TECHNISCHE UNIVERSITÄT WIEN

# Master Thesis

## Kinetic Study of High Temperature Thermochemical Heat Storage Materials

performed for the purpose of obtaining the academic degree of master of science

under supervision of

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Franz Winter

and

Projektass. Dipl.-Ing. Markus Hellmuth Deutsch

submitted to the

Faculty of Technical Chemistry

Vienna University of Technology

Institute of Chemical Engineering

by

Viktor Čider, BSc

Matr.Nr.: 1127 577

Hans-Sachs-Gasse 13/7, 1180 Wien

Vienna, on May 18th 2016

# Affidavit

I, Viktor Cider, hereby declare

1. that I am the sole author of the present Master's Thesis, "Kinetic Study of High Temperature Thermochemical Heat Storage Materials", 127 pages, bound, and that I have not used any source or tool other than those referenced or any other illicit aid or tool.

2. that I have not prior to this date submitted this Master's Thesis as an examination paper in any form in Austria or abroad.

Mai 18th, 2016; Vienna

Signature

# Acknowledgment

I would like to thank my parents for supporting me throughout my studies, my supervisors Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Franz Winter and Dipl.-Ing. Markus Deutsch for providing me with the opportunity to work on this project and my colleagues Dipl.-Ing. Dr.techn.Amon Purgar, Dipl.-Ing.Markus Bösenhofer, Dipl.-Ing.Robert Pachler and Dipl.-Ing.Thomas Karel for their advice.

I would further like to thank the Vienna University of Technology for offering this course of studies with a liberal choice of subjects, the students' union and the SAVT for organizing lots of events that made these past 4 years an incredible experience.

And lastly, I'm thanking my workaholic friends Vera Truttmann and Max Geismann for encouraging me to work harder, and all of my friends for providing a healthy counterbalance to my work.

## Abstract

In the field of renewable energies, heat storage is a big issue. Heat can be stored by thermal decomposition of certain solids and released again after storage and/or transport by reaction with a gas.

This thesis deals with the construction and commissioning of a reactor, which shall be viable for studies of such thermochemical reactions up to 1050°C. Furthermore, a programmable logic controller (PLC) was installed, which can automatically execute temperature and gas flow programs. The temperature of the bed and the composition of the off-gas can be viewed in real time and are stored for further evaluation.

The reactor was tested with the $CuO/Cu_2O$ system and returns satisfying results. It will be used for further research into kinetics and cyclic stability.

The focus of this thesis lies on reactions at high temperatures (>600°C) and the realization of the necessary apparatus.



## Kurzfassung

Im Bereich der erneuerbaren Energien ist insbesondere die Speicherung thermischer Energie ein großes Thema. So kann durch thermische Zersetzung von Feststoffen Wärme gespeichert werden, welche nach Lagerung und/oder Transport durch Reaktion mit einem Gas wieder freigesetzt wird.

Im Rahmen dieser Arbeit wurde ein Reaktor konstruiert, an welchem solche thermochemischen Reaktionen bis 1050°C erforscht werden können. Weiters wurde eine speicherprogrammierbare Steuerung (SPS) eingerichtet, mit der automatisiert Temperatur- und Gasdurchflussprofile gefahren werden können. Die Temperatur in der Schüttung und die Abgaszusammensetzung werden in Echtzeit angezeigt und auch zur späteren Auswertung gespeichert.

Der Reaktor wurde mit dem Reaktionssystem $CuO/Cu_2O$ erprobt und liefert zufriendenstellende Ergebnisse. Er wird zukünftig zur Erforschung der Kinetik und Zyklenbeständigkeit eingesetzt werden.

Der Fokus dieser Arbeit wurde auf Reaktionen im Hochtemperaturbereich gesetzt (>600°C), und aufgrund der besonderen Anforderungen an das Material, steht die apparative Umsetzung im Zentrum.

# Table of Contents

# Table of Figures

## Table of Tables

# 1. Introduction

## 1.1. Electricity Generation

Worldwide electricity consumption is quickly rising, having doubled in the last 20 years (figure 1, top). The largest share of the energy demand is covered by fossil fuels, whose consumption has also risen. Renewables do make up 28% of the Organisation for Economic Co-operation and Development (OECD) electricity generation, but this is for the largest part hydroelectricity, as shown in figure 1 [1]. Renewable energies like wind and solar were strongly pursued as a response to the oil crisis of 1973 and have been strongly growing since [2] [3].

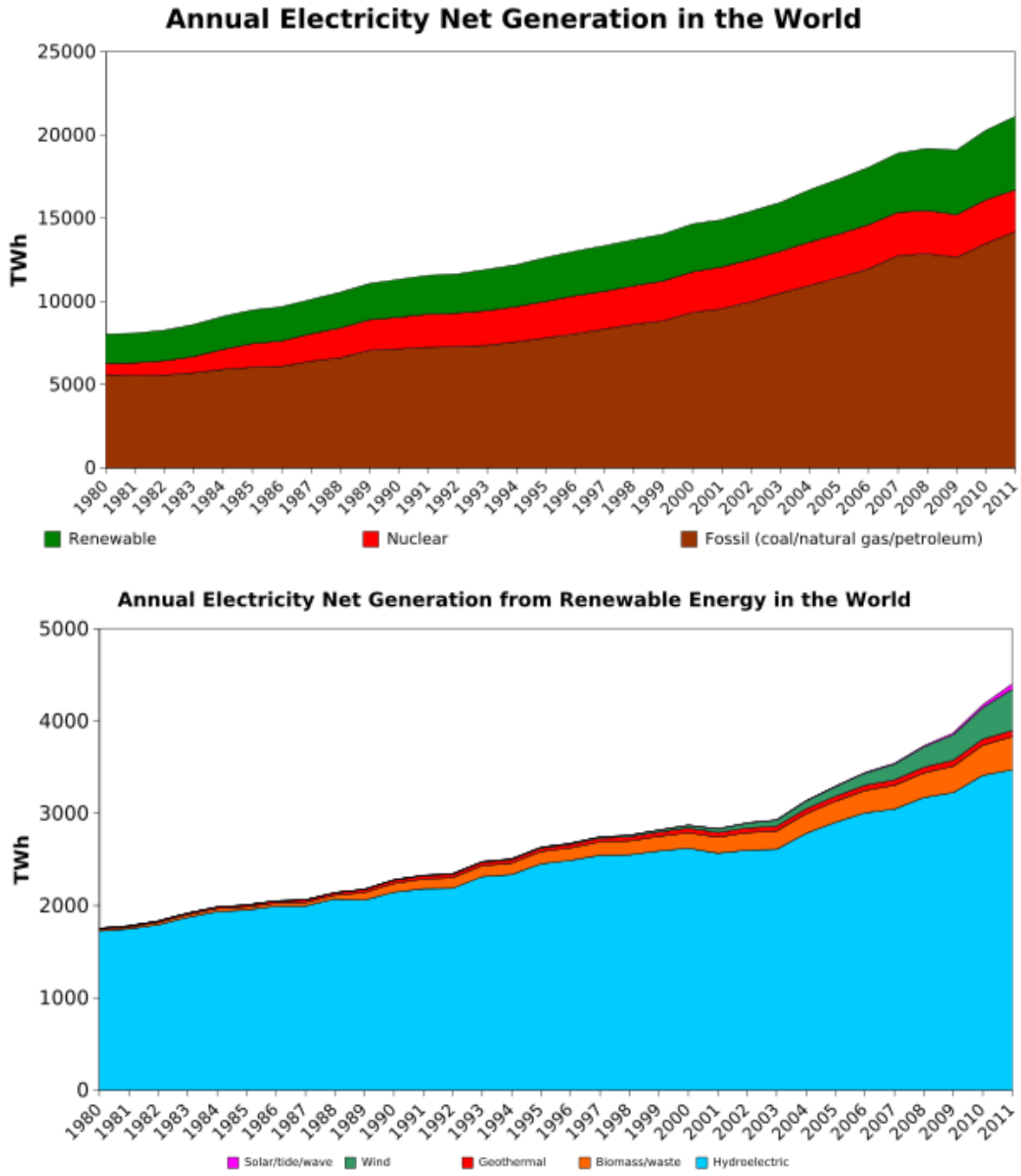



*Figure 1: Annual electricity net generation total (top) and from renewable energy (bottom) in the world*

## 1.2. Energy Demand by Temperature

Industrial processes run at various and sometimes very high temperatures, as is shown in figure 2. While the necessary heat is generally provided by fossil fuels, high temperature thermochemical heat storage materials could be used to replace them in this sector. Some interesting processes in the relevant temperature range (600-1100°C) are the production of ammonia and cement [4], but the heat could also be used to power steam turbines at a higher efficiency than with current heat storage materials, thus increasing the efficiency of the process. [5]

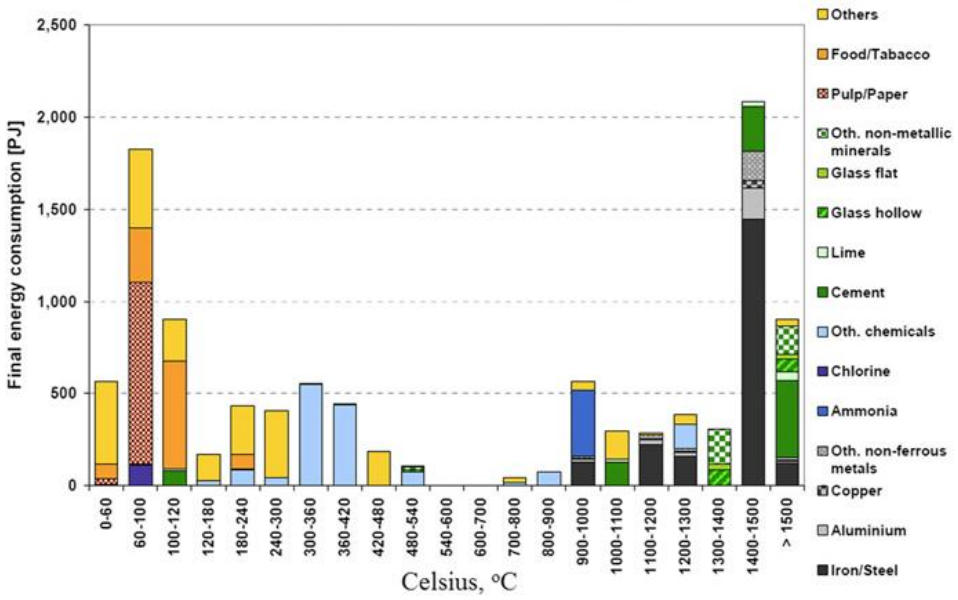



*Figure 2: Heat demand of industrial sector by temperature (EU-27) [4]*

## 1.3. Energy Sources

### 1.3.1. General

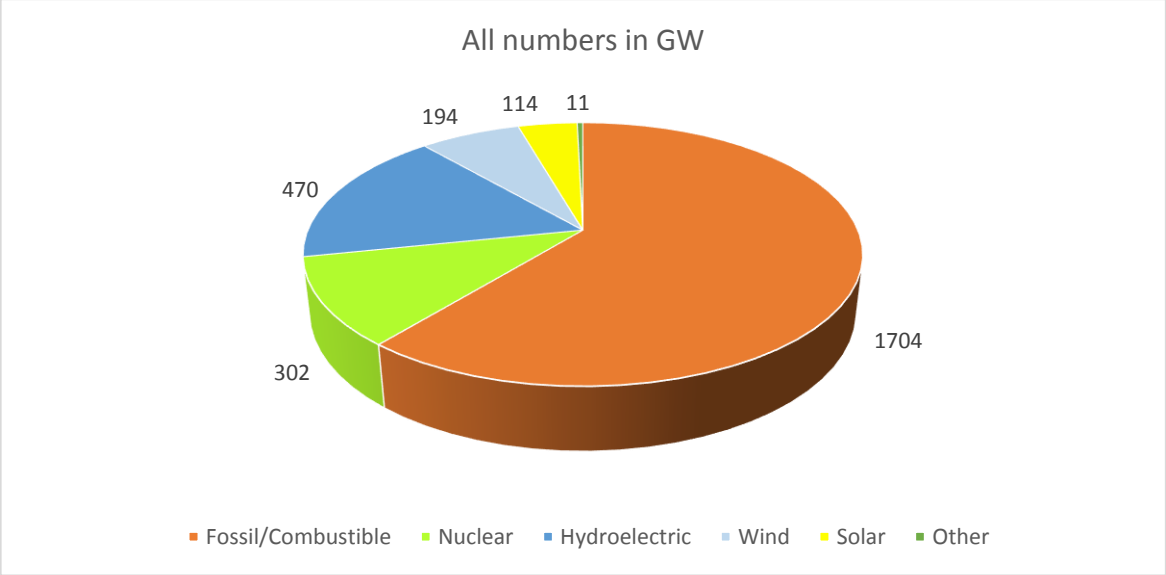



*Figure 3: OECD-wide electricity production capacity by source (2013) [6]*

Figure 3 shows the share of the common sources of energy for electricity production in the Organization for Economic Co-operation and Development (OECD). Most common are fossil and combustible fuels used in thermal power facilities, holding 61% of the total market share. These are also the largest energy sources for electricity generation in most countries, with some of the exceptions being France (nuclear) and Austria (hydroelectricity) [7]. The remainder of the market share is made up of nuclear (11%) and renewable (28%) energies. Of the total 2 794GW capacity, 114GW (4.1%) are generated by solar energy, which can be further divided into 110GW of photovoltaic and 4GW of concentrated solar power. [6]

There are disadvantages to all of these systems:

**Fossil fuel**: Emission of carbon dioxide into the atmosphere and thus contributing to climate change, finite resources, which are also located in politically unstable areas (e.g. Iraq, Syria, Lybia). [8]

**Nuclear fuel**: Highly radioactive waste that has to be safely transported and disposed [9], danger of a nuclear accident with contamination of the environment (e.g. Fukushima, Chernobyl)

**Hydroelectricity**: Possible ecologic impact [10]; limited potential, with 53% of the European potential already being used [1]

**Wind**: Dependence on wind, the energy output can thus vary with an average European capacity factor of 21% between 2004 and 2009 [11]

**Solar**: Dependence on sunshine, thus necessitating a clear sky and daylight, with a capacity factor of 22% in North America (mainly California and Arizona) [12]

Since fossil fuels are unsustainable, nuclear energy is dangerous and toxic, and hydroelectricity is limited, the use of wind, solar and other renewable energies has to be increased. But as these are unreliable due to their reliance on wind/sunshine, ways of storing their energy need to be developed.

### 1.3.2. Wind

Wind power plants take the second largest share in renewable energy after hydroelectricity. They have the advantage of not being reliant on rivers, which poses a limit on hydroelectricity. Their development since 1996 is shown in figure 4.

They are however dependent on wind being present, which is not always the case. Wind energy is thus an intermittent source of energy, with a capacity factor of about 21%. [11]

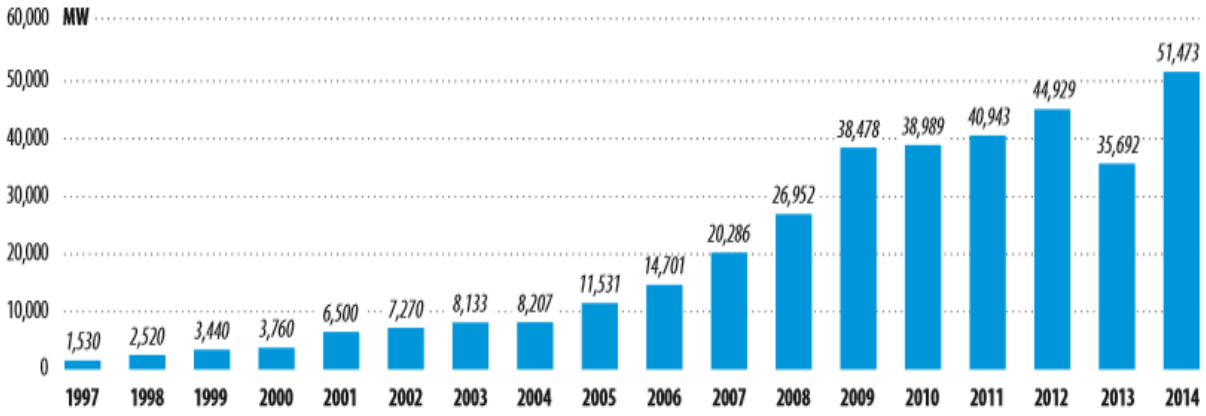



*Figure 4: Global wind power cumulative capacity [13]*

### 1.3.3. Photovoltaics

Photovoltaic (PV) modules directly convert sunlight into electricity with an efficiency of around 4-17%. However, the total efficiency further depends on the capacity factor, which is around 22% in California and Arizona [12]. They have the advantage over CSP of being able to use indirect insolation, thus not necessitating a clear sky. The use of PV has steadily risen in the last years and, as of 2014, is at 178GW (figure 5). [14] [15]

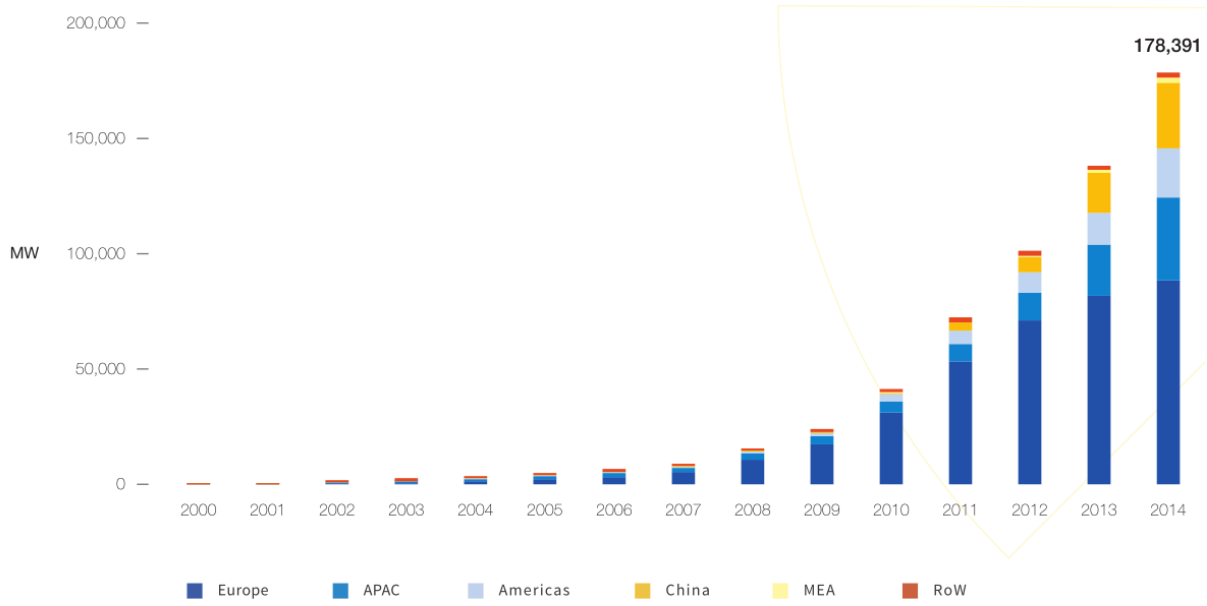



*Figure 5: Development of global cumulative PV capacity in MW since 2000 [15]*

PV has a restriction in that the panels require elements like Tellurium, Gallium, Germanium or Ruthenium, the current reserves of which are too low to cover a significant part of the world energy production with current technology PV. Future developments may overcome this constraint, as pyrite ($FeS_2$) looks promising and contains common elements. [16]

### 1.3.4. Concentrated Solar Power

Concentrated solar power (CSP) plants consist of 4 parts: a concentrator, a receiver, a medium and steam tubes. The concentrator can be either a parabolic trough, focusing the light on a receiving pipe, or it can consist of multiple mirrors pointing at a single receiver. The receiver absorbs as much of the light as possible and transfers it to a fluid, which can be air, water/steam, molten salt or sodium. This medium then transfers the heat to a pipe within which the hot steam for a steam turbine is generated. [12]

There have been some projects with CSP in the 80's, but as shown in figure 6, the technology was left unused until 2007, when it rapidly started rising. Between 2009 and 2014, CSP generation increased by 46% annually on average. [17]

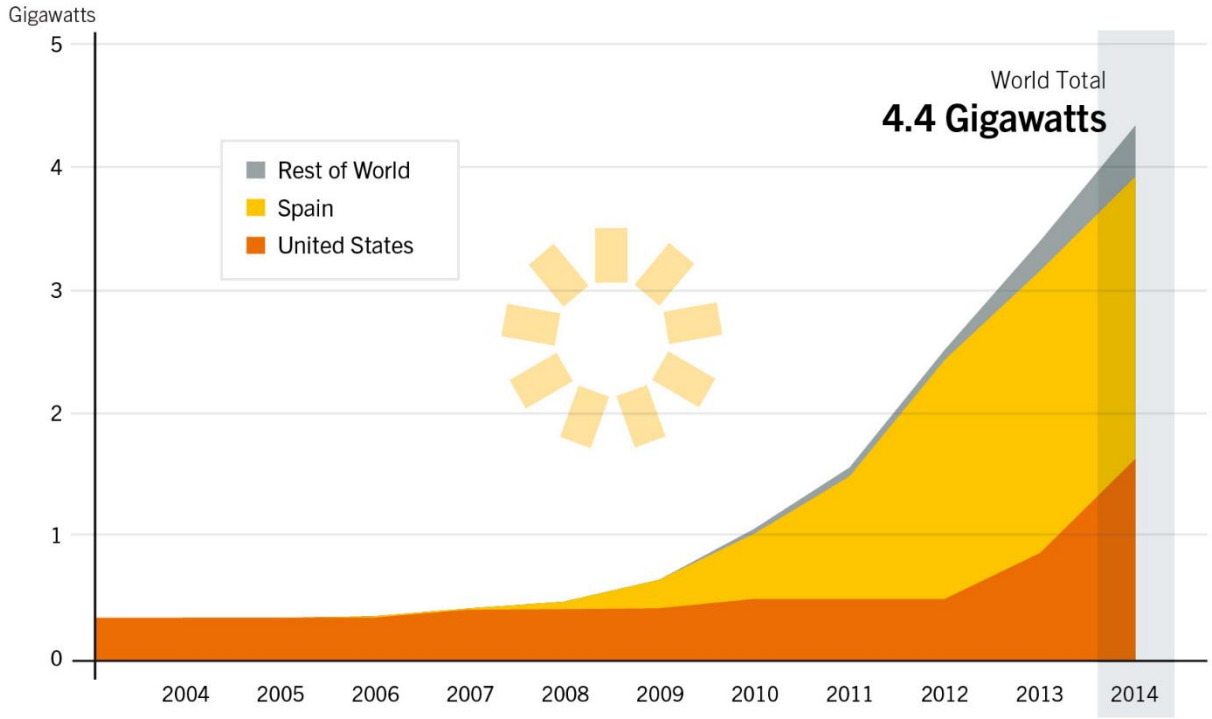



*Figure 6: Global concentrated solar power output by year [17]*

## 1.4. Energy Storage

### 1.4.1. Purpose

Energy storage can be used to balance out times when supply is higher than demand and when demand is higher than supply. This is very important with solar energy, as the power generation peaks at noon, but the power consumption peaks in the evening.

