resulted with the diameter of the lower part of the conical shape of 4.075 mm. The removal tool shown in figure 13 (b) is used to remove the ventiles from the array to rearrenge the sample distribution. It consists of copper wire, shaped to a hook at one side and soldered to a loop at the other side (figure 13 (c)). To be able to remove the ventiles they contain a hollow structure at their top shown in figure 13 (a). It allows the tip of the removal tool to be inserted and rotated by 90°. By throwing at the removal tool the ventile can be removed from the array.

### 3.1.3 Printed fluidic parts

Figure 14 shows the printed fluidic parts. In the top part of the array the gold plated electrical contacts (**A**) are wrapped around the sealing rings (**B**). The sample reaches the chips through the channels (**C**). The ventiles (**D**) determine the way of the samples through the array. The chip holding mechanism was adopted from the commercial available flow cell QFM 401 contained in the Qsense 4 QCM-D system. The used chips, sealing rings and fittings were original spare parts available for the system (LOT-QuantumDesign GmbH, 2015).

(a) 3D model of the ventile with internal structures

(b) Printed ventile with tip of the removal tool

(c) Printed ventile with fitted removal tool

Figure 13: Internal structure of the ventiles and the ventile removal tool

### 3.1.4 Flexible sample routing

Figure 15 shows the printed fluidic part equipped with the sealing rings and the glass sheets loaded with ten QCM-D sensors. The ventiles are inserted in the positions allowing a routing of four samples over three groups of three chips and the remaining single chip as shown in figure 16. After closing the array four pump tubes and four tubes for the sample inflow were connected to the array.

The result of using four samples differently colored using food color is shown in figure **??**. To be able to see the samples in the cooling channels the upper part of the array was fixed with nuts and the array was placed upside down. The channels are all tight and the ventiles are able to route the samples over the chips as intended (compare to figure 16.

### 3.1.5 Cooling plate holder with fans

The peltier elements were fixed into the hollow structures of the cooling plate using thermal compound pads. Chipset coolers were used to archive a better heat flow. They consist of a heat sink with termal compound and a fan mounted onto it. The direction of the airflow of the fans was initially from the ambient to the cooler. As this leads to a heating up of the plate in the cooling mode by the warm air flowing over the plate the fans were removed and mounted in the opposite direction using wire as spacer. The cooling plate is fixed in a holder made of insulated copper wire shown in figure 18. At its back side additional fans suck in ambient air and remove the hot or cold air from the chipset coolers to the back side.

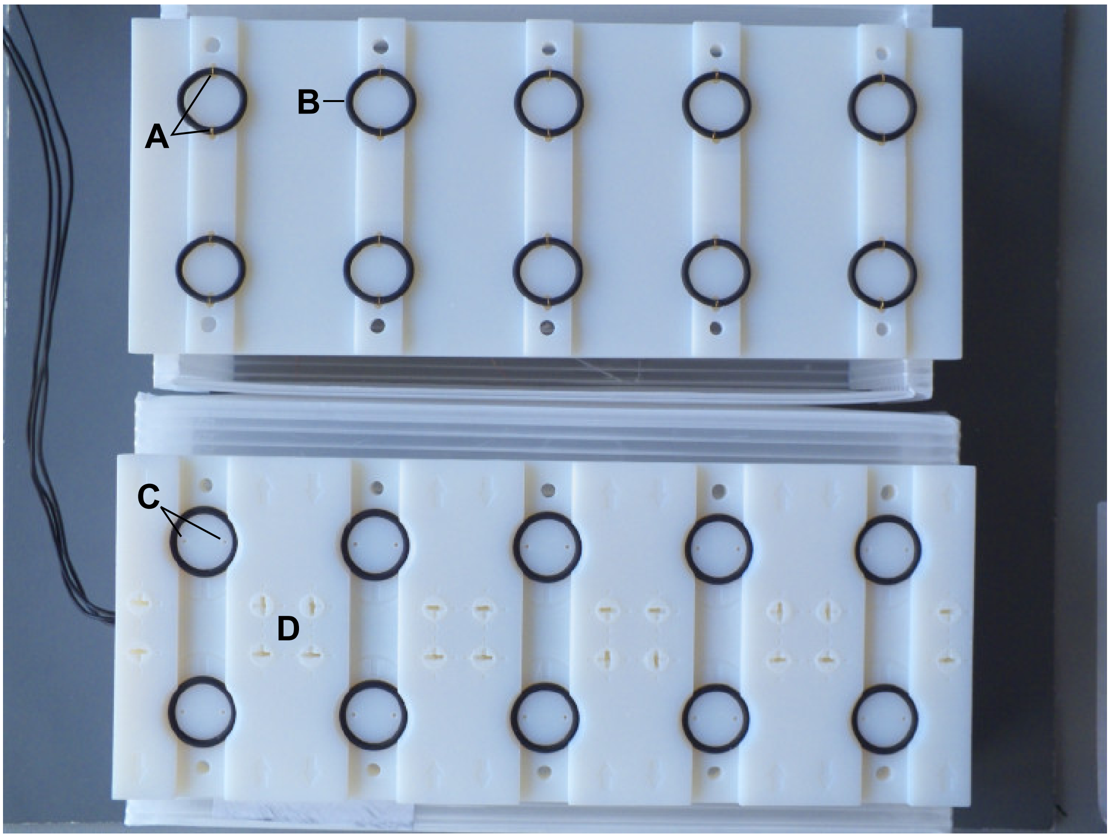



Figure 14: Printed fluidic parts; A: Gold plated electrical contacts, B: Sealing ring, C: Sample channels for the QCM-D chip, D: Ventiles

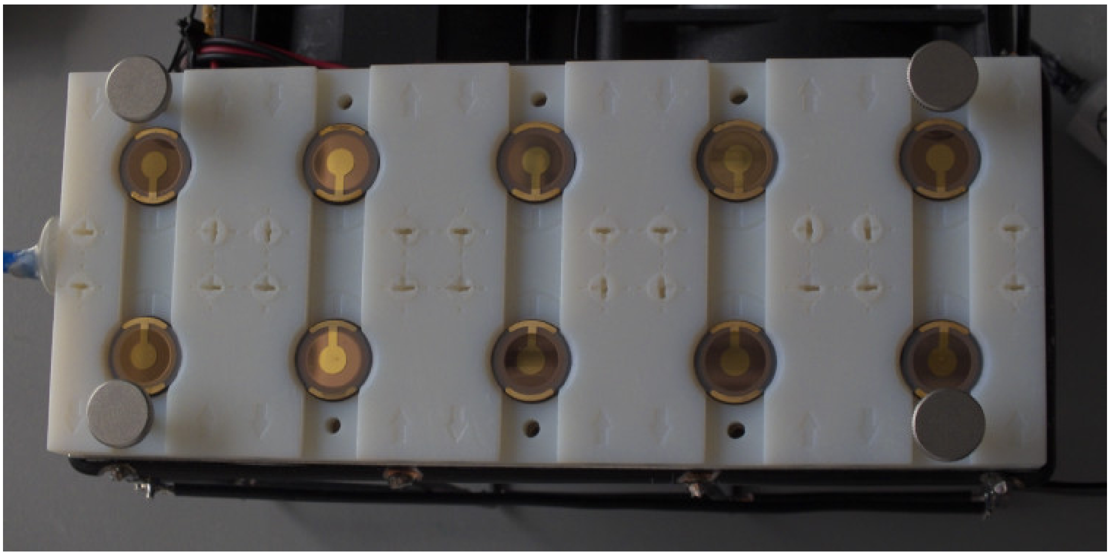


Figure 15: Fluidic loaded with QCM-D chips

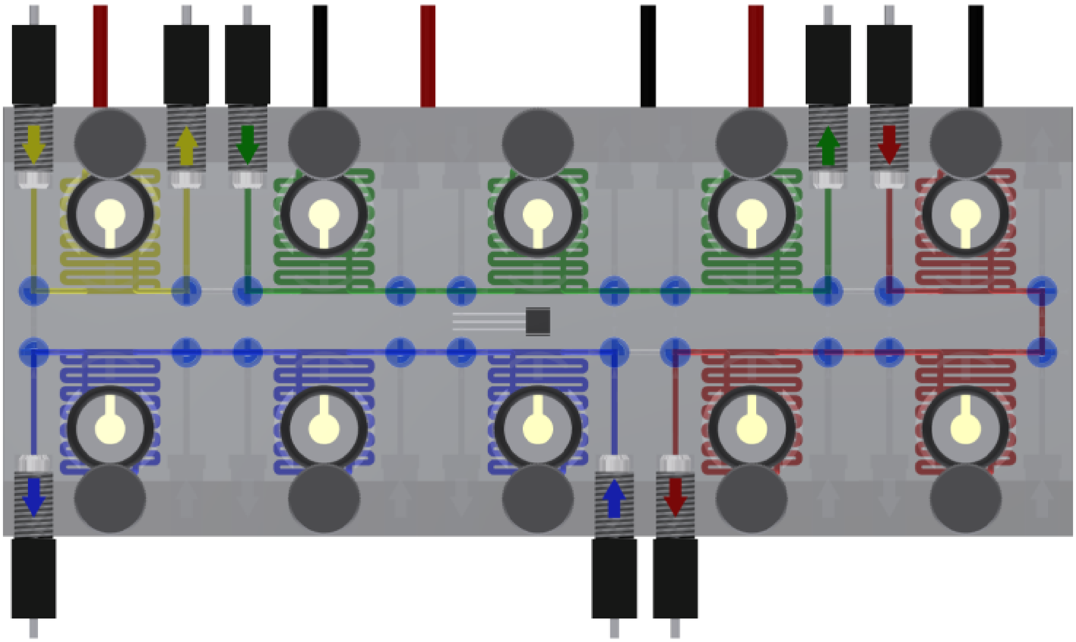


Figure 16: Flexible sample routing

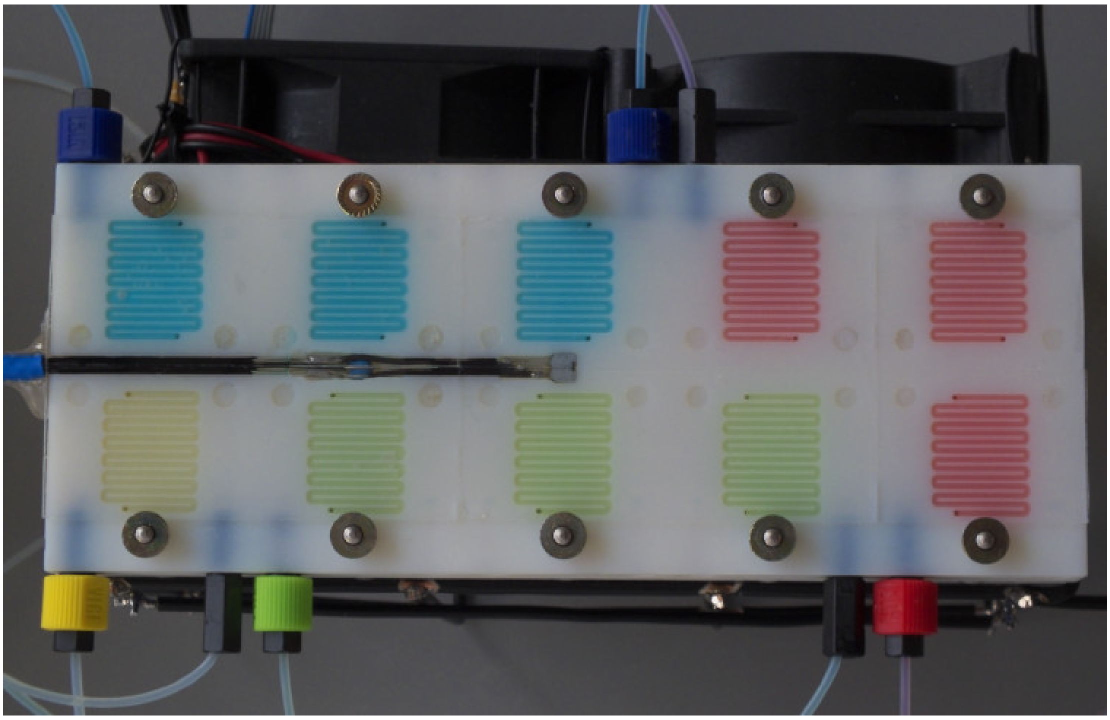


Figure 17: Routing of four samples in the printed array

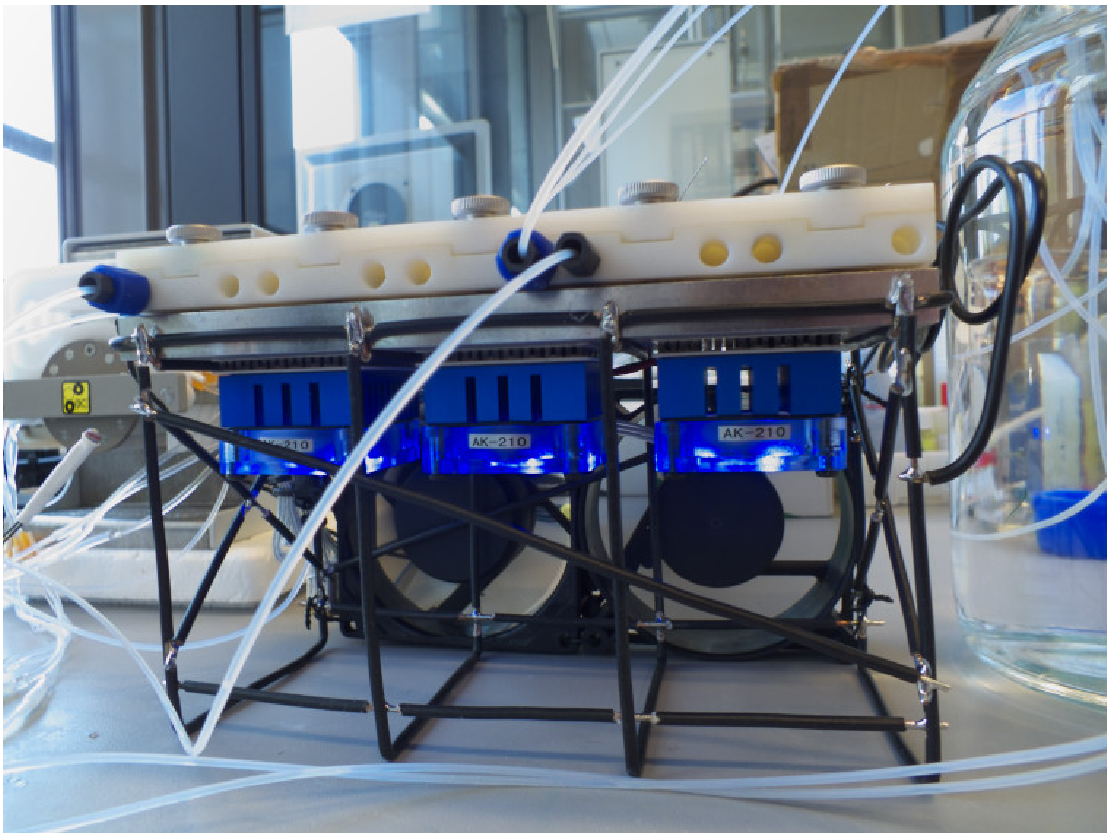



Figure 18: Cooling plate holder with fans

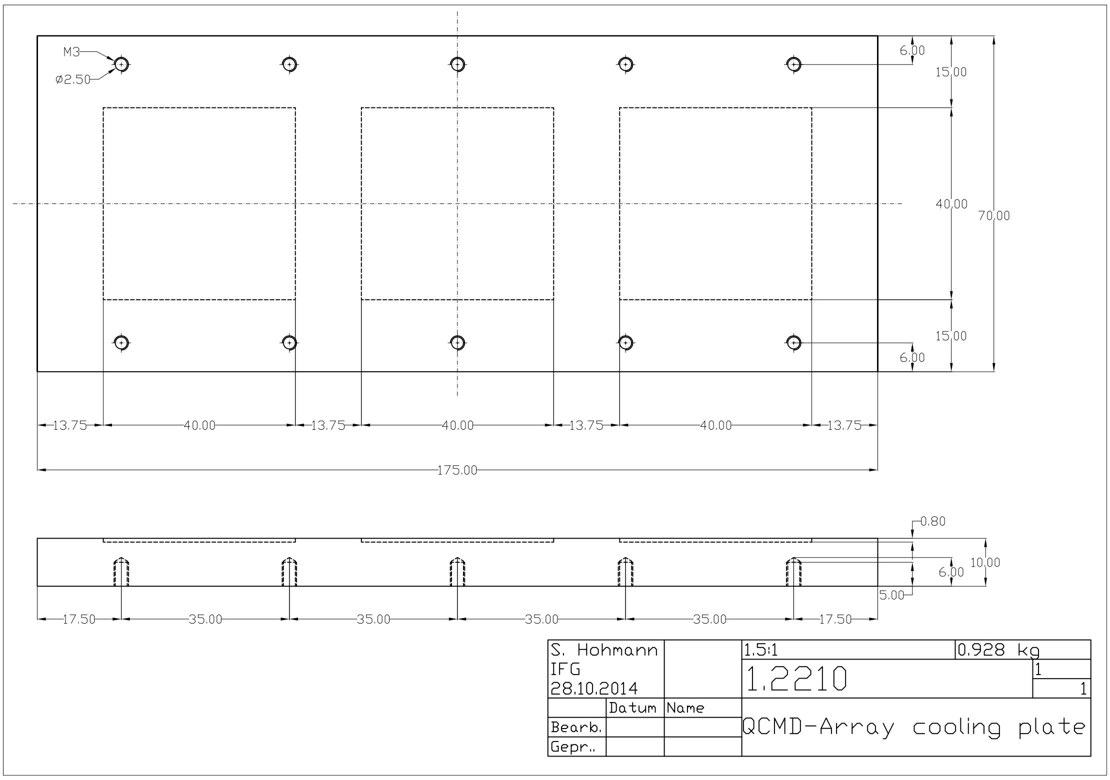



Figure 19: Temperature control steel plate

## 3.2 Temperature control

### 3.2.1 Concept

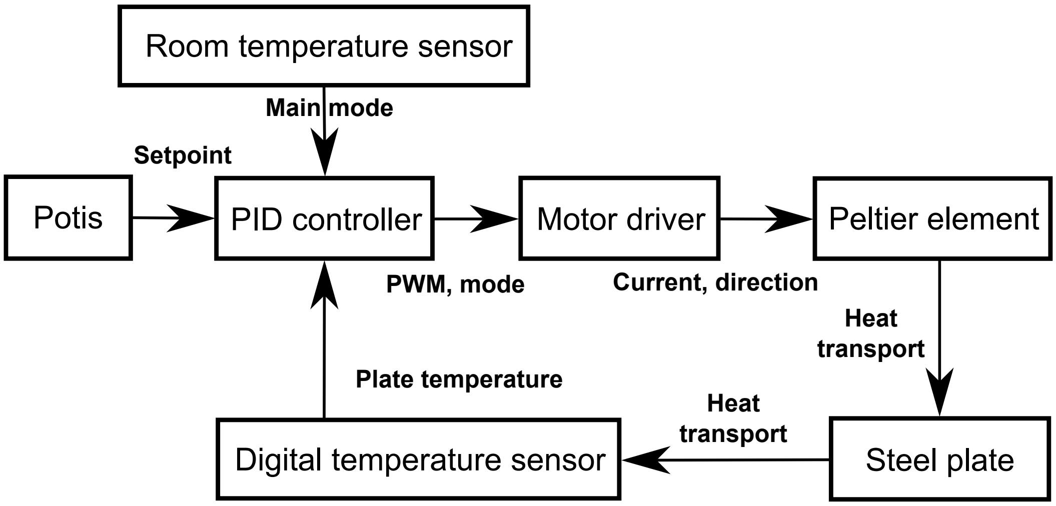



Figure 20: Control loop

The conception of the temperature controller is shown in figure 20. The wanted temperature is set using two potentiometers. They are connected as voltage dividers between ground and the stabilized voltage of the arduino board. They divide the voltage on their outer contacts to a fraction of it on their sliding contacts depending on the angle their axis is moved to. The voltage is measured using two of the analog to digital converter channels contained in the microcontroller board. One potentiometer is used to set the coarse temperature. Its ADC reading is mapped to an output between 20 and 40. The output of the one used to set the fine part of the temperature is mapped to values from 0 to 9. Combining their outputs the temperature can be set between 20.0 and 40.0 degree celsius. Readings of the fine potentiometer are ignored, if the coarse setting is 40 °C. The room temperature sensor is used to estimate the room temperature and thereby to determine the main operation the controller has to fullfill. If the setpoint is below the room temeperature the main mode is cooling and the determined controller parameters for cooling are used. If it is above the room temperature the heating mode and parameters are used. The outputs of the PID controller are the mode signal changing the direction of the current supplyed to the peltier elements and a pulse width modulated signal, controling the amount of current. The peltier elements mounted onto the cooling plate heat or cool the plate depending on the direction of the current flow through it. The plate temperature sensor is mounted inside the fluidics at the same hight, the sample flows through the cooling channels. Its output is read by the PID controller closing the control loop.

### 3.2.2 Circuit

#### 3.2.2.1 Power supply

Figure 21 shows the circuit diagram of the power supply unit of the temperature controller. An external laboratory power supply is connected between $12VC$ and $GND$. The electrolyte capacitor $C1$ buffers the voltage against fast changes in the drawn total current. The $12V$ voltage is used to drive the peltier elements and the fans. $U3$ is a linear voltage regulator of the type 7809, which stabilizes its input voltage down to 9 V. The $9V$ voltage is used as input for the microcontroller board, which stabilizes down to 5 V internally. This $5Varduino$ is used as reference voltage for the ADC channels of the board. To recieve accurate readings it it also used for the potentiometers, the voltage divider of the room temperature sensor and the current measurement. The $9V$ voltage is further stabilized down by the second linear voltage regulator $U6$ of the type 7905. Its stabilized 5 V voltage is used for the 7 segment display and the bus driver bewteen the micropocessor board and the peltier driver units.

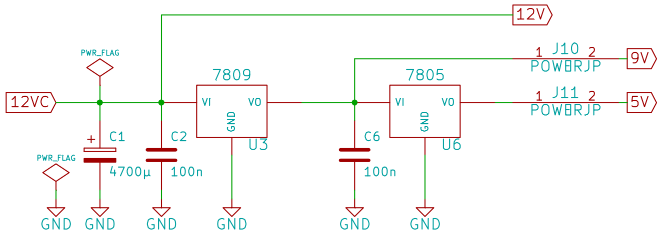



Figure 21: Circuit of the power supply for the temperature controller

#### 3.2.2.2 Peltier driver unit

One of the three used peltier driver units is shown in figure 22. The used motor driver $U7$ of the type L6203 is supplied with voltage from the $12V$ line. Its input voltage and its internal voltage reference output are buf-fered using a 100 µF elecrolyte and a 100 nF ceramic capacitor. They were located as close as possible to the driver on the circuit board. The bootstrap capacitors $C11$ and $C12$ and the fast switching diodes $D5$ and $D6$ of the type UF 4508 help to supply enough charge for the transistors in the driver allowing a fast switching. The capacitors $C13$ and $C14$ are used to smooth the PWM current for the peltier elements. They are connected between the two outputs of the driver. The shunt resistor $R19$ is connected to the sense output of the driver. The voltage drop on it can be used to measure amount of current, because all current flowing through the peltier element is also flowing through it.

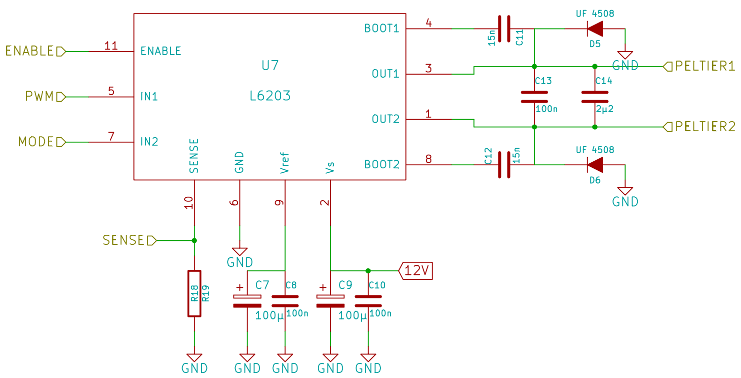



Figure 22: Circuit of the peltier driver unit of the temperature controller

The inputs of the driver are $ENABLE$, $PWM$, and $MODE$ connected to the enable and switching inputs of the driver. The enable function is used to turn on the peltier driver units after the init process to allow the control. One of the inputs is used to determine the direction of the current flow. The other one is supplyed with the PWM output of the PID controller. By changing the duration it is switched to one of its states while the total interval of the pulse width allows to control how long the peltier element is supplied with current (Pang et al., 2015).

### 3.2.2.3 Current measurement

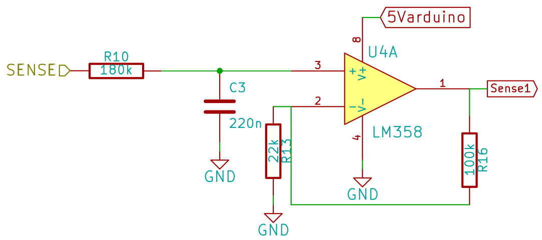



Figure 23: Circuit of the current measurement in the temperature controller

One unit of the current measurement is shown in figure 23. The voltage from the shunt resistor is connected the RC pass built by $R10$ and $C3$. It converts the pulse width modulated input into a direct voltage. This voltage is connected to the positive input of the operational amplifier $U4A$ of the type LM358. The amplifier is connected as non inverting amplifier and supplied by the $5Varduino$ voltage. Its gain is determined by the ratio of the resistors $R13$ and $R16$. It was set to result in the maximum output voltage of the amplifier for a voltage of 0.9 V at the shunt resistor. Its output is connected to a ADC channel of the microcontroller board (Tieze and Schenk, 1990).

### 3.2.2.4 Fan driver units

The fan driver units of one of the chip driver fans and the back fans are shown in figure 24. To switch the fans on the peltier elements the general purpose small signal transistor $Q2$ of the type BC547 is used. To switch the back fans the transistor $Q4$ of the type BC639 is used. It allows a higher collector emmiter current needed to drive the 80 mm fans on the back. The transistors are connected with common emmitter.

Between their collector and the $12V$ line the fans and the diodes $D3$ and $D4$ of the type 1N4148 are connected. Applying a voltage to the resistors $R2$ or $R4$ by the microcontroller switches the transistors and connects the teminal of the fan to ground. The diode is used to protect the circuit from incuced voltages in the coils of the fans by shorting them. The values of the basis transistors were calculated depending on the amount of collector emmitter currend needed for the fans and the gain of the transistor. The transistors are saturated at these working points to allow a fast switching.

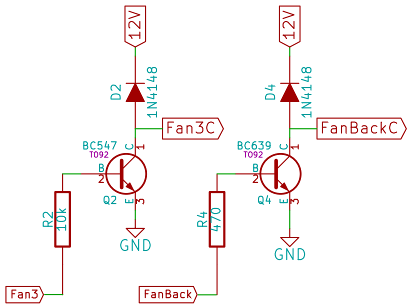
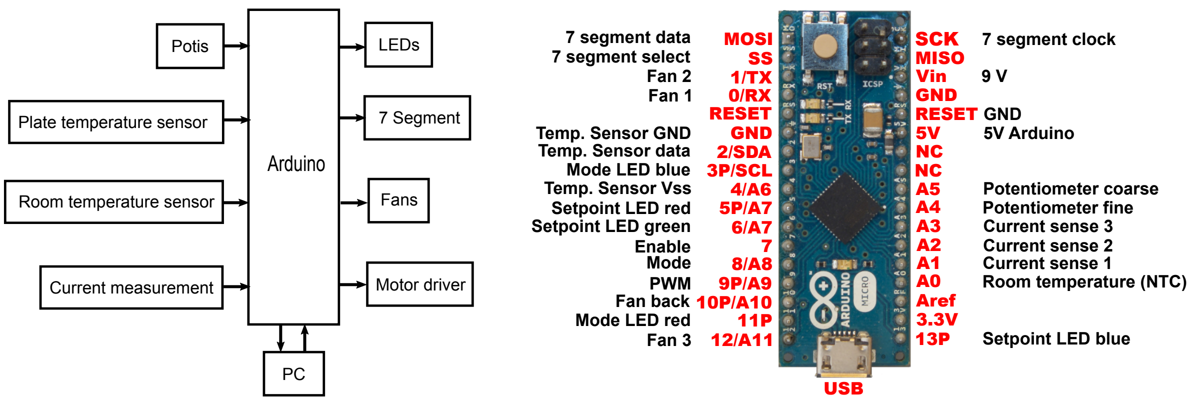



Figure 24: Circuit of the fan driver unit of temperature controller

#### 3.2.2.5 Usage of the microcontroller board




(a) Overview of the connections

(b) Detailed pin usage of the arduino board

Figure 25: Usage of the arduino microcontroller board as PID controller

Figure 25 (a) gives an overview of the connections to the microcontroller board. The potentiometers to set the wanted temperature, the plate with the room temperature sensor and the current measurement units are connected to input lines. The setpoint and the mode LEDs, the 7 segment display, the fans and the peltier driver units are connected to its outputs. The USB connection allows the programming of the controller and transmits the control parameters to a PC.