Another possibility is the balancing of summer, when solar power is readily available and power consumption for heating is minimal, and winter, when solar power generation is low and heating power consumption peaks. The surplus energy in summer can be used to charge the heat storage material, which can then be discharged in winter to support the existing power generation facilities.

### 1.4.2. Daily Net Load Curve

Figure 7 shows the net load curve of the Californian power grid for 11 January 2012 and projections for the years 2013 to 2020. The net load curve represents the difference between total electricity demand and variable electricity supply like wind and solar, e.g. sources that cannot be controlled by the operator.

The projected curves show an increase in PV use, which reduces the net load during noon but leaves the remainder of the curve untouched. This demonstrates how much of an advantage the energy storage abilities of CSP really are, as CSP plants can produce electricity throughout the day with sufficient storage.

Increased use of PV poses another problem: the steep ramp from 3PM to 6PM, which are a problem for grid operators as very flexible power plants have to be used to follow this curve. Furthermore, the low electricity demand during noon may lead to overproduction. This problem is nicknamed the "duck curve", due to the 2020 net load curve's resemblance with a duck. [18]

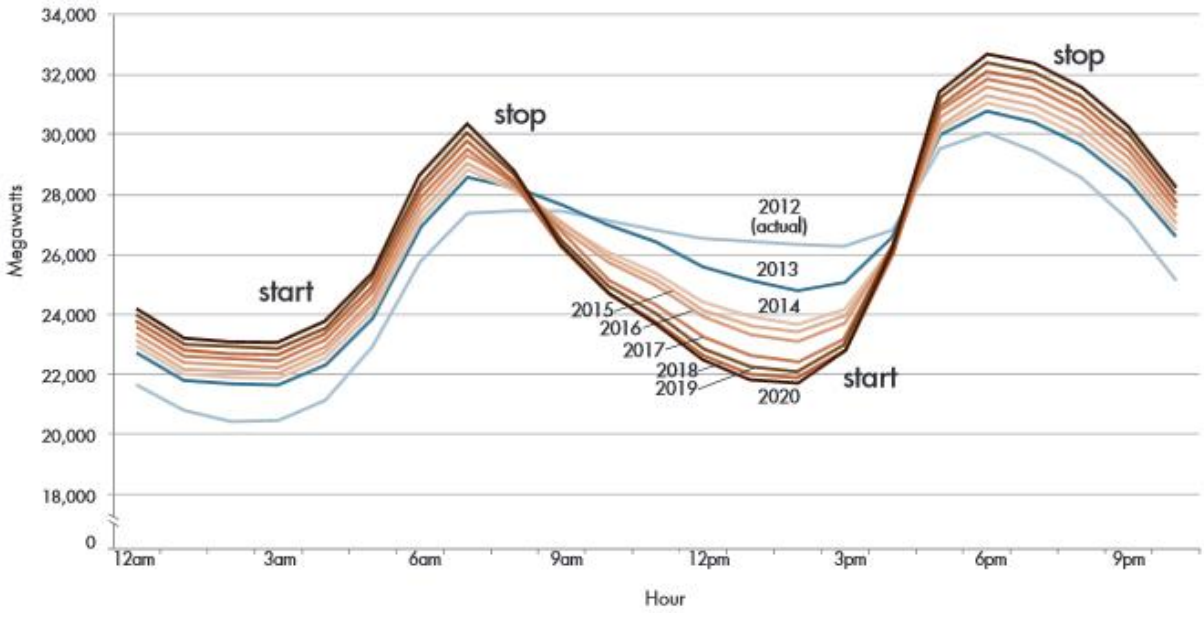



*Figure 7: Daily load curve of the Californian power grid; actual (2012) and projection (2013-2020) [18]*

### 1.4.3. Electrochemical Energy Storage

Energy can be stored in batteries as chemical energy. They have the advantage that they store and produce electricity directly, thus eliminating the need for larger machines like turbines. This makes it possible to use them in small-scale applications like cars, cell phones and artificial pacemakers. There are also large-scale applications for batteries, for example to ensure continuous power supply to factories [19].

### 1.4.4. Mechanical Energy Storage

#### *1.4.4.1. Compressed Air*

In compressed air energy storage, surplus electricity is used to compress air into a cavern, building its pressure up to 60-70bar. During compression, the air heats itself up and has to be cooled. The air can later be released and expanded through a gas turbine to generate electricity again, but it has to be heated up again. Latent heat storage can be used to store the heat from compression to be used at expansion. [20]

In diabatic compressed air energy storage, the air is mixed with fuel and combusted, while in the adiabatic variant the heat from cooling during compression is stored and released at this part of the process. [20]

#### *1.4.4.2. Flywheel*

Flywheels consist of a massive wheel which can be accelerated by a motor, thus storing kinetic energy. When withdrawing energy, the motor acts as a dynamo and transforms the kinetic energy back into electric energy, decelerating the wheel. Modern machines use magnetic bearings for the wheel and operate in a vacuum to reduce friction. Flywheel energy storage is used when high power outputs and short response times are desired, for example in vehicles. [21]

#### *1.4.4.3. Pumped-Storage Hydroelectric*

Pumped-storage hydroelectric power plants consist of an upper and a lower reservoir connected by tunnels or pipelines. Water can be pumped from the lower to the higher reservoir to store energy, usually done during off-peak hours when electricity is cheaper, and generate electricity during peak hours by letting the water flow back down, powering turbines [1]. As of 2012, this makes up more than 99% of energy storage capacity [22].

### 1.4.5. Sensible Heat Storage

In sensible heat storage, heat is stored by heating a material up and keeping it hot until the energy is extracted again. The storable heat per unit of volume $Q_V$ can be expressed as the product of the heat capacity c, the density $\rho$ and the temperature difference $\Delta T$, assuming the heat capacity and the density are constant over the given temperature range (equation 1.1).

$$Q_V = c \cdot \rho \cdot \Delta T \left[\frac{J}{m^3}\right] \tag{1.1}$$

This method of storage has the downside that the heat dissipates into the surroundings over time. The energy density is generally low, which makes the size of the required container rather big, in turn increasing the surface and thus the heat losses. This also makes the material unviable for transportation.

An example of this are common household water storage tanks, which can be heated up during the night - when energy is cheap - and provide warm water throughout the day.

Water limits the operating temperature to 100°C, so there are several other liquids, such as synthetic oils and molten salts that can be used instead. These are being used in concentrated solar power plants. Parabolic trough plants operating between 350 and 400°C use synthetic oil for heat exchange and storage, while solar towers with their higher operating temperatures use molten salt, limiting them to 565°C. The plant is designed to collect more heat than can be used for steam generation and the surplus heat is stored as sensible heat in tanks. [12] [23]

### 1.4.6. Latent Heat Storage

Latent heat storage also heats up a material, but uses a phase transition to store additional energy. This has the advantage that a lot of energy is generated at a constant temperature. On the other hand, the material has to be stored above its phase transition temperature, and thus slowly dissipates heat into its surroundings. Two common groups of latent heat storage materials are paraffins and inorganic salt hydrates.

As an example, latent heat storage materials with a phase transition temperature of around 20°C can be put into a building's walls where during night they solidify and emit heat, keeping the building warm. During daytime, the opposite happens: they melt and absorb heat, thus keeping the building cool. The latter effect, cool storage, is also often used to cool drinks – with ice cubes. [24]

### 1.4.7. Thermochemical Heat Storage

In thermochemical heat storage, the storage material (A) is heated up to the point where it decomposes into its components B and C, with one of them being a gas (equation 1.2). $\Delta H$ represents the enthalpy used up by the reaction and $\nu$ denotes the stoichiometric coefficient of the respective species.

$$\nu_A\, A + \Delta H \rightarrow \nu_B\, B + \nu_C\, C \tag{1.2}$$

Afterwards, it is put under an inert atmosphere and cooled down or discharged the same way as a sensible heat storage material, while preserving the stored chemical energy. It can then be stored at room temperature indefinitely or transported with relative ease. Equation 1.3 shows the calculation of the storable heat per unit of volume $Q_V$ with the density $\rho$, the reaction enthalpy $\Delta H$ and the molar mass M.

$$Q_V = \frac{\rho \cdot \Delta H}{M} \left[\frac{J}{m^3}\right] \tag{1.3}$$

### 1.4.8. Comparison

There are several aspects to these heat storage materials that are compared in table 1. The first and biggest advantage of thermochemical heat storage is the energy density, being able to store 10 times as much heat in the same volume as sensible heat storage. It simplifies storage as it can be stored losslessly at ambient temperature, removing the need for thermal insulation, whereas the other systems have to stay heated and thus suffer thermal losses. But despite these advantages, thermochemical heat storage is not as well researched as the other technologies.

*Table 1: Comparison between sensible, latent and thermochemical heat storage systems [25]*

| | **Sensible heat storage system** | **Latent heat storage system** | **Thermochemical storage system** |
|---|---|---|---|
| **Energy density** | | | |
| Volumetric density | Small ~50 kWh $m^{-3}$ of material | Medium ~100 kWh $m^{-3}$ of material | High~500 kWh $m^{-3}$ of reactant |
| Gravimetric density | Small~0.02–0.03 kWh $kg^{-1}$ of material | Medium ~0.05–0.1 kWh $kg^{-1}$ of material | High ~0.5–1 kWh $kg^{-1}$ of reactant |
| **Storage temperature** | Charging step temperature | Charging step temperature | Ambient temperature |
| **Storage period** | Limited (thermal losses) | Limited (thermal losses) | Theoretically unlimited |
| **Transport** | Small distance | Small distance | Distance theoretically unlimited |
| **Maturity** | Industrial scale | Pilot scale | Laboratory scale |
| **Technology** | Simple | Medium | Complex |

## 1.5. Motivation

There is a distinctive trend towards the use of renewable energies instead of fossil fuels. This is necessary due to the limitation in fossil resources and the adverse effect they have on the climate. However, renewable energies like wind and solar energy have the downside of being non-continuous.

To abridge the disruptions, there are methods of storing this energy, but the common methods like molten salt technology rely on sensible heat storage, which has a low energy density, needs elaborate containers and yet slowly dissipates heat.

Thermochemical heat storage at high temperatures can potentially eliminate these problems and should be examined as an alternate technology. However, little is known about these materials, which is the reason that this study was performed.

## 1.6. Aim

The aim is to identify and study new materials which could be used in thermochemical heat storage applications. These should have a high energy density, high cycle durability, low toxicity and be inexpensive. For this purpose, a new reactor with appropriate instrumentation and automation shall be commissioned.

The chosen material shall then be tested for cycle stability, reaction duration and yield.

## 1.7. Comparison of Materials

Thermochemical heat storage materials can be divided into categories based on their reaction system. Several important categories are:

- oxidation/decomposition (REDOX)
- carbonation/calcination
- hydration/dehydration

As the analyzer of the test rig can measure oxygen and carbon dioxide content but not water content, only the first two categories can be studied here. For oxidation/reduction reactions, a metal oxide is needed which can change its oxidation state, such as manganese, chromium, cobalt, iron or copper. Another option is having a metal with a constant oxidation state and changing it between an oxide and a peroxide, which can be done with barium (equation 1.4)

$$BaO_2 \rightleftarrows BaO + \frac{1}{2}O_2 \tag{1.4}$$

Carbonation/calcination systems also keep the oxidation state unchanged and change between the oxide and the carbonate. A well-studied reaction is the one involving calcium, seen in equation 1.5.

$$CaCO_3 \rightleftarrows CaO + CO_2 \tag{1.5}$$

Many of the materials are already well studied, but have a heat release temperature of below 900°C. Copper oxide was found to be an interesting material with little research done and an energy release temperature of approximately 1020°C.

# 2. Theoretical Fundamentals

## 2.1. Thermodynamics

During a chemical reaction, heat is either released or absorbed. If the system in which the reaction occurs is isobaric (i.e. the pressure is constant), this heat is equal to the change in enthalpy of the system. The reaction enthalpy $\Delta_r H$ can be calculated as the difference between the sums of molar enthalpies $H_m$ times the stoichiometric factor v of the products and the reactants (equation 2.1).

$$\Delta_r H = \sum_{Products} \nu_i H_{m,i} - \sum_{Reactants} \nu_i H_{m,i} \tag{2.1}$$

If $\Delta_r H$ is negative, then heat is released and the reaction is called "exothermic", if it is positive, heat is absorbed and the reaction is "endothermic".

The thermodynamic equilibrium is a state of a reaction system when the forward and backward reaction occur at the same rate. It is dependent on the Gibbs energy, which determines in which direction the reaction is going. The Gibbs energy $\Delta_r G$ in turn depends on the reaction enthalpy $\Delta_r H$, the temperature $T$ and the difference in entropy $\Delta_r S$ (equation 2.2).

$$\Delta_r G = \Delta_r H - T\Delta_r S \tag{2.2}$$

Every system seeks to minimize its Gibbs Enthalpy. When plotted against the extent of reaction $\xi$, the Gibbs Enthalpy is convex and has a minimum, which the system tries to reach. At this point, the sums of chemical potentials on the two sides are equal to each other and no further reaction occurs (equations 2.3 and 2.4).

$$\Delta_r G = \left(\frac{\partial G}{\partial \xi}\right)_{p,T} = \sum_{Products} \nu_i \mu_i - \sum_{Reactants} \nu_i \mu_i = 0 \tag{2.3}$$

$$\sum_{Products} \nu_i \mu_i = \sum_{Reactants} \nu_i \mu_i \tag{2.4}$$

The equilibrium is however dependent on the temperature and pressure of the system. This can be described with Le Chatelier's principle, which states that

> *A system at equilibrium, when subjected to a disturbance, responds in a way that tends to minimize the effect of the disturbance.*
>
> - *Henri Louis Le Chatelier, Translation from Atkins' Physical Chemistry*

This means that increasing the partial pressure of our reactant gas increases the desire of the solid phase to bind the gas and thus reduce its pressure again. Similarly, if the temperature of the system is increased (at constant pressure) then endothermic reactions will be favored, as they absorb heat. More precisely, this can be expressed with the van't Hoff equation (2.5) which relates the natural logarithm of the equilibrium constant K derived by temperature T with the standard reaction enthalpy $\Delta_r H^{\ominus}$ and the gas constant R.

$$\frac{d \ln K}{d \frac{1}{T}} = -\frac{\Delta_r H^{\ominus}}{R} \tag{2.5}$$

If the Gibbs energy is known, the equilibrium constant can be calculated, as seen in equation 2.6.

$$K = e^{-\frac{\Delta_r G}{RT}} \tag{2.6}$$

The equilibrium constant describes the relation of activities of products and reactants to each other in thermodynamic equilibrium. A reaction such as 2.7 can then be described with equation 2.8.

$$\nu_A A(s) \rightleftarrows \nu_B B(s) + \nu_C C(g) \tag{2.7}$$

$$K = \frac{(a_B)^{\nu_B} \cdot (a_C)^{\nu_C}}{(a_A)^{\nu_A}} \tag{2.8}$$

Solid components generally have an activity of one, so they can be omitted in the equation. In a gas-solid reaction system that is present in thermochemical heat storage (eq. 2.9), there is only one gaseous component, whose activity is equal to the equilibrium constant. The dimensionless activity can also be expressed as the ratio of the partial pressure $p_C$ to $p^{\ominus}$, which is the standard pressure of 1bar, yielding equation 2.10.

$$A(s) \rightleftarrows B(s) + C(g) \tag{2.9}$$

$$K = \frac{p_C}{p^{\ominus}} \tag{2.10}$$

## 2.2. Kinetics

### 2.2.1. Models

The conversion rate is influenced by size and porosity of the particle, and the temperature, residence time and gas composition inside the reactor.

There are several models for the reaction of solid/gas-reactions, such as

- Shrinking Particle Model
- Shrinking Core Model
- Progressive Conversion Model

Furthermore, there are multiple steps in the reaction.

The **Shrinking Particle Model** assumes that the reaction happens on the surface of the particle, traveling inwards while ablating the outer product layer. In effect, this means a shrinking sphere that vanishes at full conversion. The steps of the reaction are (1) diffusion to the particle surface, (2) adsorption, reaction and desorption, and (3) diffusion from the particle surface.

The **Shrinking Core Model** a porous particle with the product staying attached to it. This creates a barrier between the gas phase and the unreacted core of the particle, which adds two more reaction steps to the ones from the shrinking particle model: diffusion from the particle surface through the product layer to the core (between 1 and 2) and diffusion of the product out again (between 2 and 3).

The **Progressive Conversion Model** assumes a homogenous reaction rate throughout the particle, without separate reactant/product zones, preserving the rough size of the particle.

In the mentioned thermochemical heat storage reactions, there is only one gaseous component, depending on the direction either as a reactant or product. This eliminates the steps before or after the reaction.

Usually, there is a step that is distinctly slower than the others, determining the overall reaction speed. This step is called the rate-determining step.

### 2.2.2. Calculation

Octave Levenspiel demonstrates the calculation of the reaction time of such particles in [26], which is reproduced in this chapter.

#### *2.2.2.1. Chemical Reaction Controls*

If the chemical reaction is the rate-determining step, the concentration of the reactant at the surface of the particle or core is assumed as equal to the bulk of the gas phase. All steps other than the chemical reaction are assumed to pose no resistance, thus it doesn't matter whether a product layer is present or not.