The detailed usage of the Arduino Micro pins is shown in figure 25 (b). The usage of the pins is shown in black, the function of the pins in red. The room temperature sensor, the three current measurement units and the potentiometers are connected to five of its analog inputs $A0$ to $A5$. The digital plate temperature sensor is connected to ground, Pin 4 and Pin 2. Pin 4 supplys the voltage for the sensor allowing to turn it on for the measurement. Its transmitted data is read from Pin 2. The 7 segment display is connected to the ICSP pins $SS$, $MOSI$ and $SCK$. $SS$ selects the display driver for data transmission. $MOSI$ is used to sent the values to display to it serially. $SCK$ provides the clock for the serial transmission. The fan drivers are connected to the digital pins 0, 1, 12 and 10. The setpoint LED is connected to the PWM capable pins 5, 6 and 13 for its red, green and blue input respectively. The mode LED is connected to pin 3 and 11 with its inputs for blue and red respectively. The green input is not connected as the LED is used to indicate the mode of heating or cooling by its intensity in red or blue only. The enable line of the peltier driver units is connected to pin 7. Weather it is heating or cooling is set by pin 8. Pin 9 is used to supply the PWM signal for the peltier drivers. The board is supplyed with power by the $9V$ line. Its on board stabilized 5 V voltage is supplyed to the potentiometer, room temperature sensor and current measurement circuits over the $5Varduino$ line (Wheat, 2011).

### 3.2.3 Printed circuit board

The copper side of the designed printed circuit board (PCB) is shown in figure 26. The common ground of the circuit is supplyed by a ground plane over the entire PCB. This allows to reduce influences between the subcircuits. Were the motor drivers are mounted the ground plane was removed to reduce the risk of short circuits and to allow a better dissipation of the heat caused by the current through the copper lines.

Figure 27 shows the component mounting diagram of the PCB. The power is supplyed though a connector at the right side in the front of the PCB. Here it has the highest distance to the other parts of the circuit. The three identical peltier driver units are located in the back half of PCB allowing a better heat dissipation of the heat sinks at the back of the planned casing. The micropocessor board is located at the left side of the front of the PCB allowing to access its USB connection from outside of the casing. The connections between the micropocessor board and the peripherals are located as close as possible to the board.

The assembled PCB is shown in figure 28. The Arduino Micro microcontroller board is plugged into a socket (**A**). The bus driver (**B**) is used to protect the microcontroller from possible errors in the peltier driver units and to supply enough current to them. The motor drivers (**D**) are mounted onto heat sinks (**C**). Termal conductive compound was applyed between the back of the drivers and the heat sinks to archive a better termal conductance. The shunt resistors (**E**) are not mounted directly onto the surface of the PCB to allow a better convection for the heat transport and to protect the insulation of the jumper wires below it. The current measurement

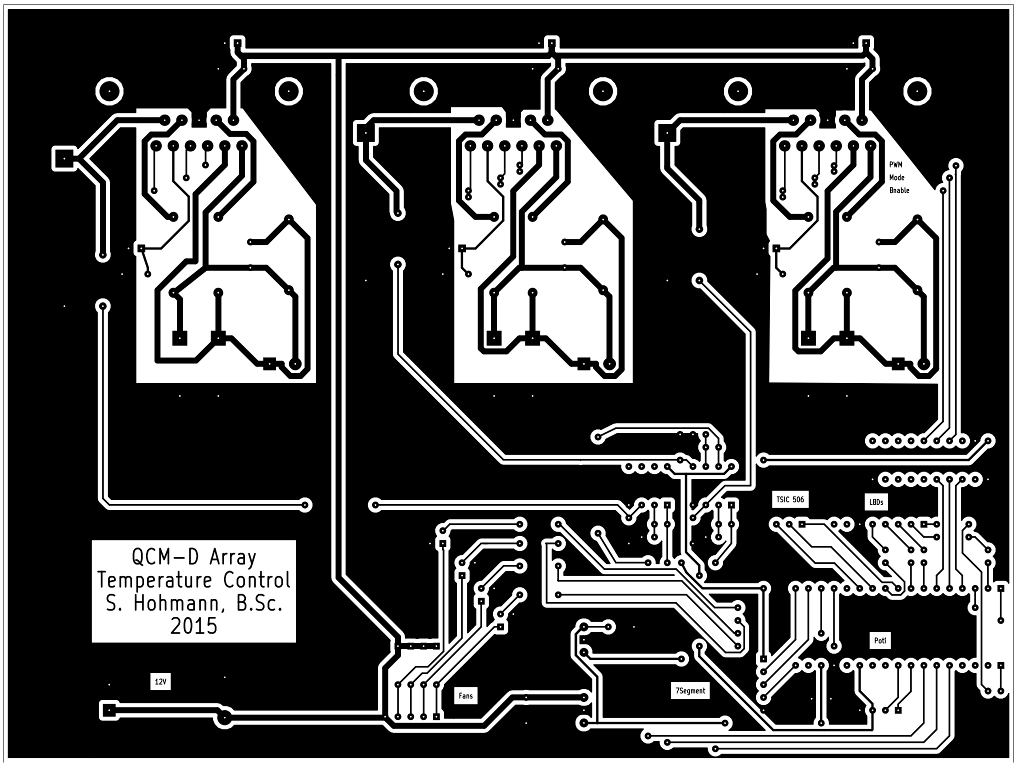



Figure 26: Layout of the printed circuit board

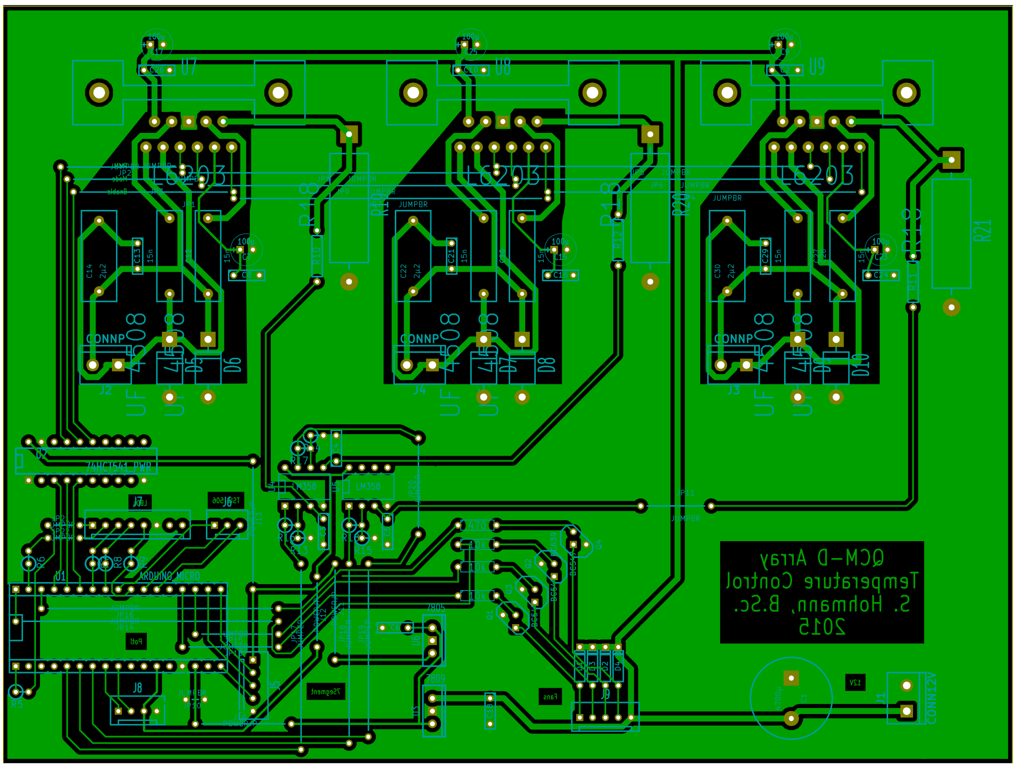



Figure 27: Component mounting diagram of the printed circuit board

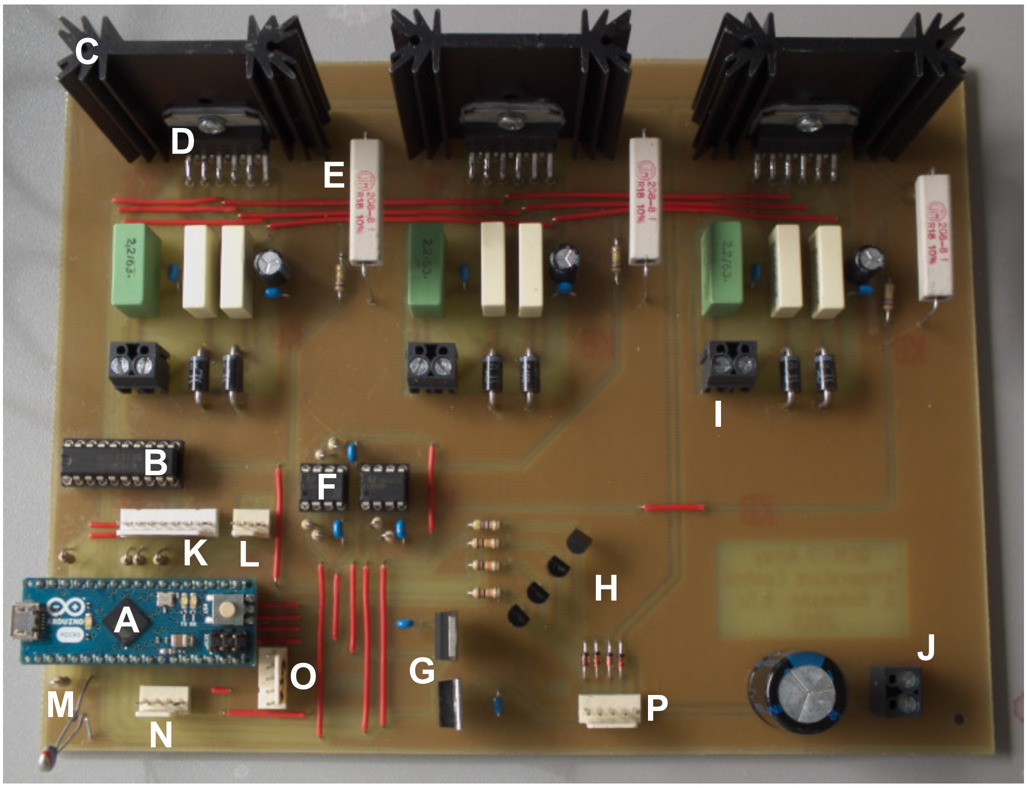



Figure 28: Assembled printed circuit board; A: Arduino Micro, B: Bus driver, C: Heat sink, D: Motor driver, E: Shunt resistor, F: Current measurement, G: Voltage regulators, H: Fan drivers, I: Peltier connector, J: Power supply connector, K: LED connector, L: Temperature sensor connector, M: Room temperature sensor, N: Potentiometer connector, O: 7 Segment display connector, P: Fan connector

subcircuits (**F**) are located near the microprocessor. Because the used LM358 containes two operational amplifiers circuits, only two of them are needed. The unused one is not connected. The linear voltage regulators (**G**) are located left of the fan driver units (**H**). This allows a short distance between the generation of the used voltages and the regulators. The fan drivers use the $12V$ voltage and are thereby located next to its input (**J**). The peltier elements are connected throgh the connectors (**I**). They allow to turn in the cables and fix them with the contained screws. This allows them to be easylie connected. The PCB connectors (**K**), (**L**), (**N**), (**O**) and (**P**) are used to connect the LEDs, the plate temperature sensor, the potentiometers, the 7 segment display and the fans to the circuit. The room temperature sensor (**M**) was initally mounted directly onto the PCB. Because it did heat up by the heat dissipated from the motor drivers, the distance was increased using wires later.

### 3.2.4 User interface

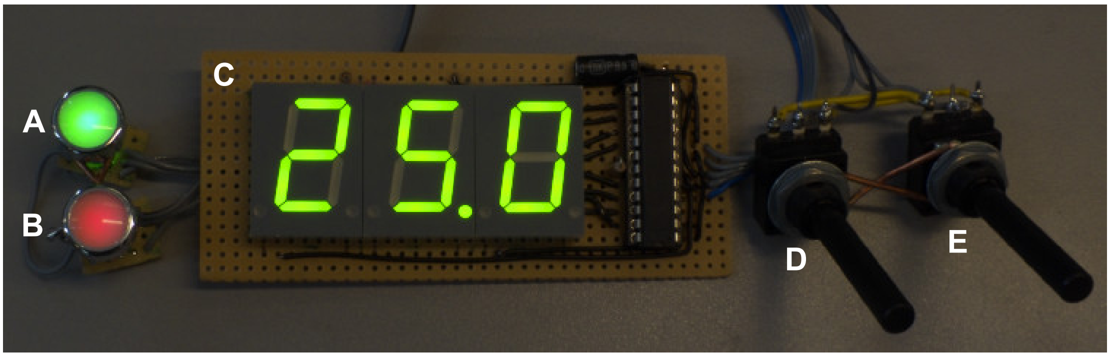



Figure 29: User interface of the temperature controller; A: Status LED, B: Mode LED, C: 7 segment display, D: Coarse setpoint potentiometer, E: Fine setpoint potentiometer

The user interface of the temperature controller is shown in figure 29. It consists of two LEDs (**A**) and (**B**), a 7 segment display (**C**) and two potentiometers (**D**) and (**E**). Their usage is described in the following sections.

#### Setting the wanted temperature

Two potentiometers are used to set the wanted temperature. The left one (D) is used to set the coarse part of the temperature between 20 °C and 40 °C. Is the temperature set with the left potentiometer to a value below 40 °C, the right potentiometer (E) can be used to set the fine part of the temperature between 0.0 °C and 0.9 °C. Is the coarse potentiometer set to 40 °C, the value of the fine potentiometer is discarded.

#### 7 Segment display

The 7 segment display (C) is used to display the actual plate temperature with reduced intensity while the controlling process. Is one of the potentiometers turned it starts to display the setpoint with maximum intensity. The status LED also indicates the setting of the setpoint by its color.

#### Status and mode leds

Table 2 shows the different colors used to signal different states of the controller and the controlling process. While the setpoint is set and the following 700 ms the color of the setpoint LED (A) color is purple. If the setpoint defined its color changes depending on the difference between the plate temperature and the setpoint. Is the difference 5 K or higher, it is red. Below 5 K but

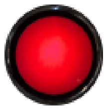
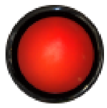
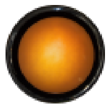
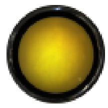
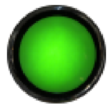
| **LED color** |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
| **Setpoint difference** | Setting | $\geq 5.0$ K | $\geq 2.5$ K | $\geq 1.0$ K | $> 0.6$ K | $< 0.6$ K |

Table 2: Signal colors of the setpoint LED

above 2.5 K it is orange followed by dark yellow for the range from 2.5 K to 1.0 K, light yellow for above 0.6 K and green for below 0.6 K. The highest accuracy of the used temperature sensor is used as the difference below the LED begins to glow green. The green color indicates, that the setpoint is reached. The mode LED (B) signals the amount of current set through the peltier drivers by its intensity and the operation of the peltier elements by its color. For the cooling process it is blue and for the heating process red.

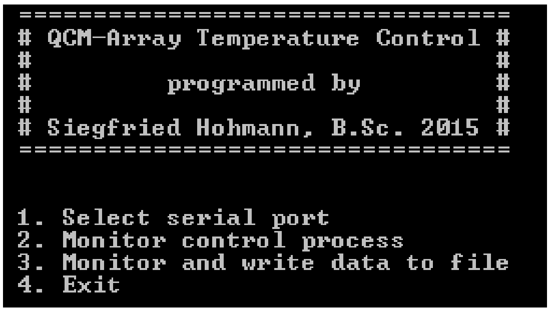
### Init sequence


Figure 30: Init message scrolled over the 7 segment display

When the controller is turned on an init sequence is executed. The status LED shows all its possible signal color while the mode LED shows the intensity of the heating or cooling current applyed to the peltier element by the intensity of its red or blue light in as many steps as the status LED has signal colors. The fans are turned on in sequence, allowing to verify their function by the user. While the init sequence the message shown in figure 30 is scrolled from right to left over the three digits of the 7 segment display.

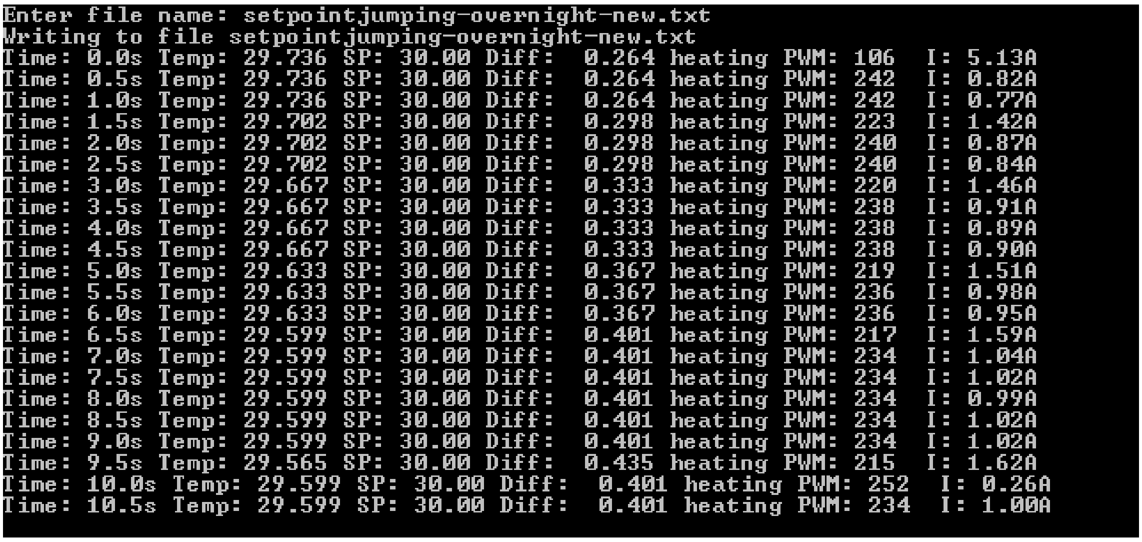
### USB communication with PC

The output of the programm written in C to read the control process data from the arduino micro to a personal computer is shown in figure 31. When executed from the command line, the program shows its menu (a). By pressing the keys for the numbers 1 to 4, the user can select between the given options. To select the serial port 1 is entered in the menu. The program forks to ListSerialPorts.exe by Tod E. Kurt piping its output into a file. The file is openend and the available serial ports and their descriptions are extracted from it. The serial ports available in the system are listed as shown in figure 31 (b). By pressing the number displayed left of the serial port, the serial port is selected and opened for writing. In this case 2 would have been selected, as it is the virtual serial port $COM7$ supplyed by the driver for the Arduino




(a) Menu


(b) Selection of the serial port




(c) Writing controller data to a file

Figure 31: Program to read to read data from the temperature controller to a PC

Micro microcontroller board. Option 2 in the menu allows to monitor the actual control process. Option 3 shows the same output as option 2 on the screen and additionally writes the recieved values into a comma seperated file. If selecting 3 the user is prompted to enter a filename. If the file could be opened for reading this is acknowledged on the next screen line as seen in figure 31 (c). Then the process data recieved from the temperature controller board is displayed in seven columns. At each system hartbeat a new packed is recieved and displayed in the next line. The first column contains the time in seconds since the control process was started. The offset to the higher actual system time was already removed in the Arduino. The next column shows the actual plate temperature in °C with 5 digit resolution. This is the highest resolution containing information, as the digitizing steps in the digital temperature sensor is 0.034 K. To archive this higher resolution the TSIC 506 libary by Wagner (2014) was modified. The next columns show the selected setpoint and the difference between the actual plate temperature and the setpoint used as input for the PID controller. The next column either shows "‘heating"’ or "‘cooling"’ depending on the value of the mode input of the peltier drivers. The next value shows the actual pulse width of the PWM input signal of the peltier drivers. Caused by the fact, that the direction selection and the PWM signal are applied to the inputs of the motor driver, the same pulse width has different effects on the resulting current. In the cooling mode a PWM value of 0 leads to no current, a value of 255 to the maximum current. In heating mode the bahaviour is reversed. This can also be seen in the first two prosess data lines in figure 31 (c). The PWM input of 106 in the first line leads to a current of 5.13 A through the peltier elements. The following value of 242 reduces this current to 0.83 A. The current is displayed in the last column of the output. The currents of the three shunt resistors in the temperature controller are determined after the RC pass had time to settle. To minimize the influence of the residual ripple after the filter the voltage on the capacitor is read with oversampling. The arithmetic averages of the measurements of the three peltier drivers are added and sent to the PC as current value (Prinz and Kirch-Prinz, 2002; Wheat, 2011).

### 3.2.5 Determination of the controller parameters

The determination of the controller parameters is shown exemplarily for on one of the jump responses recorded for the cooling process in figure 32. The measurement starts at the room temperature $Y_0$. The controller was used with the reading of the room temperature sensor as setpoint first followed by 10 minutes without peltier operation. After one hour the plate reached the final temperature $Y_B$. The inflexion tangent was determined by the inflexion point of the fitted polynom as described above. The temperature changes from the stable start value not directly after the input jump was applyed. The moment it starts to change was used as start point for the fitting of the polynom.

Table 3 shows the determined controller parameters for the three measurements for the heating and cooling mode, their arithmetic averages, their standard deviations and their procentual error. The procentual error for the time $T_u$ has a quite high value of over 16 %. This error spreads to the values for $K_p$ and $K_i$ because they depend on it.

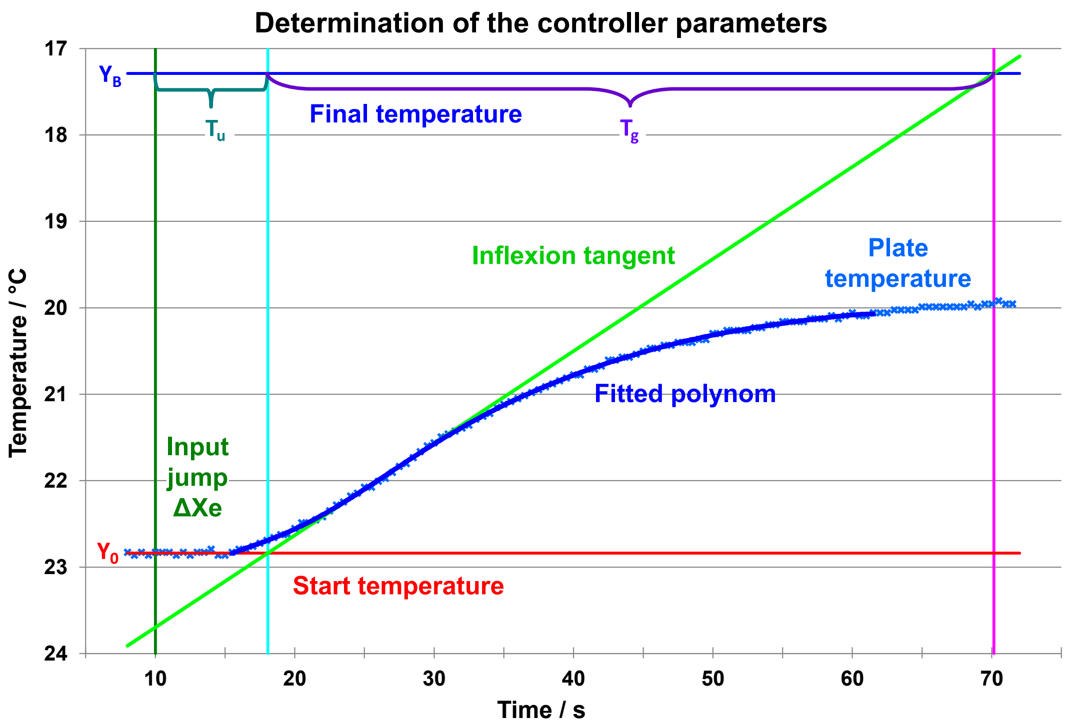



Figure 32: Determination of the controller parameters using the inflexion tangent of an input jump in the openend control loop

| **Mode** | **M. No.** | **Ks**/$\frac{kK}{PWM}$ | **Tu**/$s$ | **Tg**/$s$ | **Kp**/$\frac{PWM}{kK}$ | **Ki**/$\frac{PWM}{kK\cdot s}$ | **Kd**/$\frac{PWM\cdot s}{kK}$ |
|---|---|---|---|---|---|---|---|
| | 1 | 88.63 | 7.85 | 140.9 | 121.5 | 862.8 | 476.8 |
| | 2 | 82.68 | 8.12 | 131.5 | 117.5 | 894.2 | 477.0 |
| cooling | 3 | 90.13 | 8.07 | 142.0 | 117.2 | 824.8 | 472.8 |
| | average | 87.15 | 8.01 | 138.1 | 118.7 | 860.6 | 475.5 |
| | error | 3.9 | 0.14 | 5.8 | 2.4 | 34.7 | 2.4 |
| | error % | 4.5 | 1.8 | 4.2 | 2.0 | 4.0 | 0.5 |
| | 1 | 321.1 | 9.82 | 288.4 | 54.8 | 190.2 | 269.4 |
| | 2 | 318.3 | 11.27 | 266.3 | 44.6 | 167.3 | 251.0 |
| heating | 3 | 317.1 | 8.14 | 268.3 | 62.4 | 232.4 | 253.9 |
| | average | 318.9 | 9.74 | 274.3 | 53.9 | 196.6 | 258.1 |
| | error | 2.1 | 1.56 | 12.2 | 8.9 | 33.0 | 9.9 |
| | error % | 0.6 | 16.1 | 4.4 | 16.6 | 16.8 | 3.8 |

Table 3: Determination of the controller parameters

### 3.2.6 Response of the controller to setpoint jumps

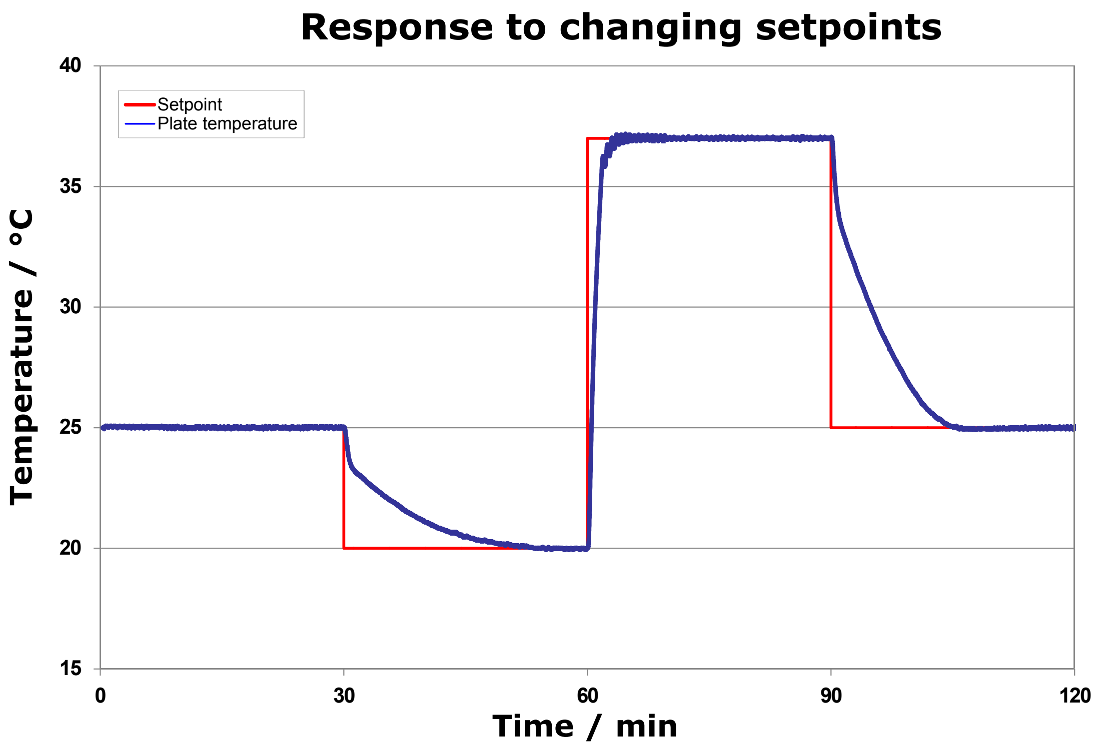



Figure 33: Respond of the control system to changes in the setpoint

Figure 33 shows the response of the control system to changes in the setpoint. Starting at a temperature of 25 °C the setpoint is changed to 20 °C. The plate needs nearly 30 minutes to cool down to this temperature, as the cooling process is the less efficient ones using peltier elements. After the setpoint is raised to 37 °C the plate temperature follows it in about 10 minutes. Here the respond shows oscillations which decay over the time. To cool down to 25 °C again needs about 15 minutes. Except of the oscillations before reaching 37 °C the system responds as wanted showing no over- or undershoots.

### 3.2.7 Long term stability

The long term stability of the control process was determined by cooling the plate down to 20 °C and then rising it to 40 °C in steps of 2.5 K. Each temperature was hold for 90 minutes. The controller response shows initial overshoots when heating up. Their amount is the highest at low temperatures, declining with rising temperature. Because the plate needs a maximum time of 30 minutes to stabilize, the first 30 minutes of every temperature setting were discarded.