Equation 2.11 shows the relation between the change in the amount of the solid and gaseous reactant, $dN_B$ and $dN_A$ respectively, which are related with the stoichiometric factor $b$. This can further be related to the molar density of the solid $\rho_B$ and the change in volume $dV$. Using the formula for the calculation of a volume, we can expand this into the 4th expression in eq. 2.11 which uses the radius of the unreacted core $r_C$. Applying the differential leaves us with the simplified 5th expression.

$$-dN_B = -bdN_A = -\rho_B dV = -\rho_B d\left(\frac{4}{3}\pi r_C^3\right) = -4\pi\rho_B r_C^2 dr_C \tag{2.11}$$

Equation 2.12 relates the rate of change in amount of the solid reactant, $\frac{dN_B}{dt}$, divided by the surface area of the unreacted core, $4\pi r_C^2$, with the rate of change in amount of the gaseous reactant A divided by the surface area and multiplied with the stoichiometric factor $b$. The factors other than $b$ can be reduced to the reaction rate $k''$ and the concentration of the gaseous reactant in the gas phase $C_{Ag}$.

$$-\frac{1}{4\pi r_C^2}\frac{dN_B}{dt} = -\frac{b}{4\pi r_C^2}\frac{dN_A}{dt} = bk''C_{Ag} \tag{2.12}$$

Entering equation 2.11 into 2.12 gives us equation 2.13, which gives a linear function for the change in core radius. Integrating eq. 2.13 yields eq. 2.14.

$$-\frac{1}{4\pi r_C^2}\rho_B 4\pi r_C^2\frac{dr_C}{dt} = -\rho_B\frac{dr_C}{dt} = bk''C_{Ag} \tag{2.13}$$

$$-\rho_B\int_R^{r_C} dr_C = bk''C_{Ag}\int_0^t dt \tag{2.14}$$

Rearranging the terms results in equation 2.15 which gives a clear function for the time at any core radius. As the reaction ends when there is no more solid reactant left, i.e. the core radius being 0, we can calculate the reaction time with eq. 2.16.

$$t = \frac{\rho_B}{bk'' C_{Ag}}(R - r_C) \tag{2.15}$$

$$\tau = \frac{\rho_B R}{bk'' C_{Ag}} \tag{2.16}$$

Dividing eq. 2.15 by 2.16 results in eq. 2.17, which gives the progress of the core radius as a function of the reaction progress.

$$\frac{t}{\tau} = 1 - \frac{r_C}{R} \tag{2.17}$$

$$1 - X_B = \left(\frac{volume\ of\ unreacted\ core}{total\ volume\ of\ particle}\right) = \frac{\frac{4}{3}\pi r_c^3}{\frac{4}{3}\pi R^3} = \left(\frac{r_c}{R}\right)^3 \tag{2.18}$$

$$\frac{r_c}{R} = (1 - X_B)^{\frac{1}{3}} \tag{2.19}$$

Using equations 2.18 and 2.19, which relate the conversion to the core radius, we can transform eq. 2.17 into eq. 2.20, which relates the time with the conversion.

$$\frac{t}{\tau} = 1 - (1 - X_B)^{\frac{1}{3}} \tag{2.20}$$

#### 2.2.2.2. Gas Diffusion Controls (with Product Layer)

For the next model, we assume that the diffusion through the gas layer around the particle is the controlling step. As the radius of the particle is constant (due to the product layer formation), the gas layer also stays the same. We look at the change of substance per unit of time $\frac{dN_B}{dt}$ in relation to the constant external surface $S_{ex}$ (equation 2.21). $dN_B$ can be exchanged with $b \cdot dN_A$, b being the stoichiometric coefficient. Furthermore we express the outer surface, using the formula for the surface of a sphere, with R being the radius of the particle including the product layer. The resulting expression is equal to the product of b, the constant mass transfer coefficient $k_g$ and the difference in concentration of A between the bulk of the gas and the solid surface $(C_{Ag} - C_{As})$. As we are assuming that the reaction happens immediately, the concentration on the surface is zero; the term can thus be reduced.

$$-\frac{1}{S_{ex}}\frac{dN_B}{dt} = -\frac{1}{4\pi R^2}\frac{dN_B}{dt} = -\frac{b}{4\pi R^2}\frac{dN_A}{dt} = bk_g(C_{Ag} - C_{As}) = bk_g C_{Ag} = constant \tag{2.21}$$

The amount of substance B can be related to the molar density $\rho_B$ and the volume V, as seen in equation 2.22.

$$N_B = \rho_B V = (\frac{moles\ B}{m^3 solid})(m^3 solid) \tag{2.22}$$

Differentiating $N_B$ using equation 2.22 and the formula for volume yields equation 2.23, $r_c$ being the radius of the unreacted core.

$$-dN_B = -bdN_A = -\rho_B dV = -\rho_B d(\frac{4}{3}\pi r_c^3) = -4\pi\rho_B r_c^2 dr_c \tag{2.23}$$

Equations 2.21 and 2.23 can be combined to get equation 2.24. Rearranging the equation and integrating yields equations 2.25 and 2.26.

$$-\frac{1}{S_{ex}}\frac{dN_B}{dt} = -\frac{\rho_B r_c^2}{R^2}\frac{dr_c}{dt} = bk_g C_{Ag} \tag{2.24}$$

$$-\frac{\rho_B}{R^2}\int_R^{r_C} r_C^2\, dr_C = bk_g C_{Ag}\int_0^t dt \tag{2.25}$$

$$t = \frac{\rho_B R}{3bk_g C_{Ag}}\left[1 - \left(\frac{r_C}{R}\right)^3\right] \tag{2.26}$$

Finally, we can express the total reaction duration by setting the final core radius zero, resulting in equation 2.27.

$$\tau = \frac{\rho_B R}{3bk_g C_{Ag}} \tag{2.27}$$

If we divide eq 2.26 by eq. 2.27, we get a relation between the time and the core radius (eq. 2.28). This can be transformed with eq. 2.18 into eq. 2.29, relating time and conversion.

$$\frac{t}{\tau} = 1 - \left(\frac{r_c}{R}\right)^3 \tag{2.28}$$

$$\frac{t}{\tau} = 1 - \left(\frac{r_c}{R}\right)^3 = X_B \tag{2.29}$$

#### 2.2.2.3. Gas Diffusion Controls (without Product Layer)

Gas diffusion without a product layer uses essentially the same equations as with a product layer (chapter 2.2.2.2), but has to make some adjustments for the particle radius R and $k_g$, which isn't constant anymore. The Frössling correlation (equation 2.30) puts the mass transfer coefficient $k_g$ in relation with the particle diameter $d_p$, the mole fraction $y$ and the particle/gas speed u. In our calculations, we're assuming small particles with low gas speed, which makes the second term of the right side of the equation negligible, yielding equation 2.31 and making $k_g$ inversely proportional to $d_p$.

$$\frac{k_g d_p y}{D} = 2 + 0.6(Sc)^{1/3}(Re)^{1/2} = 2 + 0.6\left(\frac{\mu}{\rho D}\right)^{1/3}\left(\frac{d_p u \rho}{\mu}\right)^{1/2} \tag{2.30}$$

$$\frac{k_g d_p y}{D} = 2 \tag{2.31}$$

Rearranging equation 2.30 yields equation 2.31, expressing $k_g$ with the diffusion coefficient D and the particle radius R, and assuming a mole fraction of 1.

$$k_g = \frac{2D}{d_p} = \frac{D}{R} \tag{2.32}$$

If we take equations 2.33 and 2.34, which are essentially equations 2.21 and 2.23 but consider the absence of a product layer, and use equation 2.32 to express $k_g$, we get a relation between the radius and the time which we can integrate (equation 2.35).

$$-\frac{1}{S_{ex}}\frac{dN_B}{dt} = \frac{\rho_B 4\pi R^2}{4\pi R^2}\frac{dR}{dt} = -\rho_B \frac{dR}{dt} = b k_g C_{Ag} \tag{2.33}$$

$$dN_B = \rho_B dV = 4\pi\rho_B R^2 dR \tag{2.34}$$

$$\int_{R_0}^{R} R dR = \frac{bC_{Ag}D}{\rho_B}\int_0^t dt \tag{2.35}$$

Evaluating equation 2.35 yields equation 2.36, and considering that the radius at the end of the reaction is zero, we get equation 2.37 for the total reaction duration.

$$t = \frac{\rho_B R_0^2}{2bC_{Ag}D}\left[1 - \left(\frac{R}{R_0}\right)^2\right] \tag{2.36}$$

$$\tau = \frac{\rho_B R_0^2}{2bC_{Ag}D} \tag{2.37}$$

Equation 2.38 is the result of dividing eq. 2.36 by eq. 2.37, and displays the relation between time and conversion.

$$\frac{t}{\tau} = 1 - \left(\frac{R}{R_0}\right)^2 = 1 - (1 - X_B)^{\frac{2}{3}} \tag{2.38}$$

#### 2.2.2.4. Diffusion of Gaseous Reactant through the Product Layer Controls

In this chapter we will be looking at the kinetics of a particle with a product layer, where the diffusion through the product layer is controlling the overall speed. In equation 2.39 we relate the transfer of the gaseous reactant A to the flux of A through the product layer $Q_A$ at any radius r. At the outer rim of the product layer, we have flux $Q_{As}$, on the inner boundary we have $Q_{Ac}$. All these terms are constant.

$$-\frac{dN_A}{dt} = 4\pi r^2 Q_A = 4\pi R^2 Q_{As} = 4\pi r_c^2 Q_{Ac} = constant \tag{2.39}$$

Using Fick's first law (equation 2.40) in equation 2.39 yields equation 2.41.

$$Q_A = D_e \frac{dC_A}{dr} \tag{2.40}$$

$$-\frac{N_A}{dt} = 4\pi r^2 D_e \frac{dC_A}{dr} = constant \tag{2.41}$$

Upon integration of equation 2.41 with the radii and concentrations for the inner and outer product layer boundaries (equation 2.42) we get equation 2.43.

$$-\frac{dN_A}{dt}\int_R^{r_c} \frac{dr}{r^2} = 4\pi D_e \int_{C_{Ag}=C_{As}}^{C_{Ac}=0} dC_A \tag{2.42}$$

$$-\frac{dN_A}{dt}\left(\frac{1}{r_C} - \frac{1}{R}\right) = 4\pi D_e C_{Ag} \tag{2.43}$$

Using equation 2.23 to replace $N_A$ with $r_C$ results in equation 2.44, which we can integrate to yield equation 2.45.

$$-\rho_B \int_{r_C=R}^{r_C} \left(\frac{1}{r_C} - \frac{1}{R}\right) r_C^2 dr_C = bD_e C_{Ag} \int_0^t dt \tag{2.44}$$

$$t = \frac{\rho_B R^2}{6bD_e C_{Ag}}\left[1 - 3\left(\frac{r_C}{R}\right)^2 + 2\left(\frac{r_C}{R}\right)^3\right] \tag{2.45}$$

Setting the final radius to zero results in the total reaction duration $\tau$ in 2.46. Dividing eq. 2.45 and 2.46 results in 2.47, which, using eq (2.19), yields eq. 2.48.

$$\tau = \frac{\rho_B R^2}{6bD_e C_{Ag}} \tag{2.46}$$

$$\frac{t}{\tau} = 1 - 3\left(\frac{r_C}{R}\right)^2 + 2\left(\frac{r_C}{R}\right)^3 \tag{2.47}$$

$$\frac{t}{\tau} = 1 - 3(1 - X_B)^{\frac{2}{3}} + 2(1 - X_B) \tag{2.48}$$

### 2.2.3. Choosing a Model

These 4 models produce distinctly different time-conversion-curves. If the premise of a spherical particle is fulfilled, the oxygen curve of the reactor could be evaluated to reveal which model applies to the given particle.

*Table 2: Relations of time and conversion based on various possible limiting steps*

| | |
|---|---|
| Chemical Reaction | $\frac{t}{\tau} = 1 - (1 - X_B)^{\frac{1}{3}}$ |
| Gas Diffusion with Product | $\frac{t}{\tau} = X_B$ |
| Gas Diffusion without Product | $\frac{t}{\tau} = 1 - (1 - X_B)^{\frac{2}{3}}$ |
| Ash Diffusion | $\frac{t}{\tau} = 1 - 3(1 - X_B)^{\frac{2}{3}} + 2(1 - X_B)$ |

As we can see in table 2, the possible limiting steps produce distinctly different relations between time and conversion. The time-conversion-curves would then look as given in figure 8.

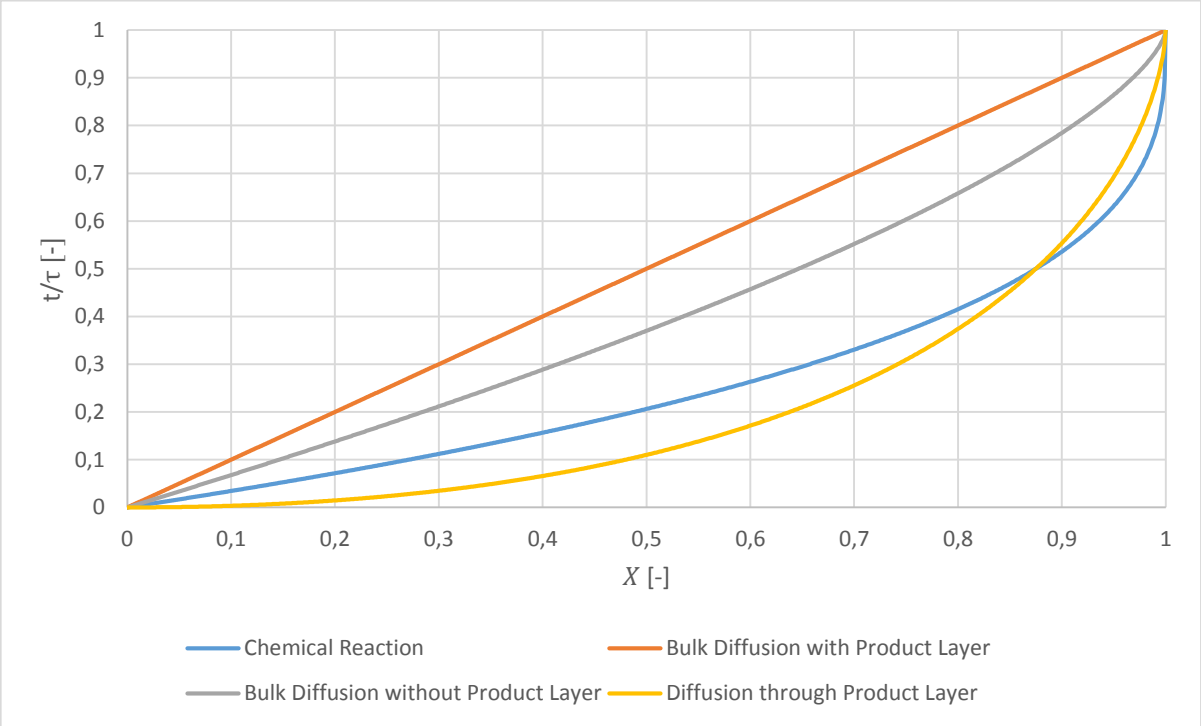



*Figure 8: Comparison of time-conversion-curves based on limiting step*

## 2.3. Yield

The yield η describes how much of the available reactants actually reacted relative to the theoretical maximum, as is sown in equation 2.49 with $m_0$ being the initial weight, $m_{th}$ being the theoretical weight at full conversion and $m_{is}$ being the actual weight at the end of the reaction. The yield directly influences the energy density of the material and thus has a big impact on the economic feasibility of later applications.

$$\eta = \frac{m_{is} - m_0}{m_{th} - m_0} \tag{2.49}$$

In cyclic reactions like this, there is a yield for the reactions in both directions, which are both important.

An alternative method for the calculation of the yield is to calculate it from the amount of oxygen absorbed or emitted from the sample. If we look at a reaction which can be described with equation 2.7, we can calculate the yield with equation 2.50, where $\Delta n_C$ is the amount of gaseous reactant absorbed or emitted and $n_{A,0}$ is the amount of A that would be present if all of the solid was in the state of A (the calculation can also be performed with B in place of A). $\nu$ denotes the respective stoichiometric coefficient.

$$\eta = \frac{\nu_C \cdot \Delta n_C}{\nu_A \cdot n_{A,0}} \tag{2.50}$$

This method is less robust than the gravimetric one above, as it is susceptible to leakage and necessitates knowledge of the dead volume of the apparatus.

## 2.4. System CuO/$Cu_2O$

The CuO/$Cu_2O$ reaction system is a redox reaction, in which copper switches between the oxidation states +I and +II (equation 2.51). To balance this out, oxygen is bound or released.

$$2CuO \rightleftarrows Cu_2O + \frac{1}{2}O_2 \quad [\Delta H_{950°C} = +130\frac{kJ}{mol}] \tag{2.51}$$

Due to the increase in entropy when a gas is released, the equilibrium of this reaction shifts to the right side when the temperature is increased. Every temperature can be assigned a partial oxygen pressure, below which $Cu_2O$ and above which CuO is stable.

During storage of the charged material, an undesired reaction may occur in moist air: the formation of copper hydroxide (equation 2.52).

$$Cu_2O + 2H_2O + \frac{1}{2}O_2 \rightarrow 2Cu(OH)_2 \tag{2.52}$$

This discharges the material and thus has to be avoided. In extreme cases, the generated heat might even pose a danger to the storage.

When evaluating thermochemical heat storage materials, important side reactions to keep in mind are the formation of hydroxides and carbonates with air moisture and carbon dioxide. With copper oxide, this is not an issue because both decompose below 300°C.

The value of the Gibbs energy for the reaction can be calculated with the help of heat capacity polynomials. One program that does that and was used in this thesis is HSC Chemistry. It can calculate a stability diagram, which is a T-p-diagram (p being the partial pressure of oxygen) showing the areas where either of the phases (CuO or $Cu_2O$) is stable, separated by the equilibrium line where the thermodynamic potential of both phases is equal (figure 9). The y-axis of the graph is a logarithmic scale of the partial pressure of oxygen divided by bars.

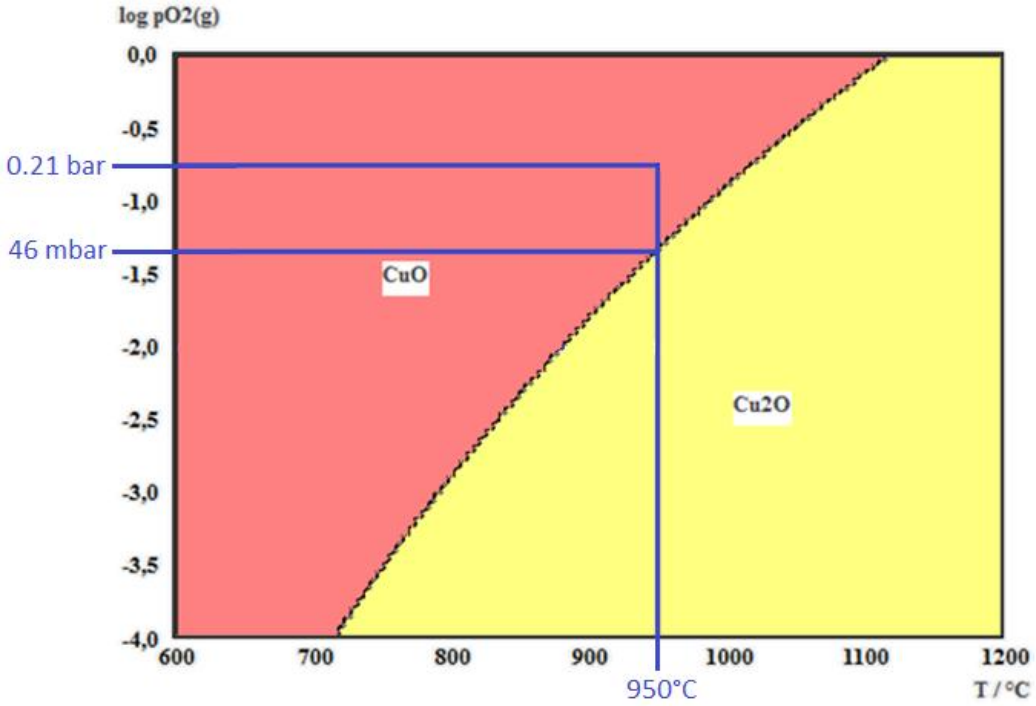



*Figure 9: Predominance diagram for the Cu-$O_2$-system, with working points marked*

HSC chemistry uses equation 2.53 to approximate the heat capacity at constant pressure $C_p$ of the components of the reaction system, which consists of four terms with the coefficients A, B, C and D, and the absolute temperature T. The coefficients and the enthalpy of formation are stored in the accompanying HSC thermochemical database for certain temperature ranges, which are limited by T1 and T2. The values for the copper oxide reaction system are shown in table 3.

$$C_p = A + B \cdot T \cdot 10^{-3} + C \cdot T^{-2} \cdot 10^{5} + D \cdot T^{2} \cdot 10^{-6} \qquad (2.53)$$

*Table 3: Thermodynamical data of the substances involved in the reaction*

| | Unit | CuO | $Cu_2O$ | $O_2$ | $O_2$ | $O_2$ |
|---|---|---|---|---|---|---|
| T1 | K | 298.15 | 298.15 | 298.15 | 700 | 1200 |
| T2 | K | 1500 | 1517 | 700 | 1200 | 2500 |
| H | kJ/mol | -155.80 | -170.60 | 0.00 | 0.00 | 0.00 |
| S | J/(mol*K) | 42.74 | 92.55 | 0.00 | 0.00 | 0.00 |
| A | J/(mol*K) | 48.59 | 64.55 | 22.06 | 29.79 | 34.86 |
| B | | 7.20 | 17.58 | 20.89 | 7.91 | 1.31 |
| C | | -7.50 | -6.39 | 1.62 | -6.19 | -14.14 |
| D | | 0.00 | 0.00 | -8.21 | -2.20 | 0.16 |

The enthalpy of substances for 25°C, also known as the standard enthalpy of formation $H^{\ominus}$ is given, enthalpies for other temperatures have to be calculated separately. The change of enthalpy with temperature is proportional to the heat capacity (equation 2.54) and can be calculated by integration of the equation from 25°C (298.15K) to the reaction temperature, which results in equation 2.55. The heat capacity is not constant and has to be evaluated with equation 2.53.

$$dH = C_p dT \tag{2.54}$$

$$H = H^{\ominus} + \int_{298.15K}^{T} C_p dT \tag{2.55}$$

# 3. Test Rig

## 3.1. Overview

At the heart of the test rig (figure 10) is the reactor, where the solid is kept and reacts at a defined temperature with a defined gas flow (from bottom to top). It is encased in a heating mantle and several isolating elements. The temperature inside the reactor can be measured with several thermocouples, and increased or decreased by regulating the heating mantle power. The off-gas is cooled and measured with an analyzer.

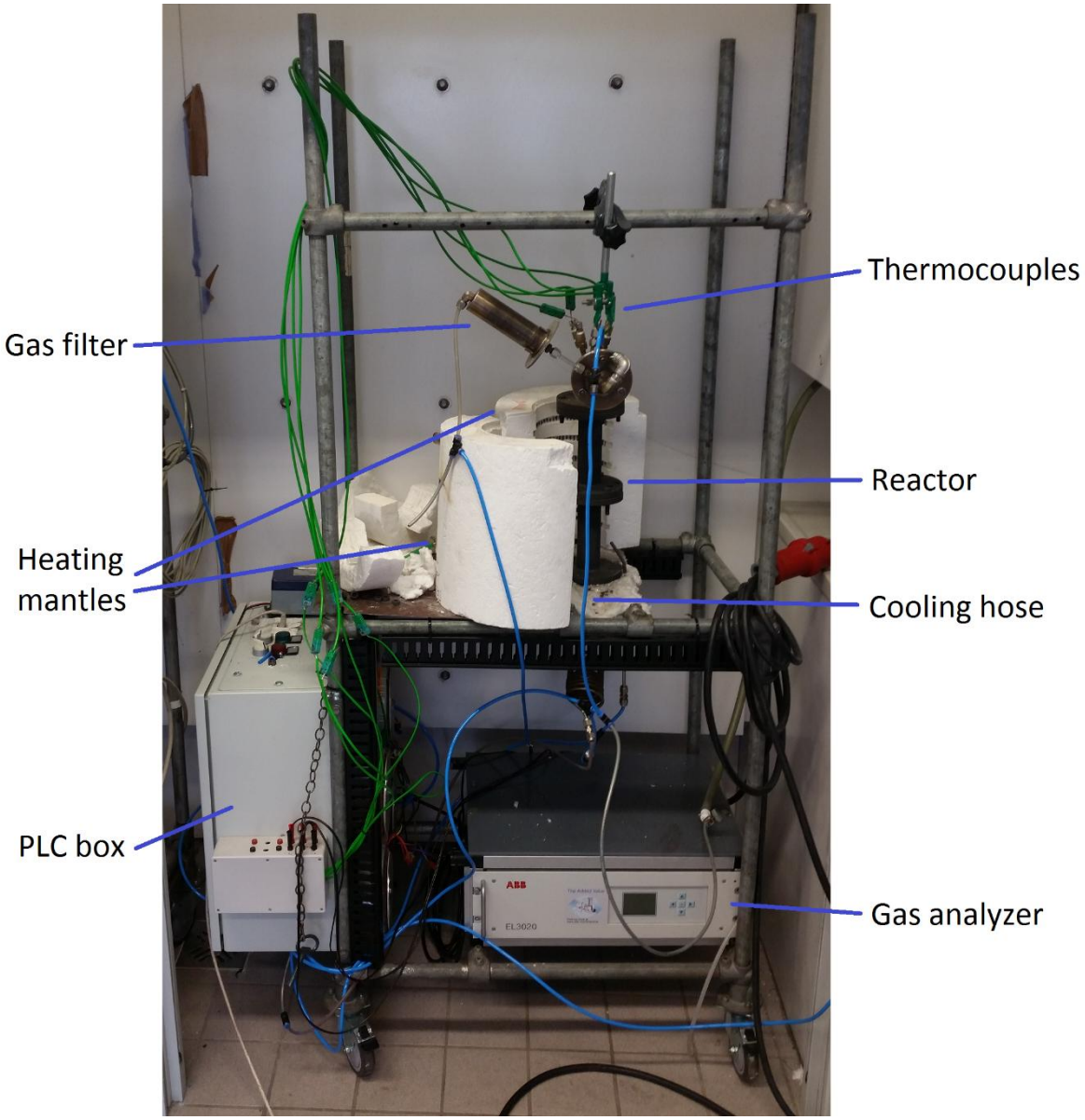



*Figure 10: Testing rig with heating mantle and isolation removed*

All measured data are collected by the programmable logic controller (PLC, German: Speicherprogrammierbare Steuerung, SPS), which uses a set temperature and the actual temperature to control the heating mantle power.

## 3.2. Reactor

### 3.2.1. Necessity

For the survey of these high temperature thermochemical heat storage materials, a new reactor had to be constructed. This reactor was conceived with two goals in mind: easy movement of the reactant into a different vessel and the placement of multiple thermocouples in order to measure the temperature at different heights of the bed, which would make reaction progress across the bed possible. Other than these functions, the reactor also has to endure temperatures of more than 1000°C and fit into the preexisting heating mantle, which is 14cm in inner diameter.

### 3.2.2. Design

The reactor is shown in figure 11 and was designed as a vertical tube with a flange at each end. To the bottom of the reactor, a socket in the form of a non-attached flange can be affixed, which provides a pocket for a sintered frit and keeps the material inside the reactor and distributes the feed-gas. The reactor and the socket can then be screwed to the pre-heating section and easily detached again at the end of the experiment, thus making easy removal and transfer of the material possible.

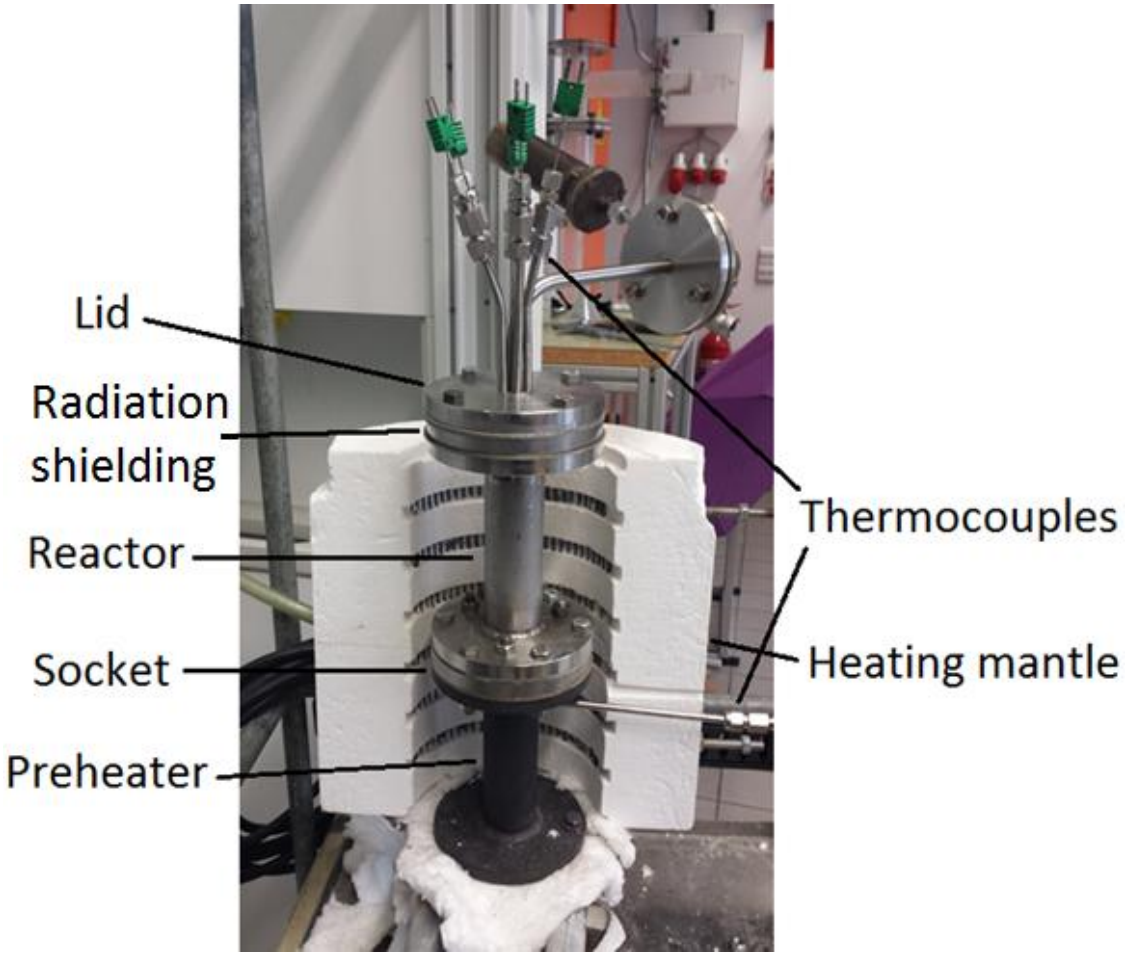



*Figure 11: Assembled apparatus (with thermocouple wires and one heating mantle removed)*

To the top of the reactor, a lid can be attached, which introduces 5 thermocouples into the reactor and provides a pipe for the off-gas. Every part of the lid except for the flange is outside of the hot zone, which is covered by a pad of mineral wool. The thermocouple and off-gas pipes are very hot nonetheless, but the thermocouple connectors, which have a design temperature of 135°C, stay intact and with the aid of air cooling, the off-gas pipe cools down to below 80°C, the design temperature of the polyurethane hoses attached to the end.

The pre-heating tube features a thermocouple, which can be used to hold the temperature of the gas constant. This is useful when the reaction is induced by changing the gas composition and the change in temperature inside the reactor is to be observed. The bottom of the tube has a connector for the gas feed, which lies outside of the hot zone. Nonetheless, it is quite hot and there is a coiled metal tube attached between the connector and the polyurethane hose.

Figure 11 displays a radiation shielding that was initially included, but has since been deemed unnecessary.

### 3.2.3. Steel

The entire reactor is built with 1.4841 steel, which is an iron based alloy with high nickel and chromium contents and some silicone, manganese and carbon. Its heat resistance up to 1150°C makes it a prime choice for our applications.

### 3.2.4. Heating Mantle

The heating mantle is a cylindrical shell consisting of two halves, with the top and the bottom open. It has an inner diameter of 14cm, a height of 30cm and a thickness of 8cm.

The mantle is made of compressed mineral fiber with electrical wires running through it and feeding the heating rods, which are exposed on the inside of the shell and radiate inward. The power supply is connected to two steel rods on each halve. The shells need 55V each and can be fed with a 55V supply line in parallel or 110V line in serial connection.

### 3.2.5. Gaskets

The gaskets have an outer diameter of 120mm, same as the flanges, and an inner diameter of 40mm, which is approximately the inner diameter of the reactor. They have 10mm holes for the M8 screws but differ in the number and positioning of the holes. All gaskets are 2mm thick. There is also a smaller gasket between the frit and the socket to provide some cushioning, but its format is arbitrary, except for the center hole through which gas can flow through.

The gaskets are 2mm thick, made of muscovite with a 0.1mm metal insertion and per manufacturer specification resistant up to 900°C. They have been successfully used at 1000°C, but become brittle with time and have to be handled very carefully and occasionally replaced.

There are 4 gaskets in the apparatus: between preheater and socket, between socket and frit, between socket and frit, and reactor, and between reactor and lid.

## 3.3. Wiring/Contact Plan

The contact plan of the test rig is shown in figure 12 (the symbols are explained in table 4). It runs on 230V AC power, which is supplied through an IEC 60309 L+N+PE 6h IP44 power plug specified up to 16A (**PS**). This power is directly distributed to 2 standard europlugs, one of which feeds the gas analyzer. When the main switch (**S1**) is activated, this power line also feeds the B+R PLC.

The heating elements (**H**) use a voltage of 55V each, which necessitates 110V if the two are wired in series. For this, the standard 230V grid supply has to be transformed down, which was done using 4 transformers (**T**).

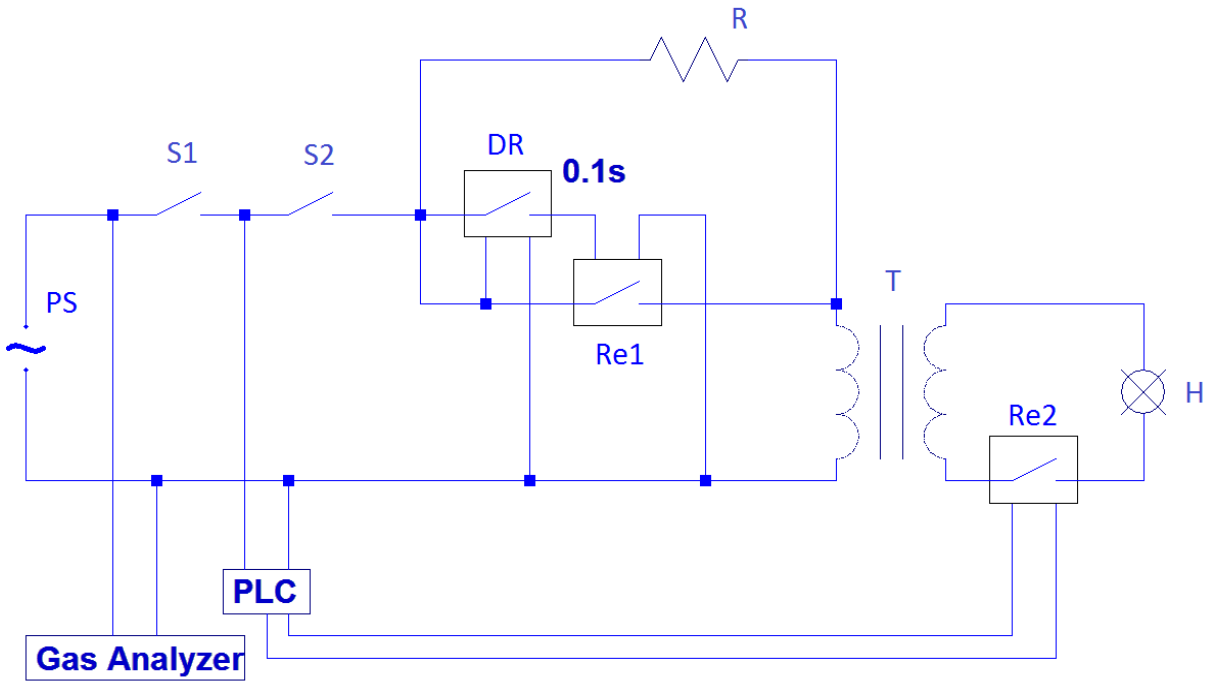
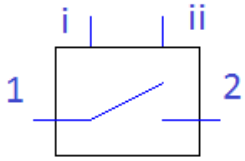
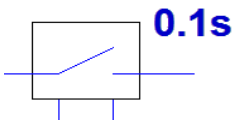
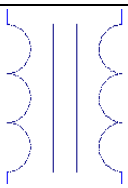



*Figure 12: Contact plan of the test rig*

*Table 4: Explanation of the symbols used in figure 12*

| | |
|---|---|
|  | Switch |
|  | Relay<br><br>1 and 2 connect when voltage is applied at i and ii |
|  | Delayed Relay |
|  | Transformers |
|  | Resistor |
|  | Power Supply (AC) |

When the heating switch (**S2**) is activated, the power line feeds the four transformers which reduce the voltage to 115V, a voltage that can be used to power the heating elements. When transformers are powered on, there may be a spike in current. To prevent this, a 6.8Ω resistor (**R**) is installed on the primary side. After 0.1s, a delayed control relay (**DR**) is closed, which activates another relay (**Re1**), because the first one cannot handle the amperage) that passes the current by the resistor, thus avoiding power losses and overheating.

The power is then passed on to the heating elements through a solid-state relay (**Re2**). Due to their high switch speed and endurance, they can be used to control the power feed by pulse width modulation (PWM). PWM is the go-to method when power has to be reduced but voltage reduction is not an option, it even has a dedicated function in the PLC programming libraries. PWM works by not letting the power run permanently, but instead letting through a pulse of varying width every period. It takes a load as input, which determines the ratio of pulse to period, as is shown in figure 13.

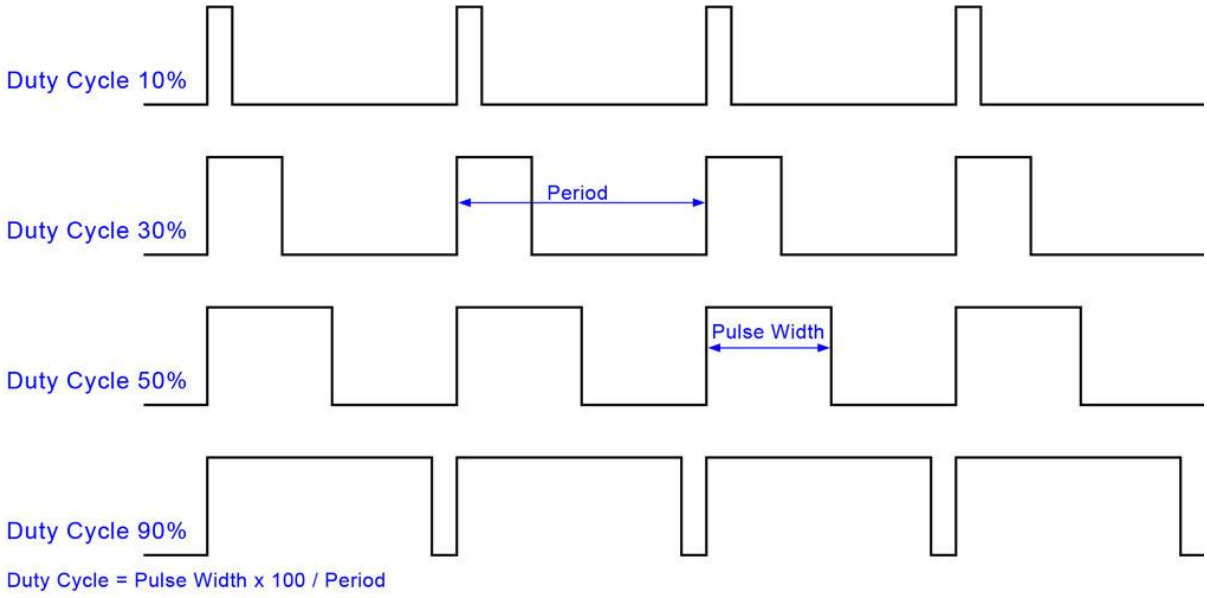



*Figure 13: Explanation of pulse width modulation (PWM) [27]*

Solid–state relays lose some power as heat, so special care has to be taken during installation to ensure that the cooling fins are vertical, to support convection cooling.

## 3.4. Programmable Logic Controller

### 3.4.1. System

The test rig uses a B&R CP1585 Programmable Logic Controller to record and process data, and to control the heating mantle and the mass flow controllers (MFCs). The system has an SD memory card that can store the data as well as a USB interface which can be used to store data on a flash drive.

The PLC executes a program which is loaded onto the memory card. A program can be loaded from the computer onto the memory card either through an Ethernet connection with the PLC or by connecting the SD card through a card reader to a computer.

The PLC is modular and can use various modules for input, output and temperature measurement. In the case of input and output, various modules are available that utilize either voltage or amperage for measurement/control and offer various resolutions. The modules used in this rig are

- 2x X20AO4622
- 3x X20AI4622
- 1x X20AT6402
- 1x X20AI8221

### 3.4.2. Languages

There are multiple languages in which a PLC can be programmed. A classic method of programming them is ladder logic, which represents simple conditions and actions in a graphical way. A more advanced and powerful way of programming is using text-based languages like C and ST. These can perform much more complex functions through the implementation of libraries.

### 3.4.3. Libraries

Libraries are sets of functions which can be added to a program. When these functions are implemented into the code, the compiler will recognize them and execute them as they are coded inside the library. Some libraries included offer advanced mathematical functions, file input/output operations and date/time processing.

### 3.4.4. Program Cycles

Every program has an initial, a cycling and an ending sequence. The initial sequence is executed during start-up to perform functions which only have to be done once, or to initiate some variables. The cycling part includes functions that periodically receive an input (from the input modules or from another function), process it and then create an output. The ending sequence can be used to store data at the shut-down of the PLC.

### 3.4.5. PLC Software

#### *3.4.5.1. General*

The PLC serves two purposes: to control the temperature and gas flow inside the reactor to a given set point or even a curve, and to monitor and save the data.

The former can either be done via the interface (set point only), or through a txt-file via USB-drive (temperature and/or gas flow curve).

Regarding the latter, monitoring can be done through the interface, which is important to find the end of the reaction and therefore proceed with the planned schedule of experiments. The saving of data on the USB-drive is important for evaluation of the data.

The PLC programming is made up of several routines with their own tasks:

- main conversion of several variables into another
- read_data connection of USB drives
- FileHandling reading and storing of data
- altControl control of the MFCs and heater

The "main" routine mainly converts the input signals into meaningful numbers, such as gas flows, which are then further used in the other routines or in the visualization. The input is connected to the PLC modules and can either be a voltage or an amperage, and is converted by the PLC into a digital signal. The modules used operate in the range of -10 to 10V, which is proportionately converted to a digital output between -32766 and 32767.

$$u = \frac{n}{n_{max}} u_{max} \tag{3.1}$$

As shown in equation 3.1, the input voltage u can then be calculated as the digital signal n divided by the positive range $n_{max}$ and multiplied with the positive range of the input voltage $u_{max}$. Similarly, amperages between 4 and 20mA are converted to numbers between 0 and 65535. The voltage or amperage is then converted to gas flows or gas concentrations. The output signals are then calculated in the reverse way (equation 3.2).

$$n = \frac{u}{u_{max}} n_{max} \tag{3.2}$$

The USB interface is essential to storing the measured data for evaluation. The detailed and commented code can be found in the annex. For this chapter, a simplified explanation will suffice: the "read_data" program instructs all interfaces to check for connected devices and if one is found, a new file device is registered using its path and name. This file device is then passed on from the "read_data" routine to the "FileHandling" routine.