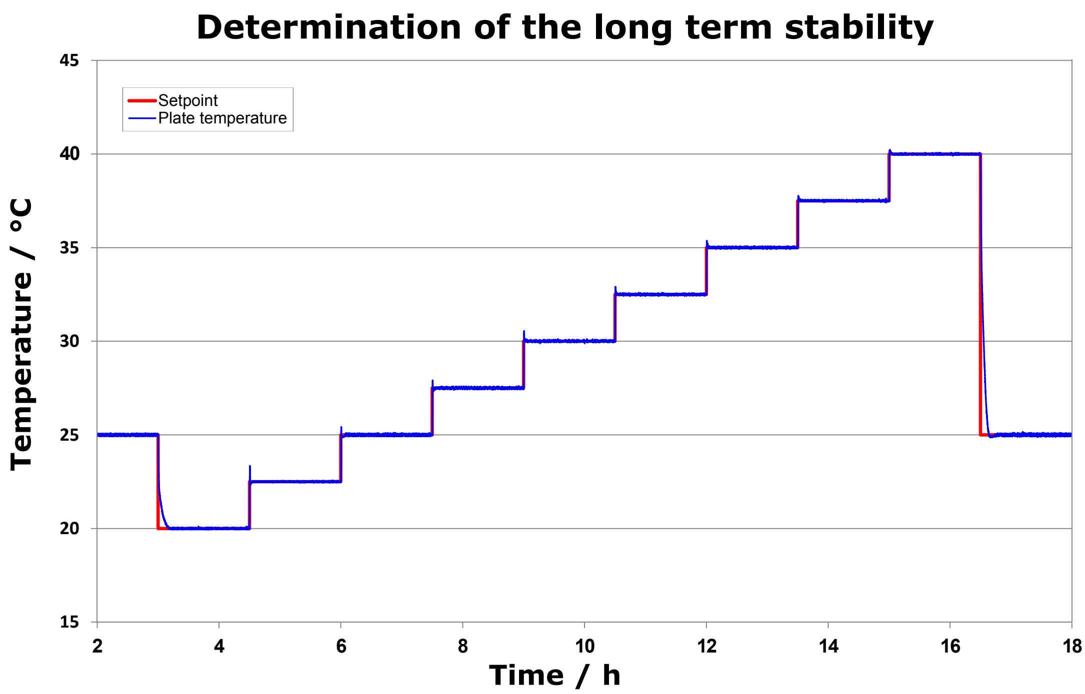



Figure 34: Determination of the long term stability

| Setpoint | Mean temp. | Std. dev. | Max. neg. dev. | Max. pos. dev. | Min. value | Max. value |
|---|---|---|---|---|---|---|
| 20.00 | **20.001** | 0.021 | 0.050 | 0.090 | 19.95 | 20.09 |
| 22.50 | 22.500 | 0.021 | 0.050 | 0.050 | 22.45 | 22.55 |
| 25.00 | 25.000 | 0.027 | 0.050 | 0.080 | 24.95 | 25.08 |
| 27.50 | 27.500 | **0.032** | 0.090 | 0.110 | 27.41 | 27.61 |
| 30.00 | 30.000 | 0.029 | **0.100** | 0.110 | 29.90 | 30.11 |
| 32.50 | 32.500 | 0.026 | 0.070 | 0.100 | 32.43 | 32.60 |
| 35.00 | 35.000 | 0.025 | 0.070 | 0.070 | 34.93 | 35.07 |
| 37.50 | 37.500 | 0.026 | 0.070 | **0.130** | 37.43 | 37.63 |
| 40.00 | 40.000 | 0.018 | 0.080 | 0.060 | 39.92 | 40.06 |
| | Average | 0.025 | 0.070 | 0.089 | | |

Table 4: Long term stability of the temperature control

Table 3.2.7 shows the average temperatures, their standard deviations, their maximum negative and positive read deviations and the maximum and minimum measured values for each temperature. Except of the value of 20.001 °C all mean temperatures are equal to the setpoint. The maximum standard deviation of 0.032 K resulted at a setpoint of 27.5 °C. The maxiumum negative deviation was 0.100 K with a temperature of 29.90 °C at the setpoint 30.00 °C. The maximum positive deviation was 0.130 °C with a value of 37.63 °C at the setpoint 37.50 °C. The average values of the standard deviation and the maxiumum negative and positive deviation were 0.025 K, 0.070 K and 0.089 K respectively. As the accuracy of the digital temperature sensor is 0.06 K in the used range, this values show, that the temperature controller is able to control the temperature with a high enough accuracy over long time.

# 4 Discussion

## 4.1 Fluidic design

### 4.1.1 Integration of the temperature sensor

The tolerance of the hollow structures at the bottom of the array, taking up the temperature sensor and its blocking capacitor, was choosen to low. To insert the sensor the structures were deepend using a rotary tool and a scalpel. Figure 35 shows the result of the UV bonding. The bright structures below the temperature sensor consist of air. The cables, connecting the temperature sensor with the temperature controller, did not flush plane with the outer structure. So the glass plates did not get in contact with the glue and the bonding remained incomplete. The stuctures connect the lower cooling channels with the channel containing the temperature sensor. This would lead to sample reaching the unisolated parts of the sensor and disturbing its function. The glass plates were scraped off the printed structure with acetone and a knife. This caused partly destructions of the cooling channel structures, leading to the formation of air bubbles when the channels are probed. The hollow structures, taking up the temperature sensor, should be deeper in the next version to avoid such bonding errors.

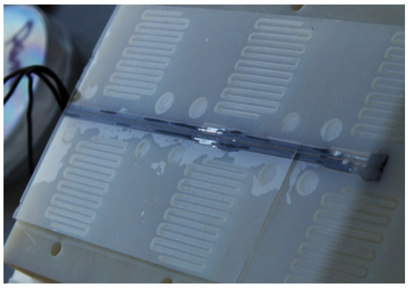


Figure 35: Bonding failure caused by to low tolerances for the included parts

### 4.1.2 Diffusion inside the cooling channels

The diffusion occuring inside the cooling channels while probing three sensors in a row is shown in figure 36. The effect is only visible after the second chip chamber. If it has an influence on the obtained QCM-D signals has to be shown in further experiments. At the left side of figure 36 the destruction of one channel divider and the air bubbles resulting from smaller destructions of the structure can be seen.

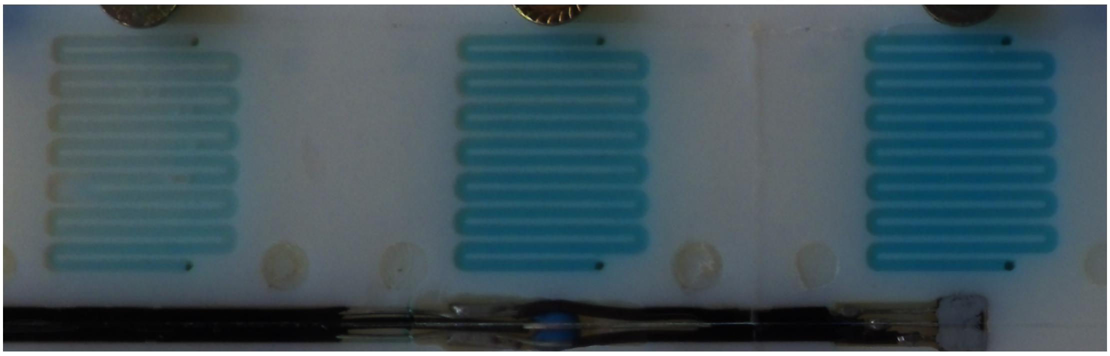


Figure 36: Diffusion inside the cooling channels

#### 4.1.3 Inserting of the ventiles

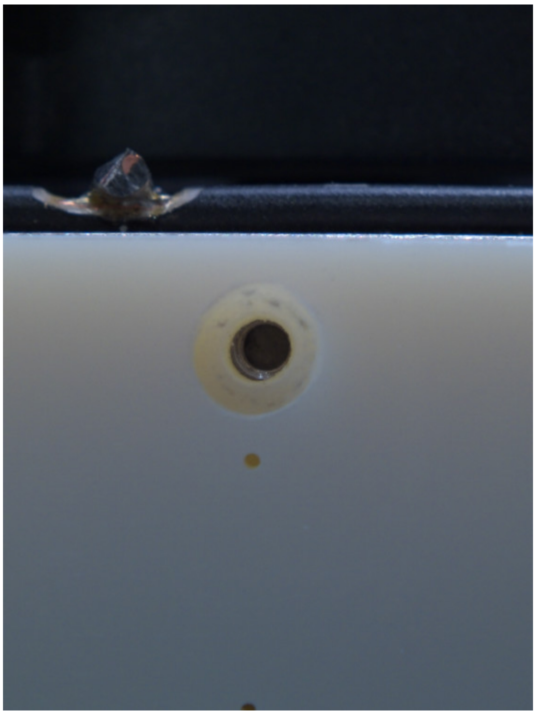
Figure 37: Venting channels for the ventiles

The ventiles were developed and tested using the test bodys shown in figure 6. Their bottom part is open, allowing the air, replaced by the ventile, to escape. In the assembled array the bottom part of the ventiles is closed by the glass sheets glued to the cooling channels. The closed structure did not allow the air to escape and the ventiles could not be inserted completely. Venting channels were cut into the hollow structures taking up the ventiles to alow the air to escape from the ventile. Through this channels air can escape and the ventiles can be inserted completely. The channels should be included in the next version of the array to avoid that problem. Instead of locating them inside the hollow parts of the bottom array a venting channel could be included in the ventile itself. This would not affect the outer surface of the ventile, which is important for the tightness.

#### 4.1.4 Manufacturing of the cooling plate

The correction of tolerance effect while the manufacturing of the cooling plate is shown in figure 38. The positions of the threads, cutted into the plate, are not all accurate (a). To be able to fix the array with all its screws the printed parts were modified using a round file and a scalpel until all screws did fit in without mechanical stress. The plate is much more difficult to manufacture than the printed parts. In further versions of the array the exact position of the threads in the cooling plate should be determined and the position of the holes for the fixing screws in the 3D model of the array should be aligned to the positions in the plate.

(a) Tolerance error of the threads



(b) Corrected printed part

Figure 38: Tolerance effects of the steel plate and correction by modification of the printed parts

## 4.2 PCB design

The amount the Arduino microcontroller exceeds its pins at the opposite side of the USB connector was not correctly factored in. The connector for the 7 segment display located next to the microcontroller board does thereby overlapp with the board. To be able to plug the connector in, parts of it were removed using a cutting disk mounted onto a rotary tool as shown in figure 39. The connector should be placed considering to the dimensions of the arduino board in further versions of the PCB design to avoid the need for this modification.

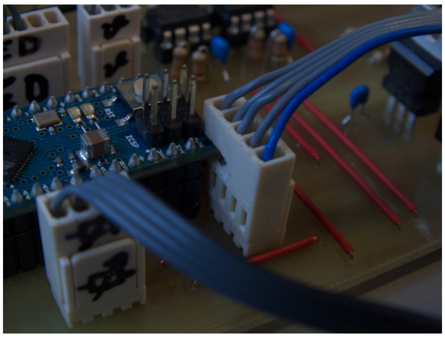


Figure 39: Modification of a PCB connector

# 5 Condluding Remarks

## 5.1 Conclusion

A functional fluidic array for up to ten QCM-D chips was designed. Ventiles allowing a flexible routing through the array were developed. The array was shown to be tight and the sample routing was visualized using colored samples. A temperature controller with an easy to use interface for the user and a USB connetion to a PC was developed and build up. The controller parameters were determined. The controller was able to keep the temperature stable in a range below the accuracy of the used temperature sensor over long time.

## 5.2 Outlook

A second version of the fluidic array should be printed considering the design issus disusses above. The temperature controller should be put into a casing to protect it from outside influences and to facilitate its handling. If a second temperature controller would be build, the PCB design should be adopted to the actual size of the microcontroller board. The QCM-D chips should be connected to the existing commercial QCM-D electronics to determine weather the chips are able to oscillate in it. To actually measure the mass adsorbing onto the QCM-D chips in the array their electrical connections have to be included in an oscillator or vector analyzer circuit.

# References

## a) Books and magazine articles


B. A. Auld. *Acoustic fields and waves in solids*. John Wiley & Sons, Inc, 1. edition, 1973. ISBN 0-471-03702-8.

R. Felderhoff. *Elektrische und elektronische Meßtechnik*. Carl Hanser Verlag, 5. edition, 1990. ISBN 3-446-15608-9.

H. Herold. *Sensortechnik*. Hüthig Buch Verlag, 1. edition, 1993. ISBN 3-7785-2138-1.

S. Hohmann, A. Neidig, B. Kuehl, F. Kirschhoefer, J. Overhage, and G. Brenner-Weiss. A software routine facilitating the identification of pseudomonas aeruginosa proteins involved in conditioning film formation on a titanium dioxide surface monitored by qcm-d/maldi-tof/ms. *Proteomics*, 2015. submitted.

F. Kirschhöfer, A. Rieder, C. Prechtl, and B. Kühl. Quartz crystal microbalance with dissipation coupled to on-chip maldi-tof mass spectrometry as a tool for characterising proteinaceous conditioning films on functionalised surfaces. *Analytica Chimica Acta*, 802:95 – 102, 2013.

D. T. Korsane, V. Yadav, and K. H. Raut. Pid tuning rules for first order plus time delay system. *International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering*, 2(1):582 – 586, January 2014.

C. K. OSullivan and G.G. Guilbault. Commercial quartz crystal microbalances theory and applications. *Biosensors and Bioelectronics*, 14(89):663 – 670, 1999. URL `http://www.sciencedirect.com/science/article/pii/S0956566399000408`. doi:10.1016/S0956-5663(99)00040-8.

D.-Y. Pang, W.-S. Jeon, K.-H. Choi, and T.-K. Kwon. Temperature control using peltier element by pwm method. *ICCAS2005*, 2015.

D. N. Perkins, D. J. C. Pappin, D. M. Creasy, and J. S. Cottrell. Probability-based protein identification by searching sequence databases using mass spectrometry data. *Electrophoresis*, 20:3552 – 3567, 1999.

P. Prinz and U. Kirch-Prinz. *C für PCs.* mitp-Verlag, 3. edition, 2002. ISBN 3-8266-0784-8.

H. Reichl and J. Ahlers. *Halbleitersensoren.* expert verlag, 1. edition, 1989. ISBN 3-8169-0221-9.

S. B. Riffat and Xiaoli Ma. Thermoelectrics: a review of present and potential applications. *Applied Thermal Engineering*, 23(8):913 – 935, 2003. ISSN 1359-4311. doi: http://dx.doi.org/10.1016/S1359-4311(03)00012-7. URL `http://www.sciencedirect.com/science/article/pii/S1359431103000127`.