The “FileHandling” routine opens the file device and can then be instructed through the visual interface to 1) read a temperature and gas flow profile or 2) record data. The data is first recorded into the internal memory on the SD card and then copied to the USB drive. This means that uncopied or lost data can always be recovered from the SD card.

The reactor temperature is set by the heating elements, which are controlled by a PID controller in “altControl”. This controller is able to set its own parameters using an auto-tuning program, and these parameters can be read and entered via the interface (see 3.3.4.2.4. Setup Page). The PID controller can be set to control the temperature of any of the thermocouples, thus making it possible to control the temperature of the feed gas in the preheater or the temperature inside the reactor.

The PLC in its current version can be connected to 4 MFCs, thus making it possible to feed the reactor with 4 different gases. As this is rarely necessary, the $4^{th}$ MFC is being used to blow air to the outside of the preheater and allow for quicker cooling. It could also be used to measure the reactor off-gas flow, which can be quite important considering the difficulties of sealing the reactor properly at these high temperatures. The true gas flow through the bed is then necessary to set up the mass balance and evaluate the yield of the reaction.

The software also has a safety feature which turns off the heating if the temperature reaches 1050°C at any of the thermocouples.

### *3.4.5.2. Graphical User Interface (GUI)*

#### 3.4.5.2.1. Home Page

Figure 14 pictures the home page, which controls the data storage, temperature and the gas flows. This is the page that will be shown on start-up of the PLC, from which the user can navigate to the other pages by pressing the buttons in **j**. Pressing button **a** makes the PLC search for USB drives and if one has been attached light **A** will light up green, showing that the PLC has successfully connected to it. Buttons **b** and **c** initiate and end the measurement, which stores data such as time, temperature power and gas flow once per second inside the PLC memory. During measurement, light **B** is green. Button **d** copies the file to the USB drive under a name which can be entered in window **D**. By default, the name is the current date and multiple measurements on the same day are denoted by indices (e.g. 2015-10-16-**1**). Afterwards, light **C** is lit. As the data is stored internally at first, a USB drive does not have to be attached during measurement.

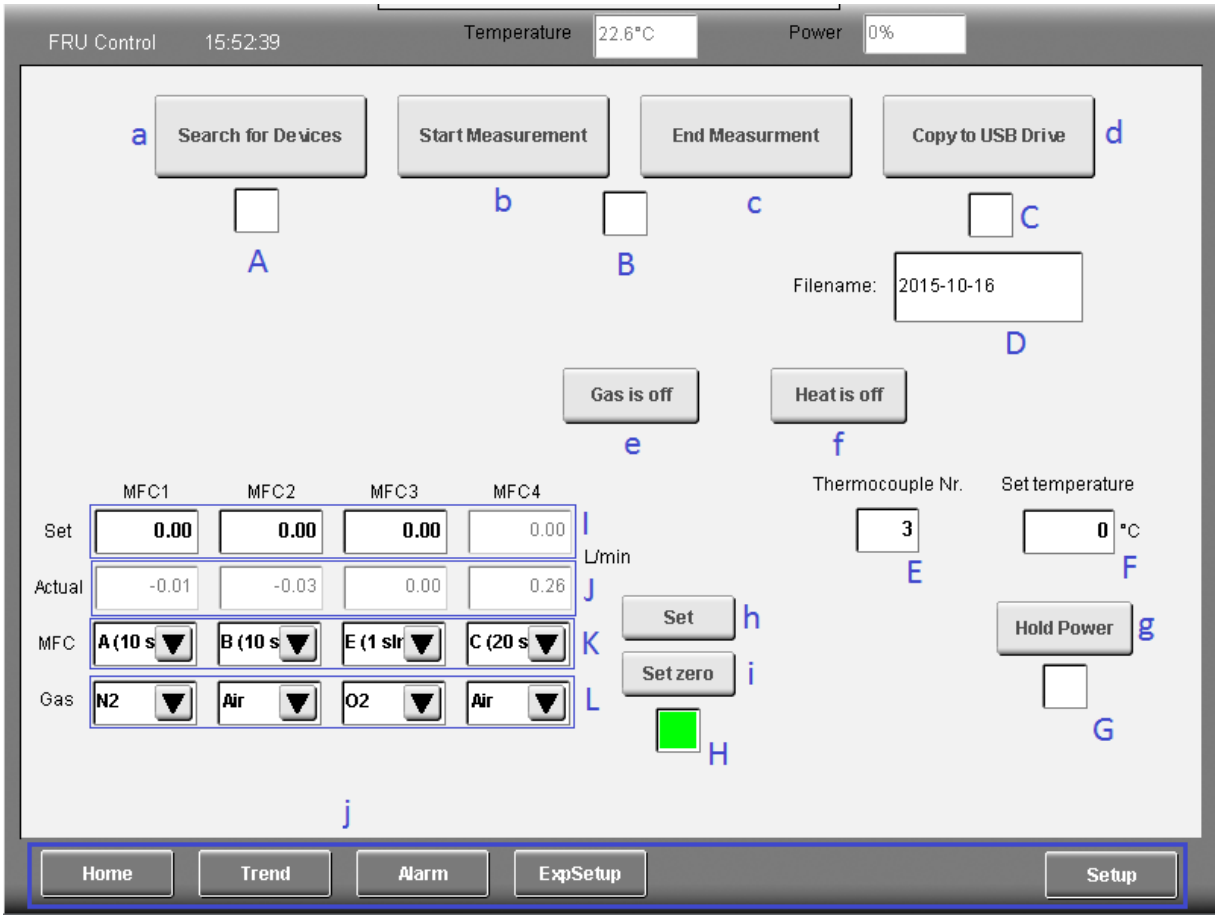



*Figure 14: PLC visualization home page*

Button **e** switches the gas flow on and off, which can be set by entering a value in L/min in **I** for the respective MFC (except for Nr.4, which is used for cooling and is PID-controlled) and has to be confirmed by pressing button **h**. This enables the operator to change two or more gas flows simultaneously, so the total gas flow can be held constant. The actual gas flow can be read in row **J**. As various MFCs with different gas flow ranges and different gas flow/voltage response curves are

being used, the specific models can be chosen in the dropdown list in **K**, corresponding to the cable they're attached to. The maximal gas flow is also shown in the list, although according to the user manual 120% of the nominal maximum can be used. Lastly, the type of gas has to be chosen in **L**.

Button **i** can be pressed (when gas flow is set to zero) to set the measured gas flow to zero and subtract the difference from further measurements. If all gas flows are within 0.1L/min of their set value, light **H** shines green.

To use the PID-controlled heating, button **f** has to be switched on and a thermocouple has to be selected (1 to 6) in window **E**. The operator should check if the PID parameters in the setup page correspond to the chosen thermocouple (numbers 2 to 6 are similar, but number 1 is different from the others, see "Settings Page"). Then, a set temperature can be entered into **F**. To hold a constant power input, button **g** can be pressed. The PLC then averages the power over the next 60 seconds, and holds this power afterwards until the button is pressed again. At the end of the 60 seconds, **G** lights up. This function is useful to keep the power input into the reactor constant so the temperature change can be measured properly. Another approach with measurement of the temperature in the preheating zone was tried, but this proved useless as the temperature – and thus the power – fluctuated significantly. The function can also be used when cooling down the reactor: The temperature is set to 0°C, at which point the power goes to -100%, and thus maximal cooling. At this point, the power should be locked, as the D component of the PID controller will shortly thereafter increase the power again.

#### 3.4.5.2.2. Trend Page

On the trend page, seen in figure 15, there are two big trend graphs. The upper graph depicts the set (dark red) and the actual temperature (black) of the thermocouple chosen on the main page. The temperature can be read on the left axis, the green line additionally depicts the power in percent, which can be read on the right axis. The temperatures of all six thermocouples can also be read in boxes e.

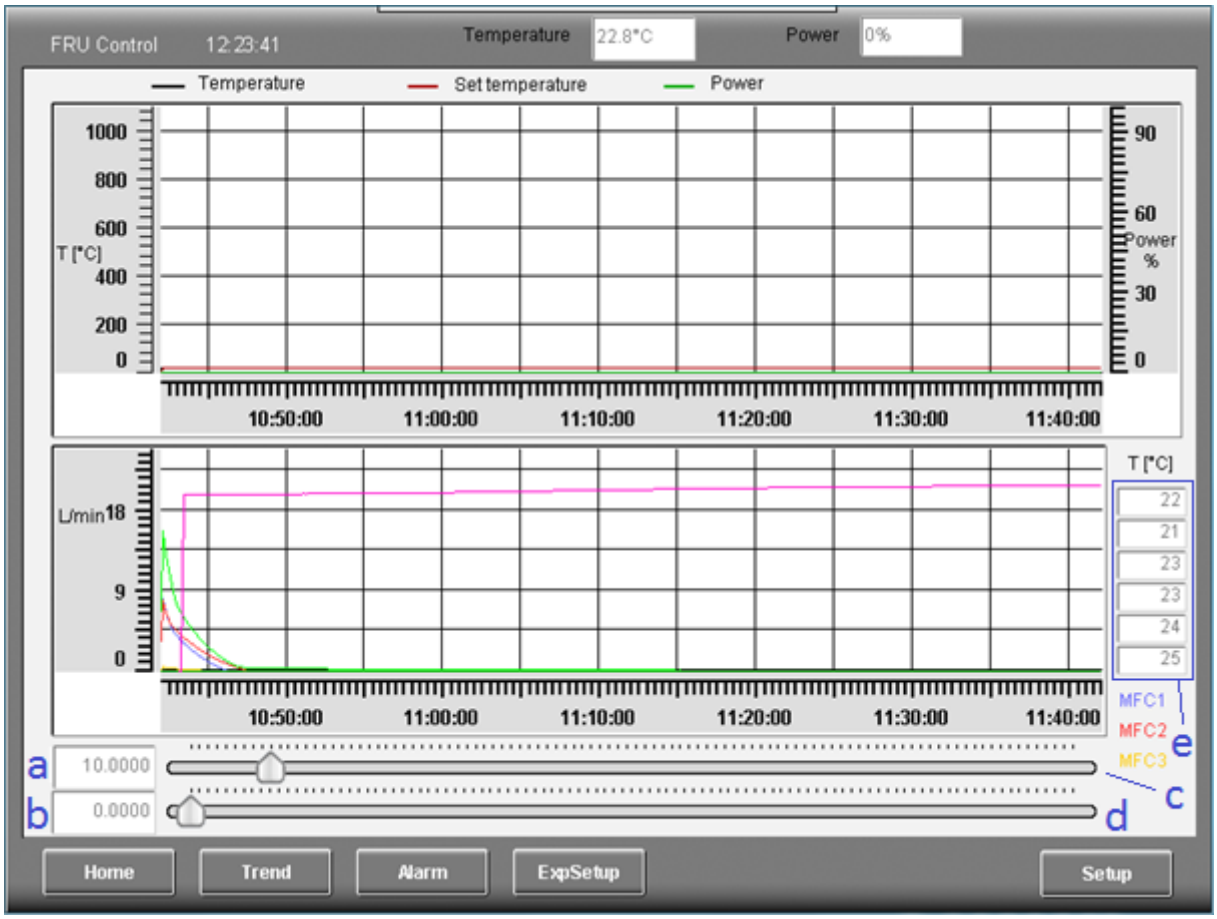



*Figure 15: PLC visualization trend page*

The bottom graph depicts the set and actual values (dark and light shades respectively) of the MFCs (blue, red, yellow and green lines for MFC 1, 2, 3 and 4 respectively). The pink line depicts the oxygen concentration. All values can be read on the left axis in L/min and % respectively.

At the bottom of the page, there are two sliders, c and d. Slider c sets the range of both graphs to 10 hours divided by the set number, which is shown in box a. The bottom slider moves the time axis of the graphs by 6 minutes per point, the points being shown in box b.

#### 3.4.5.2.3. Experiment Setup Page

To the top left of the experiment setup page (figure 16), button a can be pressed, which reads the files and folders in the main directory of the attached USB drive. This function, of course, only works after the USB drive has been connected in the main page. Due to inherent restrictions in the program's library, only the first ten files/folders can be read. The contents of the directory are displayed in the drop-down box **b** to the right. If a text file with a Temperature/MFC program was chosen and initialized by pressing button **c**, the curves will be displayed in the graph at the bottom. If a folder was chosen and opened with button **a**, the contents of this folder will be read and displayed in the drop-down box **b**. This can be repeated until the right file was found in the right path. To return from a directory to the above directory, one simply chooses the '.' or '..' in the drop-down list.

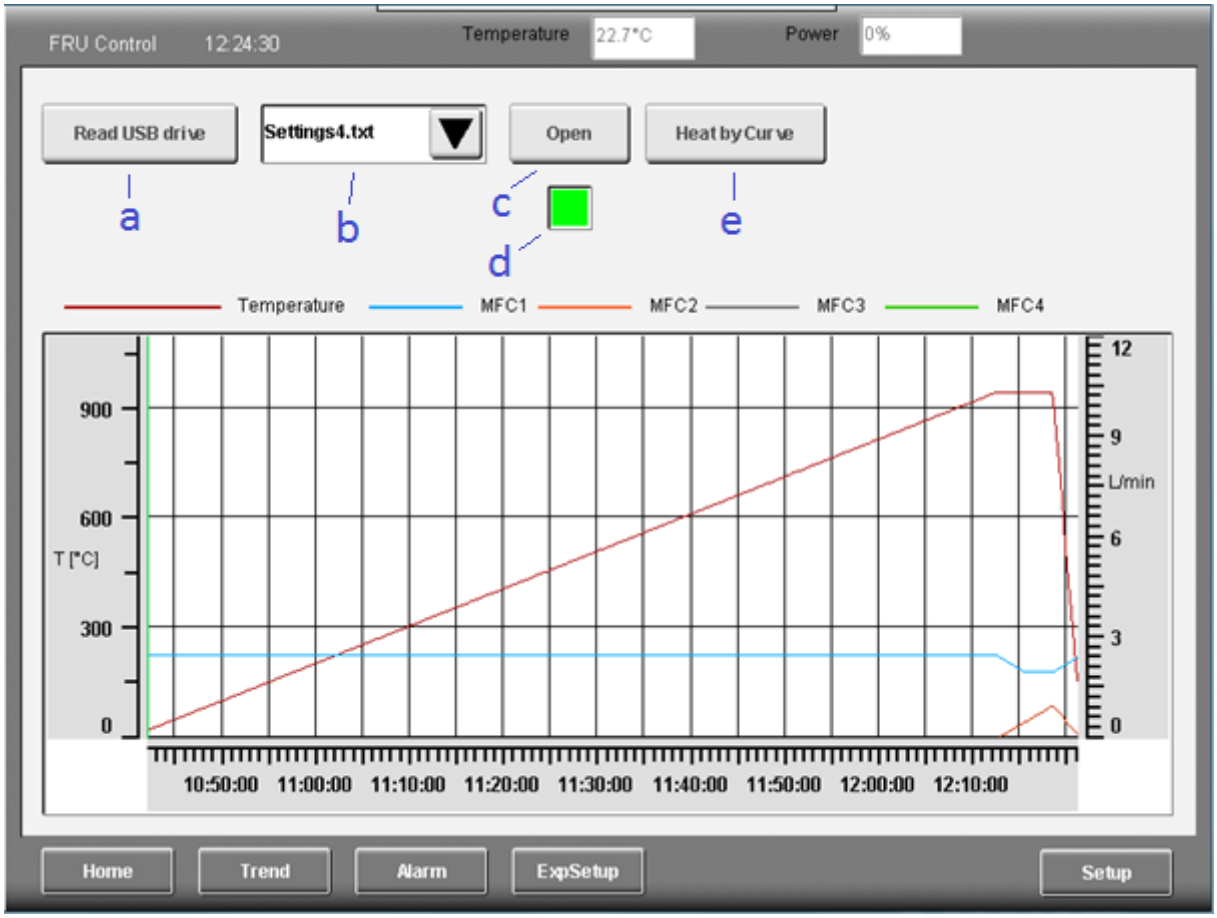



*Figure 16: PLC visualization experiment setup page*

The operator can then check the curve, which also allows for an estimate of the end time, and can then start the program with button **e**.

#### 3.4.5.2.4. Setup Page

To the top left region **a** in the setup page (figure 17), there are the two buttons "**English**" and "**German**" with which one can change the language of the interface. Below, the IP address and the node number can be read, although the former has to be known to access the visualization in the first place, and the latter is only necessary to connect the B+R Automation Studio with the PLC. In box **k**, the time and date should be set. The "**System Diagnostics Manager**" can show useful debug information. In box **b**, the CPU temperature can be read, which should stay below 70°C. The CPU is cooled with a pressurized air hose inside the PLC box. It is taped to the ceiling of the box to ensure an air flow through the box to the bottom hole, and it can be regulated with the stop-cock at the end.

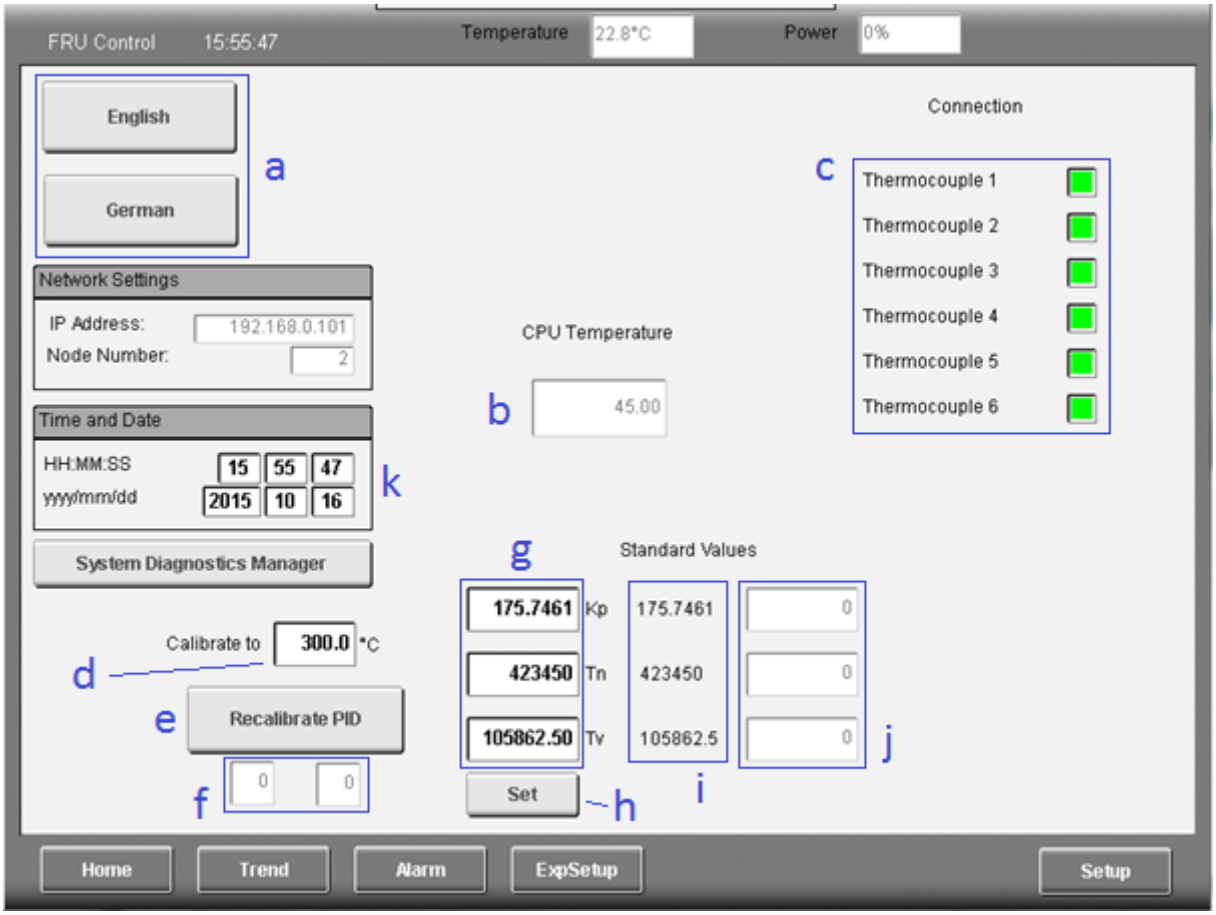



*Figure 17: PLC visualization setup page*

In region **c**, one can check which thermocouples are connected, indicated by a green light next to each one. The bottom portion of this page is dedicated to the calibration of the PID controller. The controller is calibrated at a certain temperature and while measuring with a certain thermocouple. The temperature can be entered in box **d**, the thermocouple can be chosen on the home page. The process is then started by pressing button **e**. The controller then heats up to the specified temperature and does two cycles with four repetitions each, which can be followed with the boxes in **f** (counting starts at zero). At the end, the boxes show two zeroes again, and the tuned parameters can be read from column **j**. These values can then be entered in column **g** and confirmed with button **h** to use these

parameters for the PID controller afterwards. The values should also be written down, as they will disappear after the next restart of the system or after the next calibration. In the column **i**, the usual values for thermocouples 2-6 are displayed.

#### 3.4.5.2.5. Global Layer

The global layer is visible on all pages and constitutes the top and bottom frame of the window. In the top left, the words “FRU control” and the time are visible. In the center, the current temperature of the selected thermocouple and the heater power are shown. The bottom consists of the buttons that can be used to change the page.

### 3.4.6. Temperature/Gas Flow Program Syntax

The temperature and the gas-flow of the apparatus can be programmed to run a curve of the operator's liking. This is done by writing a text file (e.g. with MS Notepad) and loading it from a USB drive, the syntax of the file will be explained here.

Any rows starting with a slash ("/") are disregarded, this is so the operator can comment the program with a date of creation, an intended use or the name of the creator.

There are 6 columns to be written, separated by tabulations. Each row corresponds to a corner point of the curve. The first column describes the process time at which that corner is reached (in seconds from start of the program). The second column signifies the temperature at that point (in °C), the last four columns set the MFC gas flow in liters per minute.

Instead of writing a time, one can also write an increment (e.g. +500, signifying 500 seconds after the last time) or a rate, which makes the PLC automatically calculate the necessary time to reach a temperature at the given rate (an upper or lower case 'r' with the rate in K/min, e.g. "r20").

If a sequence of temperature and gas flow ramps should be repeated, one can also repeat any number of previous rows multiple times, simply writing "REPnxm", n signifying the number of previous rows and m signifying the number of repetitions, not including the execution of the original written rows (REP4x5 thus repeats the previous 4 rows 5 times, for a total of 6 times). The remainder of the row can then be left blank. The PLC automatically calculates the necessary time shifts for each repetition, but the extended duration has to be taken into account for any subsequent written lines.

The program only allows ramps, but steps can also be made by writing 1 second ramps.

The curve of the program is then shown on screen after initiation and can be checked for correctness. Despite the extensive automation of the controls, the MFCs and gases still have to be manually set on the main page. An example is presented in annex C.

## 3.5. Operation

### 3.5.1. Assembly

The apparatus is mounted on two steel rails, with a mineral fiber mat for thermal isolation in between. This is done using two screws, one of which is grounded. The material that is to be studied is filled into the reactor, then the lid with the attached gas filter is screwed on top. The heating mantle is closed around the reactor and covered with two smaller isolating mantles, which are topped off with a mineral fiber mat. In total, 8 M8x50 and 7 M8x40 screws are needed.

Due to the fact that the off-gas has to be cooled from 950°C to below 80°C (maximum working temperature of the polyurethane hoses) within a relatively short section, a constant stream of pressurized air is blown at the steel pipes. This stream can be adjusted using a stop-cock and the supply valve and has to be increased when the gas flow is increased during cool-down.

The bottom pad of the isolation is checked for any gaps, which are then plugged with more mineral fiber pads. This has to be done to avoid chimney effect, which would lead to significant heat losses.

### 3.5.2. Start-Up

The power plug is plugged into a 3-phase power outlet secured to at least 16A and the PLC is started using the first (white) power switch. The off-gas analyzer is directly powered and activates independently of the switch. It has to warm up for 30min and can then be calibrated using the built-in cuvettes and the surrounding air.

The MFCs also have to warm up, and can be used when they show a gas flow of around zero. If they show a gas flow which is not zero at rest, despite the set value being zero, the MFCs can be physically zeroed by turning the “zero”-screw on the MFC, or electronically by pressing “Set Zero” in VNC (button i in figure 14). The latter option is simpler but has a downside: due to a resulting non-zero voltage and fluctuations of the MFCs, there can be a small gas flow present despite none being desired.

Also, the specific MFCs and gases have to be assigned to the MFC channels, as the MFCs have different flow-through ranges and the gases have different thermal capacities, which has to be corrected for due to the measuring method. MFCs from 1 to 20 standard liters per minute (slm) are available.

In case a temperature/gas flow program is to be executed, a USB drive can now be attached and read, and a program chosen and initiated from the same. It can then be started and the PLC runs without any user action until the end of the program.

Otherwise, gas flows and the set temperature can be set directly through the VNC interface.

As a last safety precaution, the red switch has to be flipped to activate the heating power. The measurement should now be started.

### 3.5.3. During the Experiment

As the test rig is fully automatized, there is little to be done during the experiment. The three tasks that should be done regularly are

- checking the gas supply
- checking the exit temperature
- checking the CPU temperature

If the MFCs cannot supply enough gas, light H on the home page of the GUI will stop shining green. By comparing rows I and J, the operator can identify the gas that runs low and further open the valve or change the gas cylinder if necessary.

The exit temperature has to be checked to ensure that the off-gas hose does not melt. The maximum temperature at the hose attachment should not exceed 60°C, which is about the temperature at which it is too hot to hold. If it is in fact too hot to hold with bare hands, the cooling gas flow pointed at the off-gas exit should be increased.

The CPU temperature can be checked on the setup page in window b and should not exceed 70°C. If it does, the cooling gas flow of the hose fixed to the top of the inside of the PLC box should be increased.

### 3.5.4. Power-Down

After completion of the experiment, the measurement can be stopped and the file with the measured data can be copied to the USB drive. The heating power supply should be switched off with the red switch. The top of the isolation can be removed right away with caution and placed on a non-temperature-sensitive surface.

The cooling pipe which ends within the isolation is connected to the pressurized air supply via a cock which is usually closed but can now be opened. This blows air directly onto the preheater at the base of the apparatus and through the heating mantle to the top, cooling all of the apparatus. To increase the heat transfer inside the reactor, the gas flow can be increased to 10l/min or more. But to ensure sufficient cooling of the off-gas, the air flow at the upper cooling hose has to be increased. Under no circumstance should the pipe just before the polyurethane hose be too hot to hold with bare hands for a prolonged time. After around 30min, the temperature should be around 100°C, the heating mantle can be carefully opened and – if urgent - the apparatus can be disassembled using heat-resistant gloves.

# 4. Experiments

## 4.1. Empty Apparatus Response Curve

To accurately calculate the bound/emitted oxygen, a reference curve had to be measured. For this, an experiment without a sample was made at 800°C and 950°C, consisting of 4 cycles in total with an oxygen step up from 0% to 23-24% (as seen in figure 18; slight fluctuations occur due to the inaccuracy of the MFCs) followed by a step down to 0%. The total gas flow was 2.5L/min during the step up and during the step down in cycles 2 and 4, while it was 2L/min during the step down cycles 1 and 3.

The area between the oxygen input and output curves was integrated according to equations 4.1 and 4.2. The method for integration applied here is the rectangle rule, which approximates the area under the curve with a number of rectangles. In this case we're applying 120 rectangles which are 1 second in width and use the oxygen concentration at their left boundary for height. This makes for an evaluation over 120 seconds , at which point the oxygen concentrations have returned to within 0.01% of their final value.

This volume we are calculating is the volume between oxygen feed and analyzer, i.e. the reactor, preheater and multiple hoses, and is called "dead volume". It was calculated both for the step up (eq. 4.1) and step down (eq. 4.2) and makes up about 0.2L, independent of the temperature, as shown in tables 5 and 6 (for step up and step down, respectively).

$$V_{dead,up} = \sum_{t=0s}^{119s} (c_{O_2,max} - c_{O_2,t}) \cdot \dot{V} \cdot \Delta t \tag{4.1}$$

$$V_{dead,down} = \sum_{t=0s}^{119s} (c_{O_2,t} - c_{O_2,min}) \cdot \dot{V} \cdot \Delta t \tag{4.2}$$

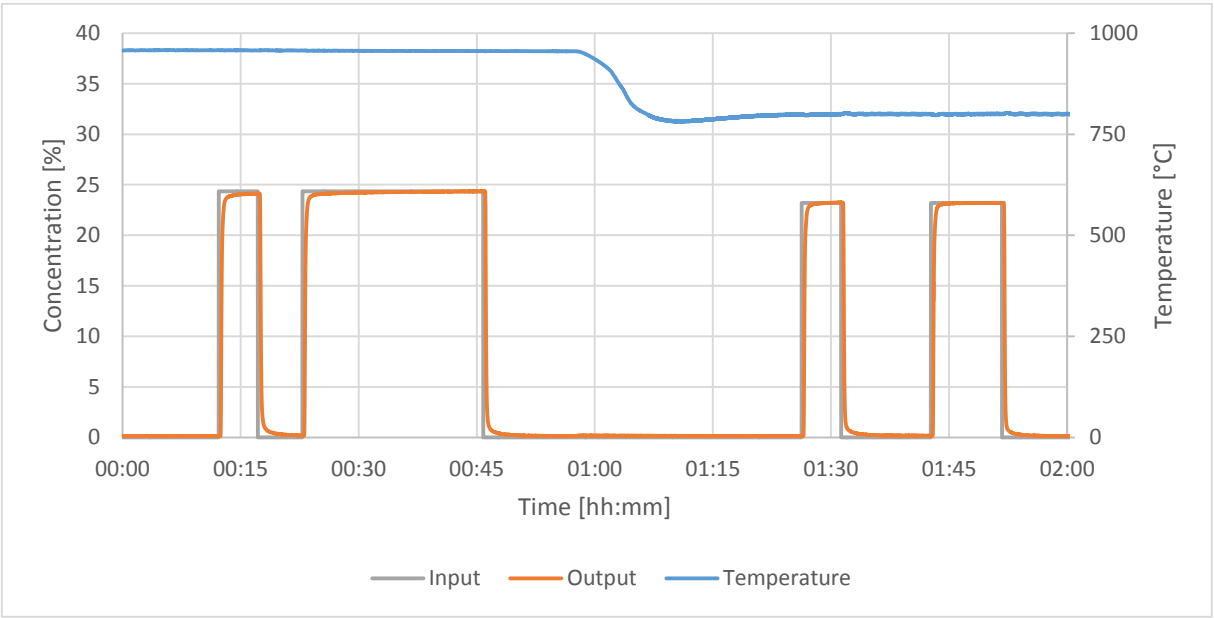



*Figure 18: Empty apparatus response curve*

*Table 5: Oxygen step up dead volumes*

| Cycle | | 1 | 2 | 3 | 4 | Average | Standard deviation |
|---|---|---|---|---|---|---|---|
| Temperature | °C | 950°C | | 800°C | | | |
| Gas flow | L/min | 2.5 | | | | | |
| Dead Volume | L | 0.231 | 0.229 | 0.222 | 0.219 | 0.225 | 0.006 |

*Table 6: Oxygen step down dead volumes*

| Cycle | | 1 | 2 | 3 | 4 | Average | Standard deviation |
|---|---|---|---|---|---|---|---|
| Temperature | °C | 950°C | | 800°C | | | |
| Gas flow | L/min | 2 | 2.5 | 2 | 2.5 | | |
| Dead volume | L | 0.213 | 0.251 | 0.193 | 0.231 | 0.222 | 0.025 |

Figure 19 shows the step up curves superimposed on each other, with the switch of the gas input composition at 00:00. The curves of cycle 1 and 3 are barely recognizable as they are overlapped by cycles 2 and 4, respectively.

With all 4 curves, it takes 14s before the analyser picks up on any change. At the 1 minute mark, the oxygen concentration is stable and stays roughly constant. The final oxygen concentration is 24.0% for cycles 1 and 2, and 23.1% for cycles 3 and 4.

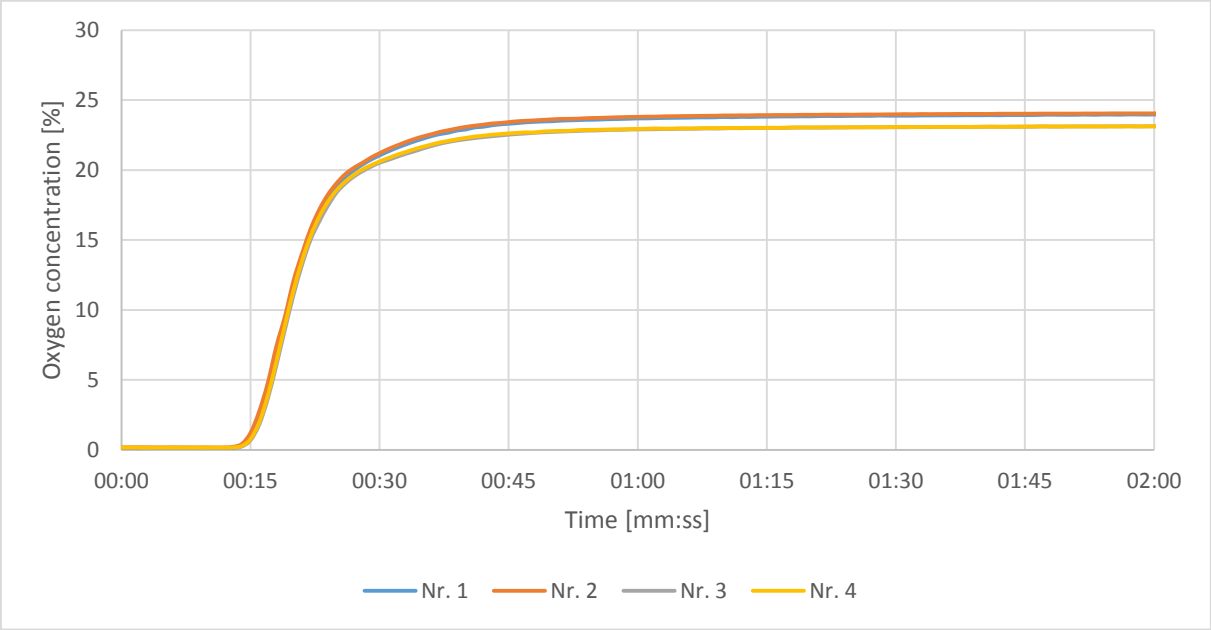



*Figure 19: Empty apparatus oxygen increase response*

Figure 20 displays the step down response curves of the 4 cycles. At step down there is one additional variable, the gas flow, which is 2L/min for cycles 1 and 3, and 2.5L/min for cycles 2 and 4.

The response starts at 14s and is near constant after the 1 minute mark, as with the step up responses. Due to the lower gas flow, cycles 1 and 3 would be expected to show a slower response, which is only present in cycle 1.

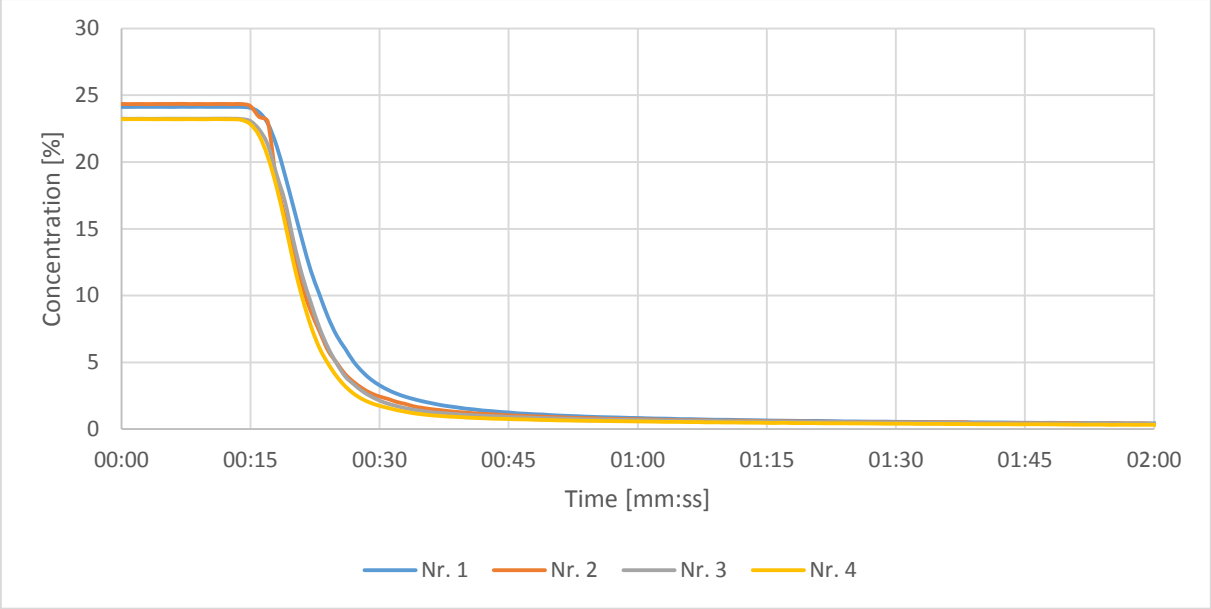



*Figure 20: Empty apparatus oxygen decrease response*

## 4.2. Experiments with $CuO/Cu_2O$

### 4.2.1. Inducing a Reaction

There are two ways to start a reaction. One is by changing the gas composition at a constant temperature, which produces a defined starting point of the reaction. As pictured in figure 21, there is an initial phase with a constant off-gas oxygen concentration, which is the equilibrium concentration at the given temperature.

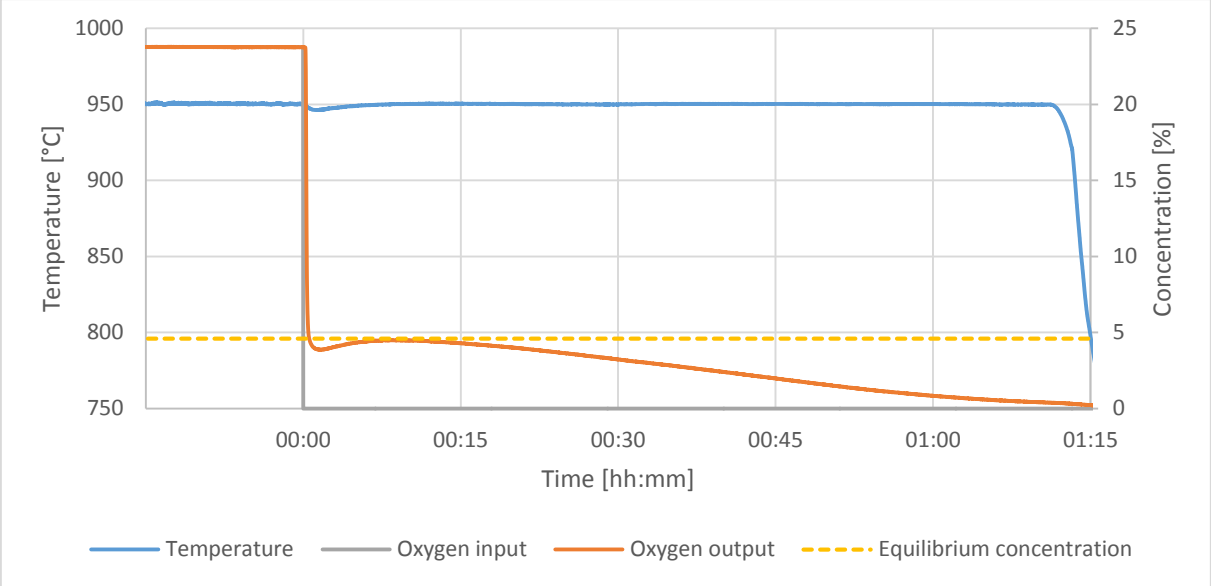



*Figure 21: Reaction start by gas change*

Figure 22 displays the predominance diagram for the copper oxide system, with the 950°C highlighted with its equilibrium oxygen partial pressure of 46mbar, which translates to the ~5% oxygen in figure 21. In the second phase, when the reaction is so slow that equilibrium is not reached anymore, there is an approximately exponential decline of the reaction rate.

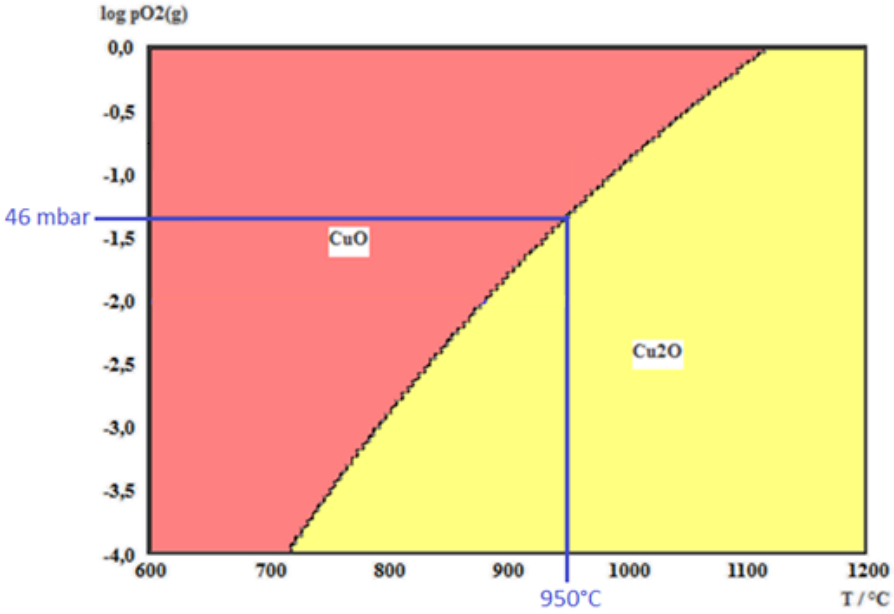



*Figure 22: Predominance diagram for copper oxide with the conditions in figure 20 marked*

The other method is to change the temperature (usually from room temperature to >800°C), where at some temperature the reaction rate slowly starts to increase, peaks and then decays exponentially, resulting in the trend in figure 23.