E. Schrüfer. *Elektrische Messtechnik.* Carl Hanser Verlag, 2. edition, 1984. ISBN 3-446-14203-7.

U. Tieze and C. Schenk. *Halbleiterschaltungstechnik.* Springer-Verlag, 9. edition, 1990. ISBN 3-540-19475-4.

H. Unbehauen. *Regelungstechnik I.* Vieweg & Sohn Verlag, 6. edition, 1989a. ISBN 3-528-53332-3.

H. Unbehauen. *Regelungstechnik II.* Vieweg & Sohn Verlag, 5. edition, 1989b. ISBN 3-528-43348-5.

M. V. Voinova, M. Rodahl, M. Jonson, and B. Kasemo. Viscoelastic acoustic response of layered polymer films at fluid-solid interfaces: Continuum mechanics approach. *Physica Scripta*, 59: 391 – 396, 1999.

Dale Wheat. *Arduino Internals.* Springer Science+Business Media,, 1. edition, 2011. ISBN 978-1-4302-3882-9.

## b) Others

Atmel Corporation. Atmega16U4/ATmega32U4 8-bit Microcontroller with 16/32 Bytes of ISP Flash and USB Controller. Datasheet, September 2014.

DELO Industrie Klebstoffe. DELO PHOTOBOND 4302. Datasheet, 2014.

S. Hohmann, A. Neidig, B. Kuehl, F. Kirschhoefer, J. Overhage, and G. Brenner-Weiss. Software routine allowing identification of extracellular biofilm proteins involved in cell adhesion based on data from qcm-d/maldi experiments. In *IWA Conference The perfect Slime*, 2014.

MAXIM. Serially Interfaced 8-Digit LED Display Drivers. Datasheet, Juli 1997.

NTS electronic and components GmbH. Peltier-element tec1-12705. Datasheet, 2014.

SGS-THOMSON Microelectronics. L6201 L6202 - L6203 DMOS full bridge driver. Datasheet, July 1997.

B+B Thermo Technik GmbH. TSIC™Digitaler Halbleitertemperatursensor TSIC 506. Datasheet, July 2014.

## c) Web sites

Arduino LLC. Arduino Micro. Website, 2015. http://www.arduino.cc/en/Main/ArduinoBoardMicro; accessed 21.04.2015.

LOT-QuantumDesign GmbH. QSense E4. Website, 2015. http://www.lot-qd.de/de/en/home/qsense/e4/; aufgerufen am 28.04.2015.

Stratasys Ltd. Objet eden 3d-drucker. Website, 2015. http://www.stratasys.com/de/3d-drucker/design-series/eden-systems; accessed 28.04.2015.

R. Wagner. Library for TSIC digital Temperature Sensor Type 206/306 and similar. Website, 2014. http://playground.arduino.cc/Code/Tsic; accessed 28.04.2015.

# List of supplementary data

# Supplement A: List of chemicals

| **Substance** | **Manufacturer** | **Order number** | **Charge number** |
|---|---|---|---|
| Water | Millipore Q-Gard 2, Membran Pure Q2 | 194-0009 | 304-1-6-4890R |
| Sodium hydroxide | Merk | 1-06498-1000 | B0127298 780 |
| Ethanol | Merk | 1.000983.1000 | K45251683 405 |
| Ethanol | Laboratory stock | Barrel in storage | n.a. |
| Acetone | Laboratory stock | Barrel in storage | n.a. |
| 2-Propanole | Laboratory stock | Barrel in storage | n.a. |

# Supplement B: List of consumables

| Component | Description | Manufacturer | Order number |
|---|---|---|---|
| Membrane filter | Aerodisk Premium 25 mm Syringe Filter with GxF 1,2 µm GHP Membrane | Pall Life Sciences | A4-4307T |
| Pump tubings | LFL Longlife ID 0,38 mm | TYGON | 070703-04 |
| Water filter | Quantum Ultrapute Organex Cartridge | Millipore | QTUM000EX |

# Supplement C: List of components

| Component | Description | Manufacturer | Order number |
|---|---|---|---|
| 7 segment display | SC08-11CGKWA | Kingbright | 1050568-62 |
| 7 segment driver | MAX 7221 | MAXIM | MAX 7221 CNG |
| Thermal compound | Arctic MX-4 | ARCTIC | ARCTIC MX-4-20 |
| Potentiometer | lin. 6mm 220k | Omeg | PO6M-LIN 220K |
| Bus driver | 74HC541 | SGS Thomson | 74HC 541 |
| Voltage regulator | 7809 | TS | TS 7809 CZ |
| Voltage regulator | 7805 | TS | TS 7805 CZ |
| Galvanisation set | Galvanisier-Zubehör-Set | Conrad | 527983-62 |
| Galvanisation set | Handgalvanisier-Set | Conrad | 530506-62 |
| Copper wire | 1 x 0.20 $mm^2$ | Conrad | 606397-62 |
| Copper wire | 1 x 1.50 $mm^2$ | Lapp Kabel | 607051-62 |
| Copper wire | 1 x 1.50 $mm^2$ | Conrad | 549236-62 |
| Microcontroller board | Arduino Micro 65192 | Arduino | 323485-62 |
| USB cable | 2m A B | Delock | 1007855-62 |
| Circuit board | Proma photoresist | Proma | 528579-62 |
| Matrix hole board | SU527769 | Conrad | 530753-62 |
| Chipset cooler | 28513C60 | Akasa | 999026-62 |
| Diode | BYW98-200 | STMicroelectronics | 155472-62 |
| Peltier element | CP12705 | TEC | 193569-62 |
| Heat sink | CTX/409/50 | CTX Thermal Solutions | 188041-62 |
| Shunt resistor | 0.18 ?5 W | Virtrohm | 427970-62 |
| Temperature sensor | TSIC506-TO92 | B+B Sensors | 506360-62 |
| Thermal compound pad | 0.3 mm 1.4 W/mK (L x B) | Keratherm | 181133-62 |

| Component | Description | Manufacturer | Order number |
|---|---|---|---|
| Electrolyte capacirors | Different values | velleman | K/CAP2 |
| Ceramic capacitors | Different values | velleman | K/CAP1 |
| Resistors | E12 different values | velleman | K/RES-E12 |
| Motor driver | L6203 | STMicroelectronics | 189-1217 |
| Sealing rings | QS-QCS 002 | Lot | QS-QCS 002 |
| Black nuts | QS-QCS 013 | Lot | QS-QCS 013 |
| Screws | 464-M3-16-NI | Ganter Griff | 464-M3-16-NI |
| Fittings | Gripper Fitting, Ferrules Flat Bottom | Diba | 002310 |
| Teflon tubing | PTFE tubing | Bohlender | 271730005 |

# Supplement D: List of devices

| **Device** | **Description** | **Manufacturer** |
|---|---|---|
| Analytic balance | Analytic AC220S | Satorius |
| Digital camera | Finepix HS20EXR | Fujifilm |
| Piston pipettors | Reference 5000 / 1000 / 100 / 10 | Eppendorf |
| Magnetic stirrer | MR 3001 | Heidolph |
| Magnetic stirring plate | Multipoint HP 15 | Variomag |
| Voltage supply | PPS 13610 | Voltacraft |
| Peristaltic pump | Reglo-Analog ISM796B | Ismatec |
| QCM-D flowcell | QFM 401 | Lot Oriel Group Europe |
| Caliper rule | 10059953 | Mitutoyo |
| Water filter | Millipore Q-Gard 2 | Millipore |
| Soldering station | i-CON 1 | Ersa |
| Multimeter | 87 V True RMS Multimeter | Fluke |

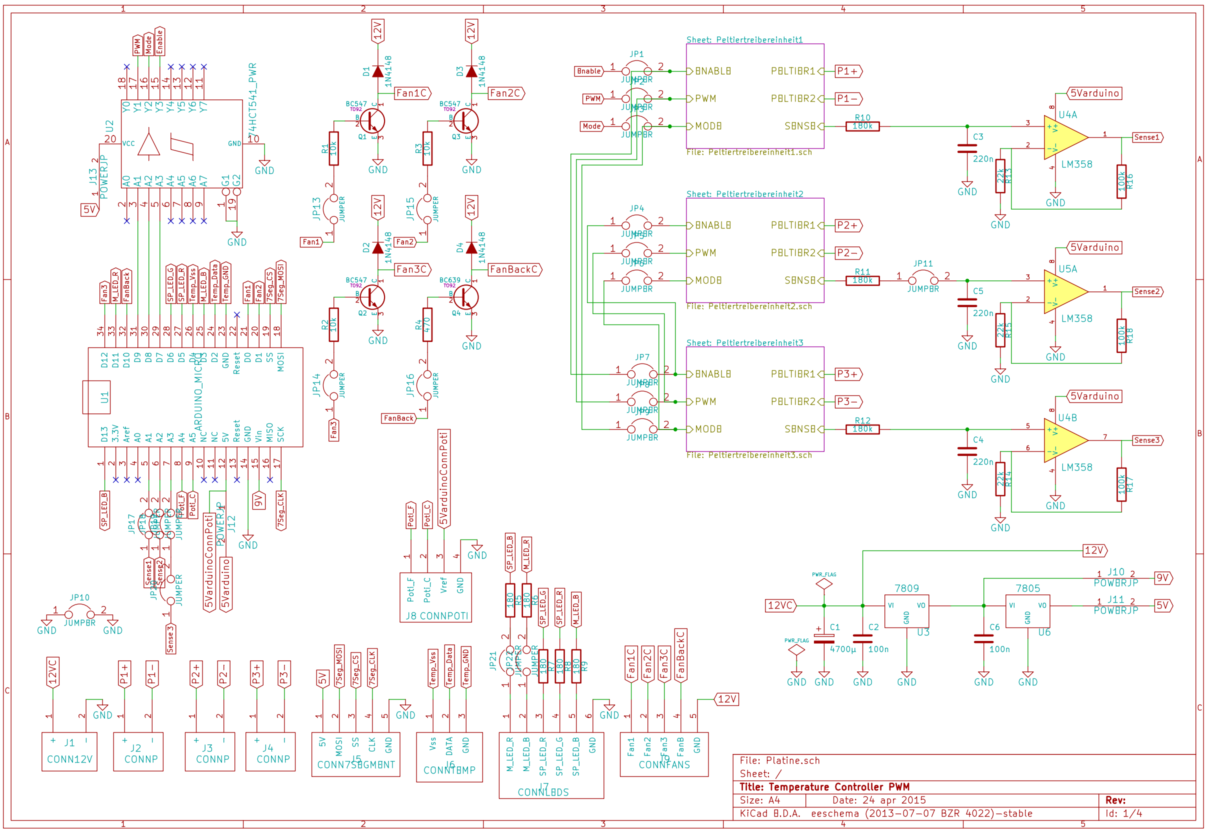
# Supplement E: List of abbreviations

| **Abkürzung** | **Bedeutung** |
|---|---|
| ADC | Analog to Digital Converter |
| DMOS | Double-Diffused Metal Oxide Semiconductor |
| dpi | dots per inch |
| EEPROM | Electrically Erasable Programmable Read-Only Memory |
| GND | GrouND |
| ICSP | In Circuit Serial Programming |
| IO | Input Output |
| LED | Light Emmiting Diode |
| MALDI-ToF MS | Matrix Assisted Laser Desorption Ionization Time of Flight Mass Spectrometry |
| NTC | Negative temperature coefficient |
| PC | Personal Computer |
| PCB | Printed Circuit Board |
| PID | Proportional, Integral, Differential |
| PWM | Pulse Width Modulation |
| QCM | quarz crystal microbalance |
| QCM-D | quarz crystal microbalance with dissipation monitoring |
| RC | Resistor Capacitor |
| SRAM | Short-Range Attack Missile Static Random-Access Memory |
| TFA | TriFluoroacetic Acid |
| TTL | Transistor Transistor Logic |
| USB | Universal Serial Bus |
| UV | UltraViolet |