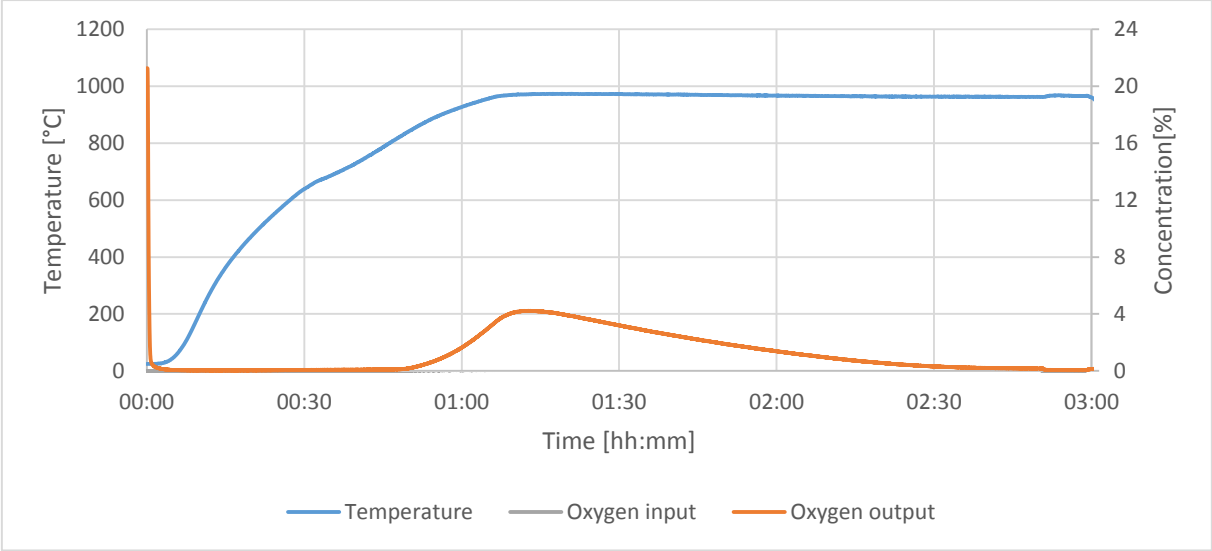



*Figure 23: Reaction start by temperature change*

### 4.2.2. Granulate

To perform multiple experiments quickly, the experiments were designed to run at the same temperature, but with different oxygen partial pressures. The higher pressure is 0.21bar (decadic logarithm: -0.67), the atmospheric oxygen partial pressure. To keep a sufficient distance from the equilibrium line, 950°C were chosen for this. The equilibrium at this temperature lies at 0.046bar oxygen (decadic logarithm: -1.33), which in theory allows for removing oxygen from the bed at 4.6% of the total gas flow, assuming 0% oxygen in the gas input and atmospheric pressure.

The experiments were performed with 44g of $Cu_2O$, which is equivalent to 49g CuO. This makes for a bed height of approximately 1.5cm. Its particle size distribution is displayed in figure 24.

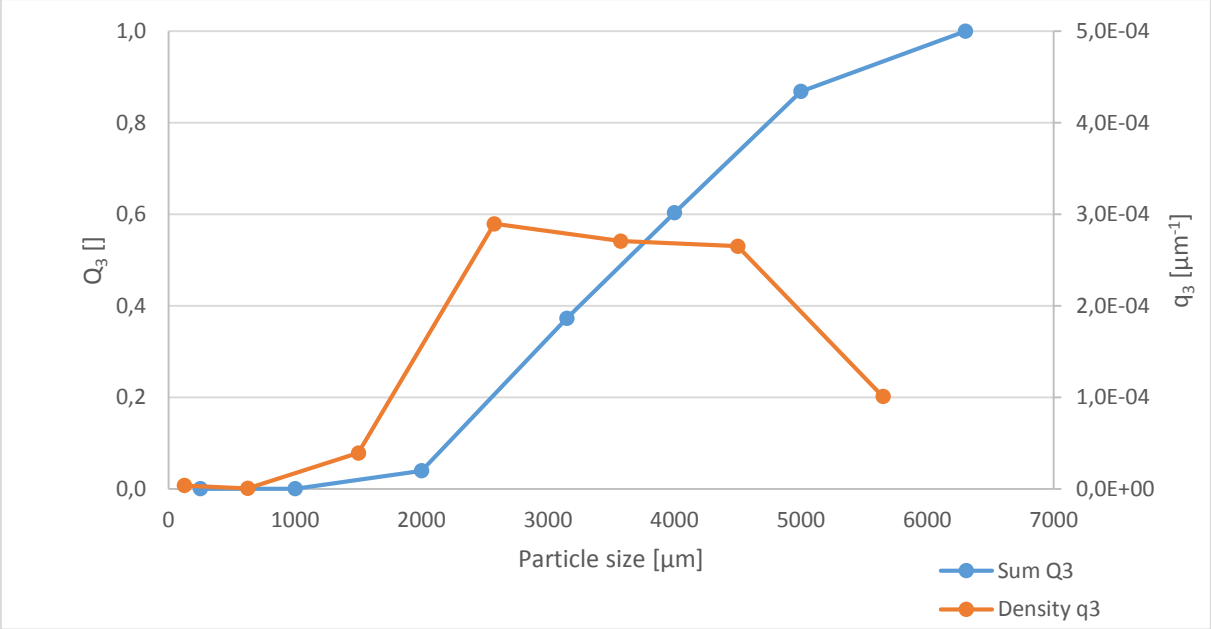



*Figure 24: Particle size distribution of the copper oxide granulate*

Multiple experiments were performed, where temperature, gas flow, oxygen concentration and reaction duration were varied. Sample trends that are characteristic of the input, the empty reactor output and the filled reactor output are pictured in figure 25.

In experiments A and C, multiple reactions were performed in succession, without a weighing in between. The yield of all but the final reaction can thus not be evaluated. The other experiments consisted of only one step and can be fully evaluated. Experiments B, D and G ended with oxidized/discharged product, their reaction parameters are shown in table 7. Experiments A, C and F ended with reduced/charged product, their parameters are shown in table 8.

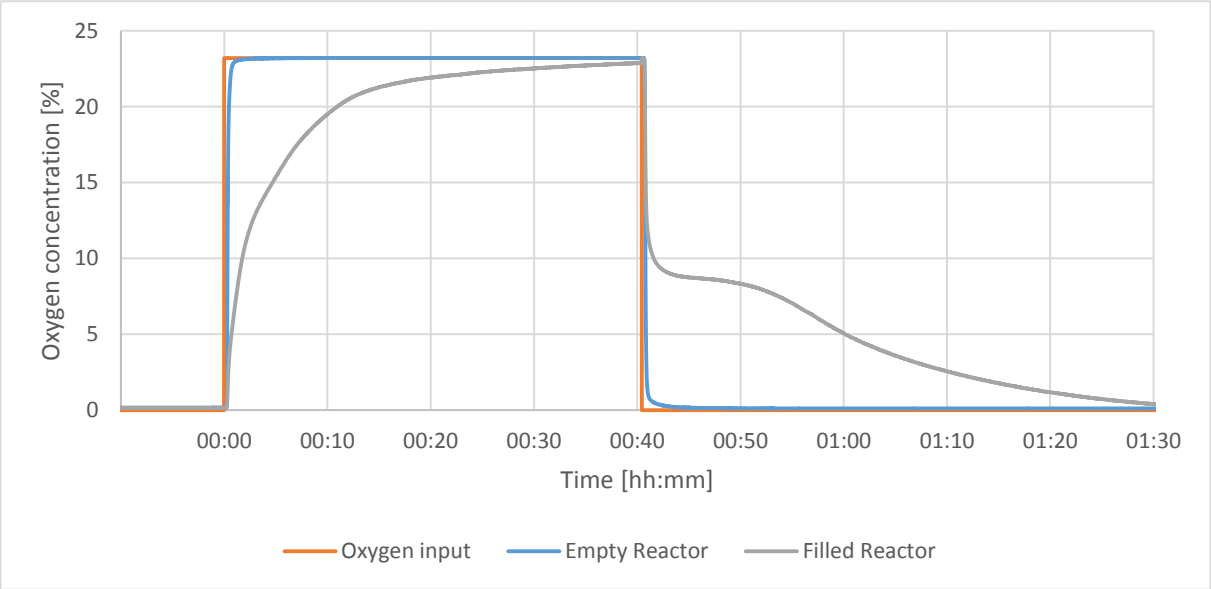



*Figure 25: Oxygen curve comparison*

*Table 7: Parameters of the discharge reactions*

| Experiment number | B | D | G |
|---|---|---|---|
| Temperature [°C] | 950 | 850 | 950 |
| Gas flow [L/min] | 4.5 | 2.5 | 2.5 |
| Gas velocity [cm/s] | 7.0 | 3.9 | 3.9 |
| Duration [h:mm] | 1:15 | 0:20 | 1:00 |
| Oxygen concentration [%] | 11 | 20 | 24 |

*Table 8: Parameters of the charge reactions*

| Experiment number | A | C | F |
|---|---|---|---|
| Temperature [°C] | 950 | 1000 | 950 |
| Gas flow [L/min] | 2 | 2 | 2.5 |
| Gas velocity [cm/s] | 3.1 | 3.1 | 3.9 |
| Duration [h:mm] | 1:30 | 0:40 | 1:20 |

### 4.2.3. Powder

There was one charging experiment performed with a copper oxide powder, which has the particle size distribution shown in figure 26, displaying a peak at about 75µm.

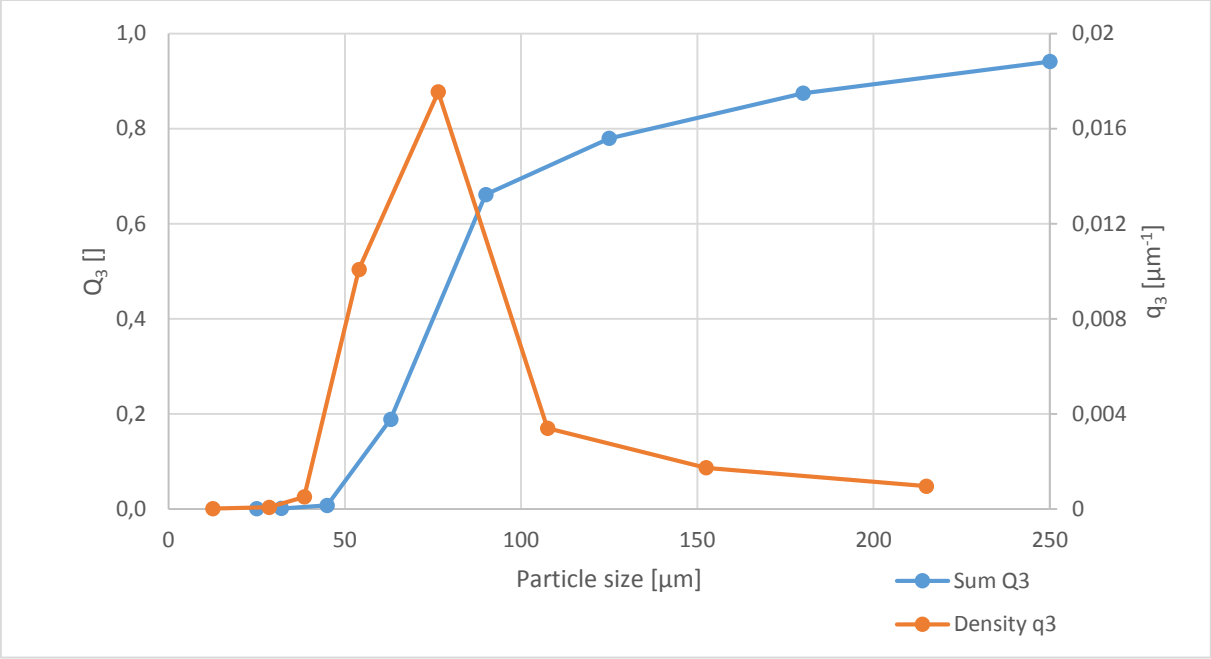



*Figure 26: Particle size distribution of the copper oxide powder*

The powder was used in experiment F, the parameters for this experiment were 2h15min of exposure to 2L/min $N_2$ and mostly 950°C, with 20min exposure to slightly higher temperatures of up to 1000°C (figure 27).

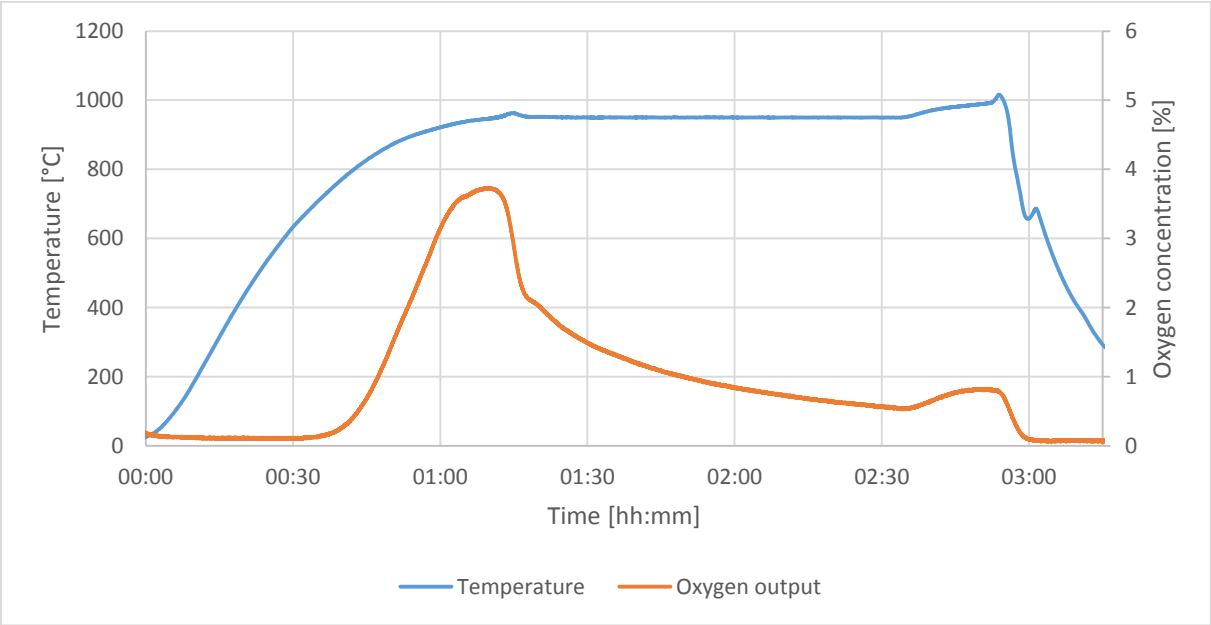



*Figure 27: Copper oxide powder experiment*

The performance of the powder was too poor to pursue further experiments.

# 5. Results & Conclusion

## 5.1. Results

The following graphs show the measured data of the 7 experiments performed with copper oxide inside the reactor. The temperature is measured at the T2 thermocouple, which is the one lowered the furthest into the reactor, with its tip approximately 3.5cm from the frit. The temperature may show some fluctuations during oxygen input concentration increases and decreases, as the addition or removal of oxygen gas flow changes the temperature transfer inside the reactor. The fact that the heating control loop is connected to a different thermocouple augments this effect.

The timing of the jumps in oxygen input concentration was taken from the PLC's log of control signals sent to the MFCs. The values however, have been taken from the values of the oxygen output concentrations once they became stationary, as the MFCs are too inaccurate to consider their measurements. The gas flow values given in the experiments' descriptions are thus quite inaccurate. The oxygen output concentration has been taken directly from the gas analyzer and is assumed to be accurate.

Reducing/charging reactions can be recognized by the output oxygen concentration (orange curve) being higher than the input oxygen concentration (grey curve). The reverse is true for oxidizing/discharging reactions.

The percentages of the oxidized or reduced component at the end of the experiments have been calculated by weight (see chapter 5.2.1. Yield) and are also mentioned with each experiment. In the further evaluation (chapter 5.2.), it is assumed to be only dependent on the final step of the experiment. If the reduced state is the final state, the theoretical share of the reduced substance may exceed 100% due to the way it is calculated; instead, 100% conversion are given, as was confirmed for some, but not all products by XRD.

All volumes are given in standard liters and all gas flows are given in standard liters per minute.

### 5.1.1. Experiment A

Experiment A (figure 28) starts with oxidized/discharged material, which is heated to 950°C at 0% oxygen and is reduced/charged from hours 0.75 to 3.25. At hour 3.25, the oxygen input concentration is increased to 24%, inducing oxidation. After 1.5 hours, oxygen input concentration is set back to 0%, initiating reduction. The reduced material is then cooled down and shows a yield of around 100%. The gas flow was 2L/min $N_2$ throughout and 0.5L/min $O_2$ during oxidation.

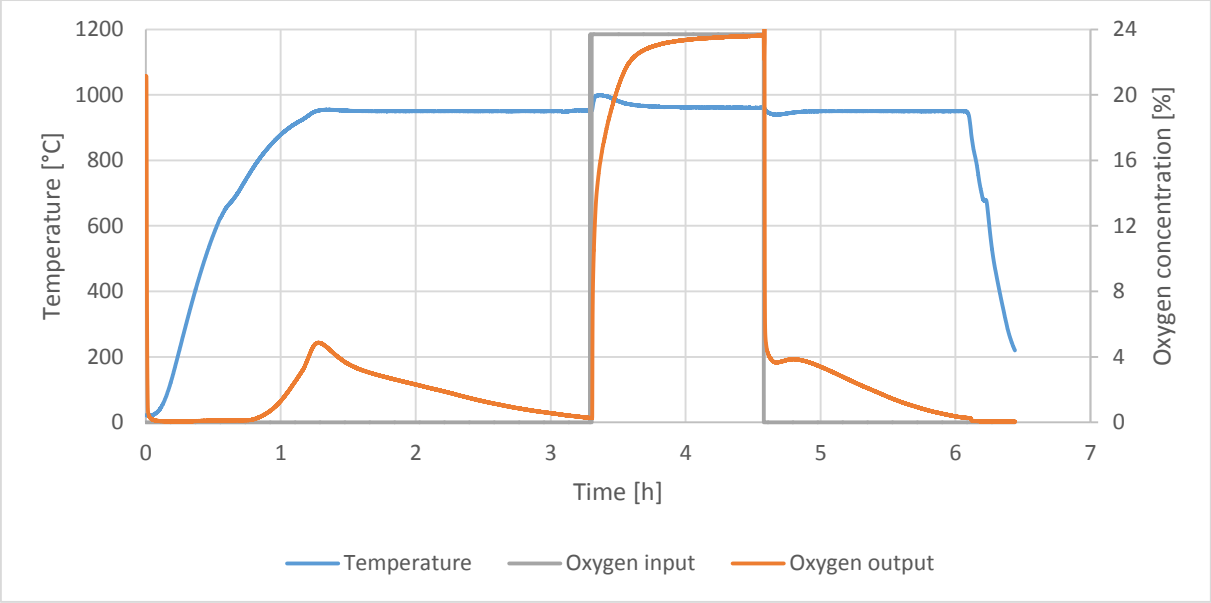



*Figure 28: Experiment A temperature/oxygen curves*

### 5.1.2. Experiment B

Experiment B (figure 29) was performed with the reduced material that was present at the end of experiment A. It was reheated, at which point it lost a little bit more oxygen (hour 0.8). At 1.9 hours, an oxygen step up to 10% was performed. After 1.5 hours, the experiment ended and the material was cooled down, resulting in 54% of the material being in the oxidized state. The gas flow was roughly 4L/min $N_2$ throughout and 0.5L/min $O_2$ during oxidation.