# Supplement F: List of figures

# Supplement G: List of tables

# Supplement H: Complete circuit diagram of the temperature controller



# Supplement I: Source codes

## Source code of the PID controller on the Arduino board

```

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                                   include librarys
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#include <math.h>
#include <tsic.h>
#include <SoftwareSerial.h>
#include <LedControl.h>

extern "C" {
  #include <inttypes.h>
}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                                   define constants
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

const int potiPinFine = A5;
const int potiPinCoarse = A4;

const int roomTemperaturePin = A0;

const int Sense1Pin = A1;
const int Sense2Pin = A2;
const int Sense3Pin = A3;

const int PWM1Pin = 9; // PWM for Peltier

const int directionPin = 8;
const int enablePin = 7;

const int Fan1Pin = 0;
```

```
const int Fan2Pin = 1;
const int Fan3Pin = 12;
const int FanBackPin = 10;

const int ledPinR = 5;
const int ledPinG = 6;
const int ledPinB = 13;
const int led2PinR = 11;
const int led2PinB = 3;

const int TempVssPin = 4;
const int TempSignalPin = 2;

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                                          define variables
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

boolean dir = false;
boolean mainMode = false;

int error;  //1 = OK, 0 = parity error     return value of getTSicTemp()
int temperatur;  // "return" of temperature in degrees Celsius * 100

uint32_t time, timesense, timeOffset, timeLast, timeSend, settingSinceTime,
    roomTemperatureSinceTime;
uint32_t Ta = 500;         // System hartbeat
uint16_t temperature;
uint16_t temperaturewanted = 30000;
uint16_t temperaturewantedLast;
uint16_t temperaturewantedSend = 30000;
int16_t temperaturedifference;
uint16_t sensorValue = 0;
uint16_t sensorValueSend = 0;
uint16_t outputValue = 0;
uint16_t outputValueSend = 0;

int potiValueFine;
int potiValueCoarse;

unsigned long potiValueFine_Sum;
unsigned long potiValueCoarse_Sum;

const int potiValueSum_Count = 50;
int potiValueSum_Counter;
```

```
int roomTemperature;   // PTC temperatur intern
double roomTemperature_Sum;
const byte roomTemperatureSum_Count = 200;
int roomTemperatureSum_Counter;
boolean roomTemperatureInit = true;


const double roomTemperatureCALslope = -0.099964;
const double roomTemperatureCALoffset = 83.480492;
double roomTemperatureCelsius;
unsigned int roomTemperatureCelsiusInt;
unsigned int roomTemperatureSetpointDiff;
unsigned int roomTemperaturePlateDiff;
unsigned int roomTemperatureSetpointDiffAbs;
unsigned int roomTemperaturePlateDiffAbs;
unsigned int roomTemperatureUsedDiff;

const byte tempDisplayNormalIntensity = 9;

int fine;
int coarse;

unsigned int temperaturePoti;
int temperatureDiffAbs;
int tempSend;

int currentPeltier1;
int currentPeltier2;
int currentPeltier3;
long currentPeltier1Sum;
long currentPeltier2Sum;
long currentPeltier3Sum;
int currentPeltierSumTotal;
int currentPeltierSum_Count = 100;
byte currentPeltierSum_Counter;

const double currentPeltier1CAL = 185.8280;
const double currentPeltier2CAL = 168.7825;
const double currentPeltier3CAL = 174.3662;

double currentPeltier1Ampere;
double currentPeltier2Ampere;
double currentPeltier3Ampere;
```

```
double currentPeltierTotalAmpere;
int currentPeltierTotalAmpereSend;

volatile byte Last_Digit;
volatile byte Digit_0_value;
volatile byte Digit_1_value;
volatile byte Digit_2_value;

volatile boolean Digit_0_DP = false;
volatile boolean Digit_1_DP = true;
volatile boolean Digit_2_DP = false;
volatile boolean Digit_Round;

byte ledValue = 0;
int counter = 0;
byte dirbyte = 0;

byte Outputarray[15];
byte tempBuffer[3];

double e, esum, ealt, y;
double AS = 0;

double Kp;
double Ki;
double Kd;

double KpC = 1.187451E-01;
double KiC = 8.605826E-07;
double KdC = 4.755258E+02;

double KpH = 5.391898E-02;
double KiH = 1.966363E-07;
double KdH = 2.580668E+02;

unsigned long delaytime=220;

byte textCounter;

byte initMessage[] =
    {0,0,254,78,118,61,1,119,5,5,119,59,0,15,79,118,103,79,5,119,15,28,5,79,
        // Q  C  M  D-  A  r  r  a  y  T  e  m  p  e  r  a  t  u  r  e

    0,78,29,21,15,5,29,14,0,219,0,55,29,23,118,119,21,21,0,109,126,48,91,0,0,0};
```

```
 //  C o n t r o l  S .  H o h m a n n  2 0 1 5

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                                  create libary objects
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

tsic Sensor1(TempVssPin, TempSignalPin);

LedControl tempDisplay = LedControl(MOSI,SCK,SS,1);
                                // data, clock, select, number of devices
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                                  fuction prototypes
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

void initLEDstep(byte step);
void initFANstep(byte step);
void scrollInitMessage();
void parseTemp(unsigned int temp);
void getRoomTemperature();

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                                  setup function
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

void setup() {

   TCCR1B = TCCR1B & 0b11111000 | 0b00000001; // set timer 1 divisor to 1
                                              // for PWM frequency of 31372.55 Hz

   Serial.begin(9600);

   pinMode(directionPin, OUTPUT);
   pinMode(enablePin, OUTPUT);
   pinMode(Fan1Pin, OUTPUT);
   pinMode(Fan2Pin, OUTPUT);
   pinMode(Fan3Pin, OUTPUT);
   pinMode(FanBackPin, OUTPUT);

   tempDisplay.setScanLimit(0,2);
   tempDisplay.shutdown(0,false);
   tempDisplay.setIntensity(0,tempDisplayNormalIntensity);
   tempDisplay.clearDisplay(0);

   scrollInitMessage();
```

```
    delay(300);

    digitalWrite(Fan1Pin, HIGH);            // all on
    digitalWrite(Fan2Pin, HIGH);
    digitalWrite(Fan3Pin, HIGH);
    digitalWrite(FanBackPin, HIGH);
    analogWrite(ledPinR, 255);
    analogWrite(ledPinG, 255);
    analogWrite(ledPinB, 255);
    analogWrite(led2PinR, 255);
    analogWrite(led2PinB, 255);
    tempDisplay.setIntensity(0,15);
    tempDisplay.setRow(0,2,255);
    tempDisplay.setRow(0,1,255);
    tempDisplay.setRow(0,0,255);
    delay(1000);

    digitalWrite(Fan1Pin, LOW);            // all off
    digitalWrite(Fan2Pin, LOW);
    digitalWrite(Fan3Pin, LOW);
    digitalWrite(FanBackPin, LOW);
    analogWrite(ledPinR, 0);
    analogWrite(ledPinG, 0);
    analogWrite(ledPinB, 0);
    analogWrite(led2PinR, 0);
    analogWrite(led2PinB, 0);
    tempDisplay.setIntensity(0,tempDisplayNormalIntensity);
    tempDisplay.clearDisplay(0);

    digitalWrite(Fan1Pin, HIGH);
    digitalWrite(Fan2Pin, HIGH);
    digitalWrite(Fan3Pin, HIGH);
    digitalWrite(FanBackPin, HIGH);

    digitalWrite(enablePin, HIGH);

    getRoomTemperature();        // discard first measurement
    delay(500);
    getRoomTemperature();

    timeOffset = millis();
    time = timeOffset;
    roomTemperatureSinceTime = time;
```

```
    timeLast = timeOffset - Ta;
    timeSend = 0;

}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                              loop fuction
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

void loop() {

  while (time < timeLast + Ta) {

    time = millis();

  }

  timeLast = time;
  timeSend = time - timeOffset;

  error = Sensor1.getTSicTemp(&temperatur);
  temperature = temperatur;

  potiValueFine_Sum = 0;
  potiValueCoarse_Sum =0;

  for (potiValueSum_Counter = 0; potiValueSum_Counter < potiValueSum_Count;
    potiValueSum_Counter++) {

  potiValueFine_Sum += analogRead(potiPinFine);
  potiValueCoarse_Sum += analogRead(potiPinCoarse);

  }

  potiValueFine = (int)(potiValueFine_Sum / potiValueSum_Count);
  potiValueCoarse = (int)(potiValueCoarse_Sum / potiValueSum_Count);

  fine = map(potiValueFine,0,1000,0,9);
  coarse = map(potiValueCoarse,0,1000,20,40);

  temperaturePoti = 10 * fine + 100 * coarse;

  temperaturePoti *= 10;

```

```
if (temperaturePoti > 40000) {
    temperaturePoti = 40000;
}
if (temperaturePoti < 20000) {
    temperaturePoti = 20000;
}

  if (temperaturePoti != temperaturewanted) {

  settingSinceTime = millis();
  temperaturewanted = temperaturePoti;
  roomTemperatureInit = true;

  tempDisplay.setIntensity(0,15);     // full intensity

  parseTemp(temperaturewanted);

  tempDisplay.setDigit(0,0,Digit_0_value,Digit_0_DP);
  tempDisplay.setDigit(0,1,Digit_1_value,Digit_1_DP);
  tempDisplay.setDigit(0,2,Digit_2_value,Digit_2_DP);

}


temperaturedifference = temperaturewanted - temperature;

if (temperaturedifference < 0) { temperatureDiffAbs = 0 - temperaturedifference;
   } else { temperatureDiffAbs = temperaturedifference; }

  if (roomTemperatureInit == true) {

  roomTemperatureInit = false;

  roomTemperatureSetpointDiff = temperaturewanted - roomTemperatureCelsiusInt;
  roomTemperaturePlateDiff =  temperaturewanted - temperature;

  if (roomTemperatureSetpointDiff < 0) { roomTemperatureSetpointDiffAbs = 0 -
  roomTemperatureSetpointDiff; } else { roomTemperatureSetpointDiffAbs =
  roomTemperatureSetpointDiff; }
  if (roomTemperaturePlateDiff < 0) { roomTemperaturePlateDiffAbs = 0 -
  roomTemperaturePlateDiff; } else { roomTemperaturePlateDiffAbs =
  roomTemperaturePlateDiff; }

```

```
    if (roomTemperatureSetpointDiffAbs > roomTemperaturePlateDiffAbs) {
   roomTemperatureUsedDiff =  roomTemperatureSetpointDiff; } else {
   roomTemperatureUsedDiff =  roomTemperaturePlateDiff; }

      if (roomTemperatureUsedDiff < 0) {

        mainMode = false;     // cooling

          Kp = KpC;
          Ki = KiC;
          Kd = KdC;

        } else {

        mainMode = true;     // heating

          Kp = KpH;
          Ki = KiH;
          Kd = KdH;

        }
    }

e = temperaturedifference;

  if ((e >= AS)||(e <= (AS*(-1))))          //
  {
    if ((y < 255)&&(y > -255))              // stop integration on overflow
    {                                       // anti windup
      esum = esum + e;                      // integrate
    }

    y = (Kp*e)+(Ki*Ta*esum)+(Kd*((e-ealt))/Ta);   // PID

    ealt = e;                               // store last error for next hartbeat
  }

  if (y > 255)                              // limit output value into range from
    -255 to 255 (9 bit PWM)
    {
      y = 255;
    }
  if (y < -255)
    {
```

```
      y = -255;
    }

  sensorValue = y + 255;

  if (sensorValue < 256) {

    dir = false;                    // cooling
    outputValue = 255 - sensorValue;
    ledValue = outputValue;
  } else {

    dir = true;                     // heating
    outputValue = 255 - (sensorValue - 256);
    ledValue = sensorValue - 256;

  }


  analogWrite(PWM1Pin, outputValue);     // adjust PWM for motor controllers


if(dir == true) {

      digitalWrite(directionPin, HIGH);

      analogWrite(led2PinR,ledValue);
      analogWrite(led2PinB,0);

      dirbyte = 1;

    } else {

      digitalWrite(directionPin, LOW);

      analogWrite(led2PinB,ledValue);
      analogWrite(led2PinR,0);

      dirbyte = 0;

  }

  if (time > roomTemperatureSinceTime + 60000) {
```

```
      roomTemperatureSinceTime = millis();
      getRoomTemperature();
      roomTemperatureInit = true;

  }

  if (settingSinceTime + 700 > time) {

    analogWrite(ledPinR, 255);            // pink
    analogWrite(ledPinG, 0);
    analogWrite(ledPinB, 180);

    tempDisplay.setIntensity(0,15);

    parseTemp(temperaturewanted);

    tempDisplay.setDigit(0,0,Digit_0_value,Digit_0_DP);
    tempDisplay.setDigit(0,1,Digit_1_value,Digit_1_DP);
    tempDisplay.setDigit(0,2,Digit_2_value,Digit_2_DP);

  } else {

    tempDisplay.setIntensity(0,tempDisplayNormalIntensity);

    tempSend = temperature;

    if (temperatureDiffAbs > 4999) {

  analogWrite(ledPinR, 255);            // rot
  analogWrite(ledPinG, 0);
  analogWrite(ledPinB, 0);

    } else if (temperatureDiffAbs > 2499) {

  analogWrite(ledPinR, 224);            // organge
  analogWrite(ledPinG, 30);
  analogWrite(ledPinB, 0);

    } else if (temperatureDiffAbs > 999) {

  analogWrite(ledPinR, 174);            // dunkelgelb
  analogWrite(ledPinG, 109);
  analogWrite(ledPinB, 0);
```

```
    } else if (temperatureDiffAbs > 60) {
  analogWrite(ledPinR, 108);            // gruengelb
  analogWrite(ledPinG, 146);
  analogWrite(ledPinB, 0);
    } else {
   analogWrite(ledPinR, 0);             // gruen
  analogWrite(ledPinG, 255);
  analogWrite(ledPinB, 0);
    tempSend = temperaturewanted;
    }

    parseTemp(tempSend);

    tempDisplay.setDigit(0,0,Digit_0_value,Digit_0_DP);
    tempDisplay.setDigit(0,1,Digit_1_value,Digit_1_DP);
    tempDisplay.setDigit(0,2,Digit_2_value,Digit_2_DP);

 }

  delay(100);  // wait for RC pass to settle

  currentPeltier1Sum = 0;
  currentPeltier2Sum = 0;
  currentPeltier3Sum = 0;

  for (currentPeltierSum_Counter = 0; currentPeltierSum_Counter <
    currentPeltierSum_Count; currentPeltierSum_Counter++) {

  currentPeltier1Sum += analogRead(Sense1Pin);
  currentPeltier2Sum += analogRead(Sense2Pin);
  currentPeltier3Sum += analogRead(Sense3Pin);

  }

  currentPeltier1 = (int)(currentPeltier1Sum / currentPeltierSum_Count);
  currentPeltier2 = (int)(currentPeltier2Sum / currentPeltierSum_Count);
  currentPeltier3 = (int)(currentPeltier3Sum / currentPeltierSum_Count);

  currentPeltier1Ampere = currentPeltier1 / currentPeltier1CAL;
  currentPeltier2Ampere = currentPeltier2 / currentPeltier2CAL;
  currentPeltier3Ampere = currentPeltier3 / currentPeltier3CAL;;

  currentPeltierTotalAmpere = currentPeltier1Ampere + currentPeltier2Ampere +
    currentPeltier3Ampere;
```

```
  currentPeltierTotalAmpereSend = (int)(currentPeltierTotalAmpere*1000);


  Outputarray[0] = lowByte(timeSend);
  timeSend = timeSend>>8;
  Outputarray[1] = lowByte(timeSend);
  timeSend = timeSend>>8;
  Outputarray[2] = lowByte(timeSend);
  timeSend = timeSend>>8;
  Outputarray[3] = lowByte(timeSend);

  Outputarray[4] = lowByte(temperature);
  temperature = temperature>>8;
  Outputarray[5] = lowByte(temperature);

  temperaturewantedSend = temperaturewanted;

  Outputarray[6] = lowByte(temperaturewantedSend);
  temperaturewantedSend = temperaturewantedSend>>8;
  Outputarray[7] = lowByte(temperaturewantedSend);

  Outputarray[8] = lowByte(temperaturedifference);
  temperaturedifference = temperaturedifference>>8;
  Outputarray[9] = lowByte(temperaturedifference);

  sensorValueSend = currentPeltierTotalAmpereSend;
  Outputarray[10] = lowByte(sensorValueSend);
  sensorValueSend = sensorValueSend>>8;
  Outputarray[11] = lowByte(sensorValueSend);

  Outputarray[12] = dirbyte;

  outputValueSend = outputValue;
  Outputarray[13] = lowByte(outputValueSend);
  outputValueSend = outputValueSend>>8;
  Outputarray[14] = lowByte(outputValueSend);

  Serial.write(Outputarray, sizeof(Outputarray));
}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                    fuction to determine the ambient temperature via the ntc
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
void getRoomTemperature(){

  roomTemperature_Sum = 0;

  for (roomTemperatureSum_Counter = 0; roomTemperatureSum_Counter <
    roomTemperatureSum_Count; roomTemperatureSum_Counter++) {

  roomTemperature_Sum += analogRead(roomTemperaturePin);

  delay(1);
  }

  roomTemperature = (int)(roomTemperature_Sum / roomTemperatureSum_Count);
  roomTemperatureCelsius = roomTemperature * roomTemperatureCALslope +
    roomTemperatureCALoffset;
  roomTemperatureCelsiusInt = (int)(round(roomTemperatureCelsius*10)*100);


}


void initLEDstep(byte step){

  switch (step) {
    case 4:
    analogWrite(ledPinR, 255);          // rot
    analogWrite(ledPinG, 0);
    analogWrite(ledPinB, 0);
    analogWrite(led2PinR, 0);           // blau stark
    analogWrite(led2PinB, 255);
      break;
    case 9:
    analogWrite(ledPinR, 224);          // organge
    analogWrite(ledPinG, 30);
    analogWrite(ledPinB, 0);
    analogWrite(led2PinR, 0);           // blau mittel
    analogWrite(led2PinB, 120);
      break;
    case 14:
    analogWrite(ledPinR, 174);          // dunkelgelb
    analogWrite(ledPinG, 109);
    analogWrite(ledPinB, 0);
    analogWrite(led2PinR, 0);           // blau schwach
    analogWrite(led2PinB, 23);
```

```
        break;
      case 19:
      analogWrite(ledPinR, 108);             // gruengelb
      analogWrite(ledPinG, 146);
      analogWrite(ledPinB, 0);
      analogWrite(led2PinR, 17);             // rot schwach
      analogWrite(led2PinB, 0);
        break;
      case 24:
      analogWrite(ledPinR, 0);               // gruen
      analogWrite(ledPinG, 255);
      analogWrite(ledPinB, 0);
      analogWrite(led2PinR, 90);             // rot mittel
      analogWrite(led2PinB, 0);
        break;
      case 29:
      analogWrite(ledPinR, 255);             // pink
      analogWrite(ledPinG, 0);
      analogWrite(ledPinB, 180);
      analogWrite(led2PinR, 255);            // rot stark
      analogWrite(led2PinB, 0);
        break;
      case 34:
          analogWrite(ledPinR, 0);           // aus
      analogWrite(ledPinG, 0);
      analogWrite(ledPinB, 0);
      analogWrite(led2PinR, 0);
      analogWrite(led2PinB, 0);
        break;

        default:

      break;
  }

}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~//
//                        fuction for fan control in init
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~//

void initFANstep(byte step){

  switch (step) {
```

```
        case 30:
          digitalWrite(FanBackPin, HIGH);
          break;
        case 33:
          digitalWrite(FanBackPin, LOW);
          break;
        case 35:
          digitalWrite(Fan1Pin, HIGH);
          break;
        case 37:
          digitalWrite(Fan1Pin, LOW);
          digitalWrite(Fan2Pin, HIGH);
          break;
        case 39:
          digitalWrite(Fan2Pin, LOW);
          digitalWrite(Fan3Pin, HIGH);
          break;
        case 41:
          digitalWrite(Fan3Pin, LOW);
          digitalWrite(Fan2Pin, HIGH);
          break;
        case 43:
          digitalWrite(Fan2Pin, LOW);
          digitalWrite(Fan1Pin, HIGH);
          break;
        case 45:
          digitalWrite(Fan1Pin, LOW);
          digitalWrite(Fan2Pin, LOW);
          digitalWrite(Fan3Pin, LOW);
          break;

        default:
          break;
    }

}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//            fuction to print welcome message on 7 segment display
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

void scrollInitMessage(){
```

```
  tempDisplay.setIntensity(0,15);

  for(textCounter = 0; textCounter<48; textCounter++) {

   tempDisplay.setRow(0,2,initMessage[textCounter]);
   tempDisplay.setRow(0,1,initMessage[textCounter+1]);
   tempDisplay.setRow(0,0,initMessage[textCounter+2]);
   initFANstep(textCounter);
   initLEDstep(textCounter);
   delay(delaytime);

  }
  tempDisplay.setIntensity(0,tempDisplayNormalIntensity);
}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//           fuction to parse a number to display it on 7 segments
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

void parseTemp(unsigned int temp) {

  Digit_0_DP = false;
  Digit_1_DP = true;
  Digit_2_DP = false;

  temp = temp / 10;

  Last_Digit = temp % 10;
  if (Last_Digit < 5) Digit_Round = false; else Digit_Round = true;
  temp = temp / 10;

  Digit_0_value = temp % 10;

  if (Digit_Round) {
    if (Digit_0_value == 9) {
      Digit_0_value = 0;
    } else {
      Digit_0_value++;
      Digit_Round = false;
    }
  }
  temp = temp / 10;

  Digit_1_value = temp % 10;
```

```
  if (Digit_Round) {
    if (Digit_1_value == 9) {
      Digit_1_value = 0;
    } else {
      Digit_1_value++;
      Digit_Round = false;
    }
  }
  temp = temp / 10;

  Digit_2_value = temp % 10;

  if (Digit_Round) {
      Digit_2_value++;
      Digit_Round = false;
  }
  Digit_Round = false;

}
```

tempcontrol56.ino

## Source code for the programm to read the control parameters via the USB interface

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                                        include librarys
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <string.h>
#include <conio.h>
#include <stdint.h>
#include <stdarg.h>
#include <string.h>
#include <inttypes.h>

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                                        define constants
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#define ESC 27
#define MAXLINE 100;

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                                        define variables
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

 int verbose = 0;
 char* VIDstr;
 char* PIDstr;
CHAR FileFullPath[] = {"COM7"} ;
DWORD dwError,mode;
BOOL fSuccess;
DCB dcb;
HANDLE hCom;
 FILE* comPortFileHandle;

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                                        fuction prototypes
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

 unsigned mainmenu (void);
```

```
void monitor (HANDLE hCom);
void warten(void);
void strremove(char* source,char ch);

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                                  main fuction
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

int main(int argc, char *argv[]) {


HANDLE keyboard = GetStdHandle(STD_INPUT_HANDLE);


CHAR ComPortFile[] = {"comports.list"} ;
CHAR puffer[100];

CHAR *comPortListe[9];

int i=1;
char *ptr;
unsigned c,d,e,f;
unsigned inputbuffer[10];


    // set keyboard to raw reading.
    if (!GetConsoleMode(keyboard, &mode))
        printf("getting keyboard mode");
    mode &= ~ ENABLE_PROCESSED_INPUT;
    if (!SetConsoleMode(keyboard, mode))
        printf("setting keyboard mode");


hCom = CreateFile( (LPCTSTR)FileFullPath ,
    GENERIC_READ | GENERIC_WRITE,
    0,     // comm devices must be opened w/exclusive-access
    NULL, // no security attributes
    OPEN_EXISTING, // comm devices must use OPEN_EXISTING
    0,     // not overlapped I/O
    NULL  // hTemplate must be NULL for comm devices
    );

if (hCom == INVALID_HANDLE_VALUE)
{
```

```
    dwError = GetLastError();
    printf("Invalid value: %d\r\n", dwError);
    // handle error
}

fSuccess = GetCommState(hCom, &dcb);

if (!fSuccess)
{

    printf("Error 1\n");
    // Handle the error.
}

// Fill in the DCB: baud=9600, 8 data bits, no parity, 1 stop bit.

dcb.BaudRate = 9600;
dcb.ByteSize = 8;
dcb.Parity = NOPARITY;
dcb.StopBits = ONESTOPBIT;

fSuccess = SetCommState(hCom, &dcb);

if (!fSuccess)
{
    printf("Error 2\n");
    // Handle the error.
}

while (c!=27) {

c = mainmenu();

switch (c)

{

       case '1' :

      system("listComPorts.exe > comports.list");

      comPortFileHandle = fopen(ComPortFile, "r");

      i = 1;
```

```
        system("Cls");
        printf("\n\n");

        while(fgets(puffer, 100, comPortFileHandle))
          {

          ptr = strtok(puffer, "-");
                  while(ptr != NULL) {

          printf("%d: ",i);
                  comPortListe[i]=ptr;
                  strremove(comPortListe[i],' ');
                  printf("%s",comPortListe[i]);
                  ptr = strtok(NULL, "-");
                  printf("%s\n",ptr);
                  ptr = strtok(NULL, "-");
                  ptr = strtok(NULL, "-");
                  i++;
          }

        }

        label:

        do f = getch(); while (!((isdigit(f)) || (f!='ESC')));

        if (f-48<i) {
        CloseHandle(hCom);
              printf("%d\n",f-48);
              printf("%s",comPortListe[f-48]);

              hCom = CreateFile( (LPCTSTR)comPortListe[f-48] ,
              GENERIC_READ | GENERIC_WRITE,
              0,      // comm devices must be opened w/exclusive-access
              NULL, // no security attributes
              OPEN_EXISTING, // comm devices must use OPEN_EXISTING
              0,      // not overlapped I/O
              NULL   // hTemplate must be NULL for comm devices
              );

              if (hCom == INVALID_HANDLE_VALUE)
              {
              dwError = GetLastError();
```

```
            printf("Invalid value: %d\r\n", dwError);
            // handle error
            }


      fSuccess = GetCommState(hCom, &dcb);

            if (!fSuccess)
            {

        printf("Error 1\n");
                // Handle the error.
            }

            // Fill in the DCB: baud=9600, 8 data bits, no parity, 1 stop bit.

            dcb.BaudRate = 9600;
            dcb.ByteSize = 8;
            dcb.Parity = NOPARITY;
      dcb.StopBits = ONESTOPBIT;

            fSuccess = SetCommState(hCom, &dcb);

            if (!fSuccess)
            {
        printf("Error 2\n");
        // Handle the error.
            }

            } else goto label;

                          break;
       case '2' :         monitor(hCom);

                          break;
       case '3' :         monitorwrite(hCom);

                          break;
       case '4' :         return(0);
                          break;
       case 'ESC' :       return(0);
                          break;
       default :          break;
```

```
}
}

    CloseHandle(keyboard);
    CloseHandle(hCom);

return 0;
    }

void warten (void) {

while(!(kbhit()));

}


unsigned mainmenu (void) {

unsigned c;

system("Cls");

printf(" ================================== \n");
printf(" # QCM-Array Temperature Control # \n");
printf(" #                                # \n");
printf(" #          programmed by         # \n");
printf(" #                                # \n");
printf(" # Siegfried Hohmann, B.Sc. 2015  # \n");
printf(" ================================== \n");
printf("\n\n");
printf(" 1. Select serial port\n");
printf(" 2. Monitor control process\n");
printf(" 3. Monitor and write data to file\n");
printf(" 4. Exit\n");

do c = getch(); while (!((isdigit(c)) || (c!='ESC')));

return c;

}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                          function to monitor
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
void monitor (HANDLE hCom) {

unsigned c;
unsigned d;
byte inputbuffer[15];

            uint32_t time;
            uint16_t temperature;
            uint16_t temperaturewanted;
            int16_t temperaturedifference;
            uint16_t sensorValue;
            byte dirbyte;
            uint16_t outputValue;


DWORD read, written;

        if (kbhit()) {
            c = getch();
        }

    do {
        // check for data on port and display it on screen.
        ReadFile(hCom, inputbuffer, sizeof(inputbuffer), &read, NULL);
        if ( read ) {

            time = inputbuffer[3];
            time = time<<8;
            time += inputbuffer[2];
            time = time<<8;
            time += inputbuffer[1];
            time = time<<8;
            time += inputbuffer[0];

            temperature = inputbuffer[5];
            temperature = temperature<<8;
            temperature += inputbuffer[4];

            temperaturewanted = inputbuffer[7];
            temperaturewanted = temperaturewanted<<8;
            temperaturewanted += inputbuffer[6];

            temperaturedifference = inputbuffer[9];
```

```
            temperaturedifference = temperaturedifference<<8;
            temperaturedifference += inputbuffer[8];

            sensorValue = inputbuffer[11];
            sensorValue = sensorValue<<8;
            sensorValue += inputbuffer[10];

            dirbyte = inputbuffer[12];

            outputValue = inputbuffer[14];
            outputValue = outputValue<<8;
            outputValue += inputbuffer[13];

            printf("Time: %.1fs Temp: %.3f SP: %.2f Diff: ",time/1000.0,
    temperature/1000.0,temperaturewanted/1000.0);
            if (temperaturedifference >= 0) printf(" ");
            printf("%.3f ",temperaturedifference/1000.0);
            if (dirbyte == 0) { printf("cooling"); } else { printf("heating"); }
            printf(" PWM:");
            if (outputValue < 10) printf(" ");
            if (outputValue < 100) printf(" ");
            printf(" %d ",outputValue);
            printf(" I: %.2fA\n",sensorValue/1000.0);
        }

        if (kbhit()) {
            c = getch();
        }


    } while (c!=27);

}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                      function to monitor and write
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

void monitorwrite (HANDLE hCom) {

unsigned c;
unsigned d;
byte inputbuffer[15];
FILE *fp;
```

```
char filename[50] = {0};

system("Cls");

printf("Enter file name: ");

scanf("%s", filename); //

fp = fopen(filename, "w");
if (fp == NULL) {
printf("Error opening %s for writing.\n", filename);

}

printf("Writing to file %s\n", filename);


            uint32_t time;
            uint16_t temperature;
            uint16_t temperaturewanted;
            int16_t temperaturedifference;
            uint16_t sensorValue;
            byte dirbyte;
            uint16_t outputValue;

DWORD read, written;


    do {

        ReadFile(hCom, inputbuffer, sizeof(inputbuffer), &read, NULL);
        if ( read ) {

            time = inputbuffer[3];
            time = time<<8;
            time += inputbuffer[2];
            time = time<<8;
            time += inputbuffer[1];
            time = time<<8;
            time += inputbuffer[0];

            temperature = inputbuffer[5];
            temperature = temperature<<8;
            temperature += inputbuffer[4];
```

```
            temperaturewanted = inputbuffer[7];
            temperaturewanted = temperaturewanted<<8;
            temperaturewanted += inputbuffer[6];

            temperaturedifference = inputbuffer[9];
            temperaturedifference = temperaturedifference<<8;
            temperaturedifference += inputbuffer[8];

            sensorValue = inputbuffer[11];
            sensorValue = sensorValue<<8;
            sensorValue += inputbuffer[10];

            dirbyte = inputbuffer[12];

            outputValue = inputbuffer[14];
            outputValue = outputValue<<8;
            outputValue += inputbuffer[13];

            printf("Time: %.1fs Temp: %.3f SP: %.2f Diff: ",time/1000.0,temperature/1000.0,temperaturewanted/1000.0);
            if (temperaturedifference >= 0) printf(" ");
            printf("%.3f ",temperaturedifference/1000.0);
            if (dirbyte == 0) { printf("cooling"); } else { printf("heating"); }
            printf(" PWM:");
            if (outputValue < 10) printf(" ");
            if (outputValue < 100) printf(" ");
            printf(" %d ",outputValue);
            printf(" I: %.2fA\n",sensorValue/1000.0);

            fprintf(fp, "%.1f,%.3f,%.2f,%.3f,",time/1000.0,temperature/1000.0,temperaturewanted/1000.0,temperaturedifference/1000.0);
            if (dirbyte == 0) { fprintf(fp, "cooling,"); } else { fprintf(fp, "heating,"); }
            fprintf(fp, "%d,%.2f\n",outputValue,sensorValue/1000.0);

        }

        if (kbhit()) {
            c = getch();
        }

    } while (c!=27);
```

```c
fclose(fp);

}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                          function to remove a string
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

void strremove(char* source,char ch) {
        char* target=source;
        for (;(*target=*source)!=0;source++)
                if (*target!=ch) target++;
}
```

main.c

# Declaration of authorship

I declare in lieu of an oath that the Master Thesis submitted has been produced by me without illegal help from other persons. I state that all passages which have been taken out of publications of all means or un-published material either whole or in part, in words or ideas, have been marked as quotations in the relevant passage. I also confirm that the quotes included show the extent of the original quotes and are marked as such. I know that a false declaration will have legal consequences.

Date Siegfried Hohmann