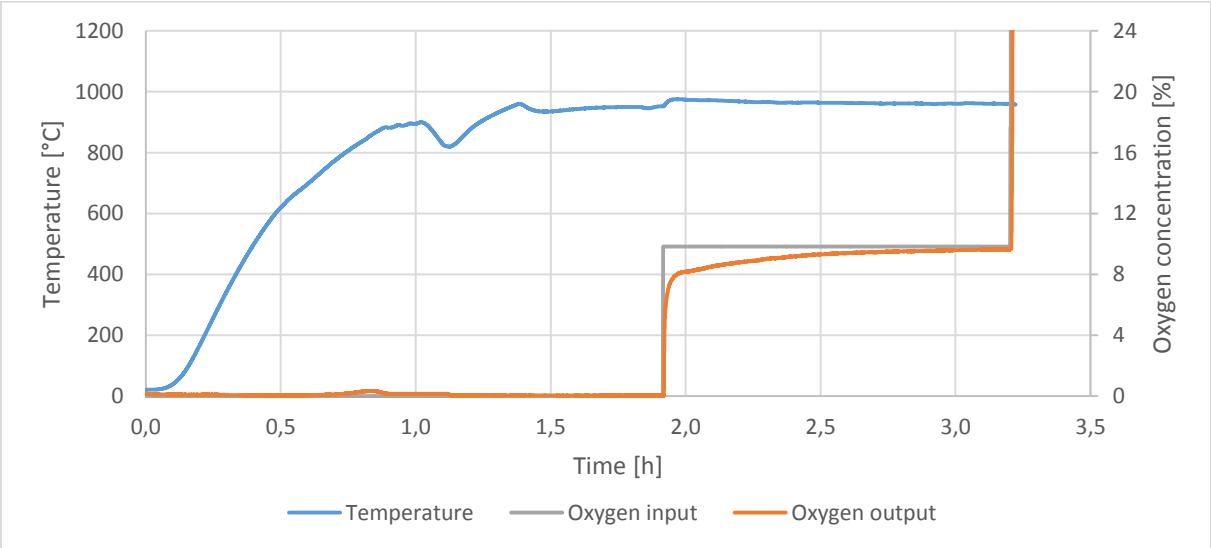



*Figure 29: Experiment B temperature/oxygen curves*

### 5.1.3. Experiment C

Experiment C (figure 30) continued with the oxidized material from experiment B and heated it up to 950°C with 2L/min $N_2$. After reduction was finished, the temperature was reduced to 900°C and the oxygen flow was increased to around 0.5L/min, resulting in an atmosphere of approximately 22% oxygen. Towards the end, the temperature was increased to 1000°C, which resulted in a decrease in offgas oxygen concentration at hour 4.4, indicating an increase in oxygen absorption. After 1h of oxidation, the oxygen was turned back off and reduction started again. In contrast to the previous experiments, the oxygen output concentration started at 8% and only took 40min instead of 1h30min, due to the increased temperature. The product was 100% reduced.

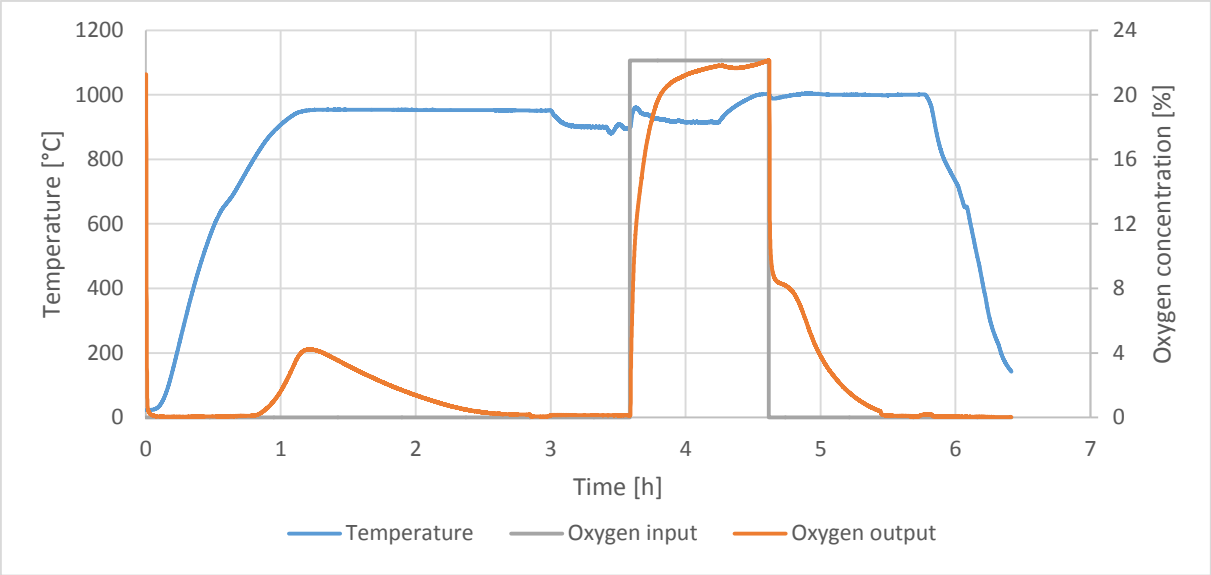



*Figure 30: Experiment C temperature/oxygen curves*

### 5.1.4. Experiment D

Experiment D (figure 31) used the material that was produced at the end of experiment C and heated it up to 900°C to reduce it completely, in case it had oxidized overnight, though this wasn't found to be the case. It was cooled back down to 800°C and 0.5L/min oxygen were added to the 2L/min nitrogen, resulting in approximately 20% oxygen in the feed. After 20min, the material was cooled down. This shorter reaction duration at a lower temperature than usual resulted in 41% conversion to the oxidized state.

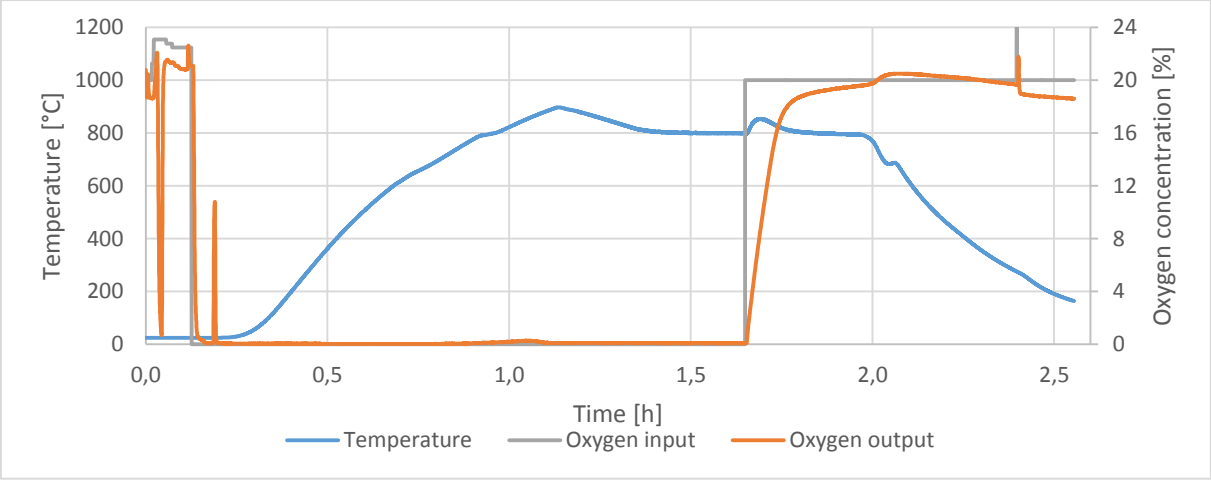



*Figure 31: Experiment D temperature/oxygen curves*

### 5.1.5. Experiment E

In contrast to the previous experiments, this experiment was performed with copper oxide powder (in its oxidized state). It was heated up to 950°C with 2.5L/min nitrogen (figure 32). It gave off a lot of oxygen initially (around 1.2h), but fell off quickly and continued even 2h after the reaction started. Thus, the temperature was increased to 1000°C. This resulted in an increase in oxygen output concentration and thus reaction speed, but after 2.5h since reaction start, the experiment was ended. Nonetheless, the material was found to be 65% reduced, though other problems ensued (see 5.2.3. Sintering).

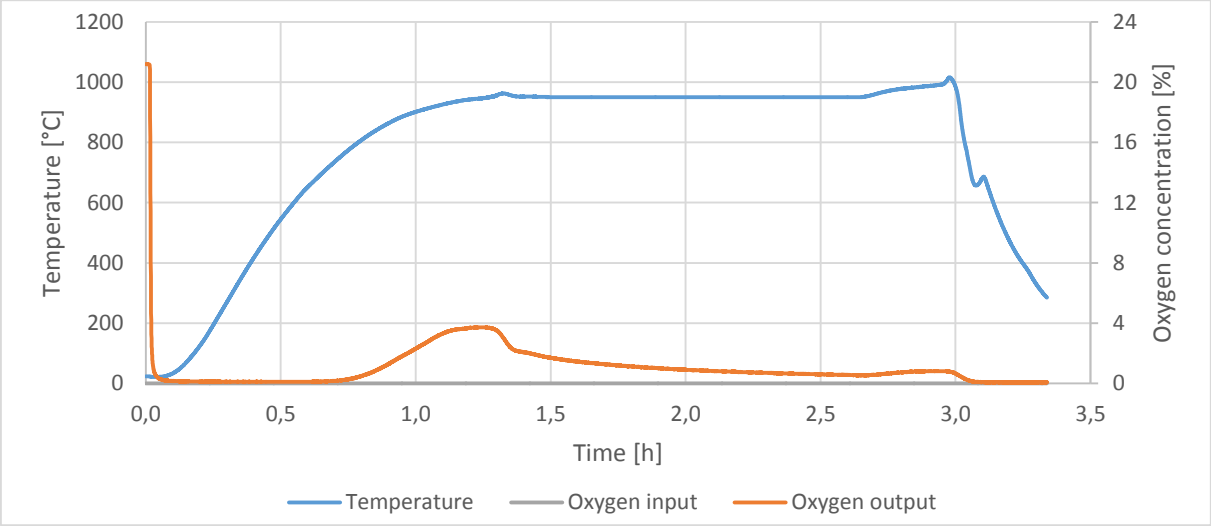



*Figure 32: Experiment E temperature/oxygen curves*

### 5.1.6. Experiment F

Experiment F (figure 33) was performed with the granulate from experiment D, which was heated up to 950°C in 2L/min nitrogen and 0.5L/min oxygen. At 2.5h, the oxygen was reduced to 0 and the nitrogen was increased to 2.5L/min, initiating reduction which ran for 1.2h before the material was cooled down. This resulted in a 100% conversion to the reduced state.

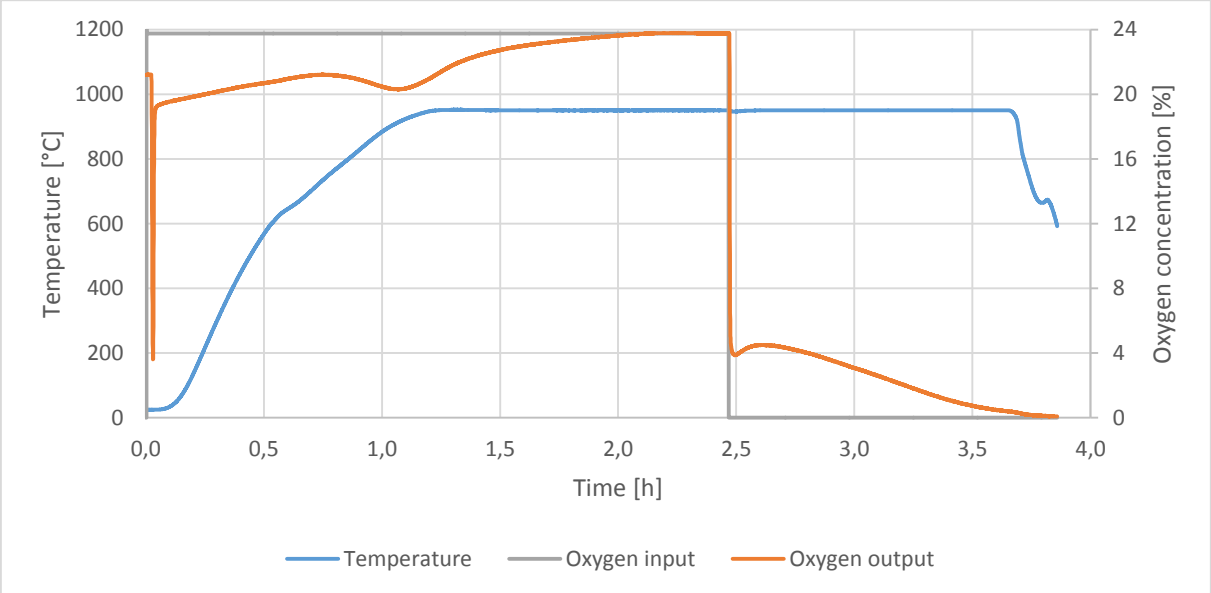



*Figure 33: Experiment F temperature/oxygen curves*

### 5.1.7. Experiment G

Experiment G (figure 34) continues with the material that resulted from experiment F and heats it up to 950°C in 2L/min nitrogen. At 1.5h, 0.5L/min oxygen are added, resulting in an oxygen concentration of approximately 24%. The oxidation is allowed to go on for 1h, at which point the experiment ends with a 66% conversion to the oxidized state.

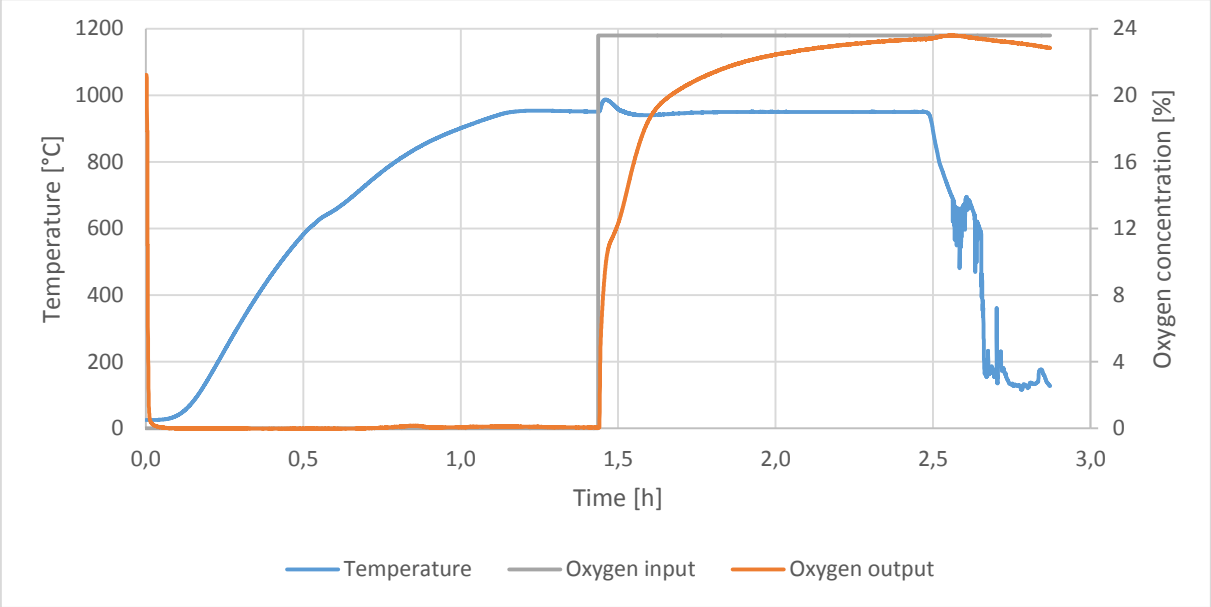



*Figure 34: Experiment G temperature/oxygen curves*

## 5.2. Evaluation

### 5.2.1. Yield

The yield of the reaction was analyzed by measuring the weight of the sample before and after the reaction and comparing it with the theoretical weight difference at full yield (equation 2.6).

*Table 9: Discharging process parameters*

| Experiment number | B | D | G |
|---|---|---|---|
| Initial weight [g] | 44.8 | 43.9 | 44.0 |
| Theoretical weight at full conversion [g] | 49.8 | 48.9 | 48.9 |
| Final weight [g] | 47.6 | 46.3 | 47.8 |
| Yield [%] | 54.2 | 40.6 | 65.7 |

The discharging reaction was tested with 3 different parameter settings, as shown in table 9. The distribution between CuO and $Cu_2O$ is shown in figure 35.

The discharging step in experiment B lasted 1h15min, longer than the other two, and was performed with an increased nitrogen flow of 4L/min, resulting in 4.5L/min total gas flow and around 11% oxygen concentration. The yield was 54.2% and thus below the yield of experiment 1.

In experiment D, the discharging reaction duration was reduced to 20min with the same gas flow and resulted in a lower yield of 40.6%.

Experiment G was performed with 2L/min nitrogen and 0.5L/min oxygen flow, yielding an oxygen concentration of 20%. The discharging reaction was ended after 1h and was found to have a yield of 65.7%.

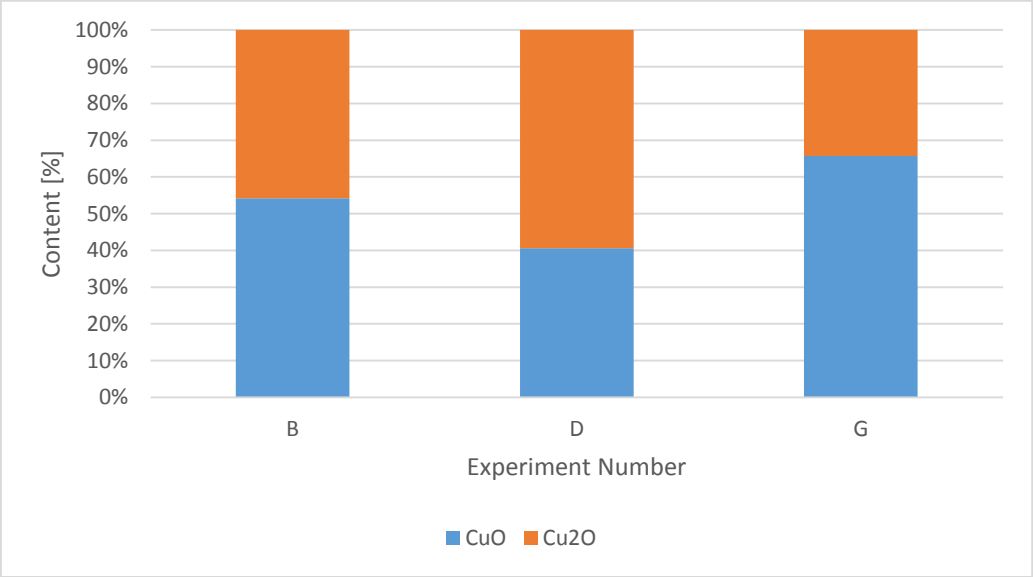



*Figure 35: Copper oxide contents after various discharging procedures, yield is equivalent to the CuO content (blue)*

The yield of the charging process, which is equal to the $Cu_2O$ content and was measured by XRD, was found to be consistently above 99%. When calculated from the mass change, as seen in equation 2.6, the yield was found to be above 100% with all experiments, as seen in figure 36, and 105% on average.

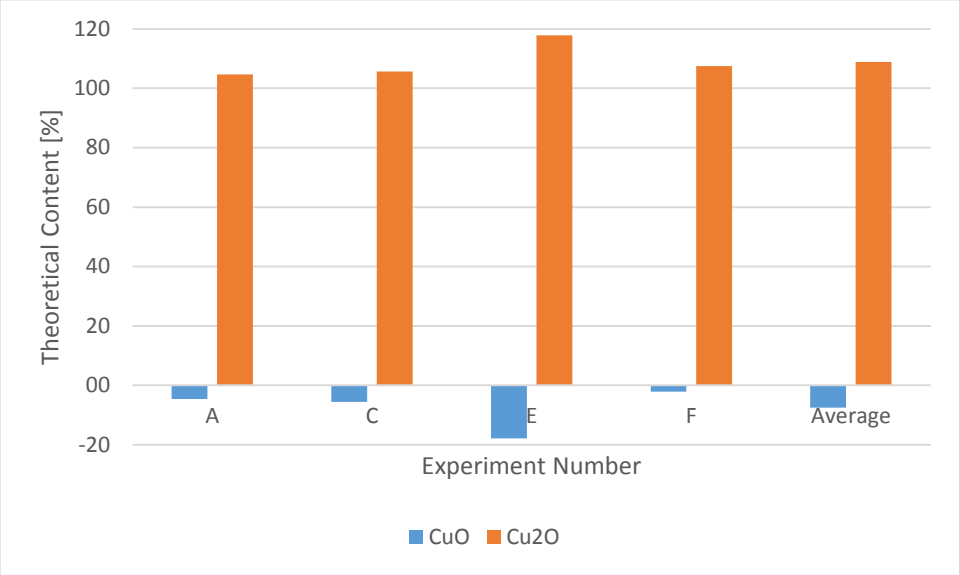



*Figure 36: Calculated charge yields after various charging procedures, yield is equivalent to the Cu2O content (orange)*

### 5.2.2. Oxygen Binding/Emission

The exact calculation of the bound/released oxygen is not possible with the current configuration of the apparatus, as there is no instrument measuring the off-gas flow. This is however an important value that is needed for the mass balance and which is not equal to the gas input, as the apparatus is leaking between 10% and 20% of the gas. It is also not known whether the gas leaks before or after it reacts with the material. Another change in gas flow occurs because of the binding/emission of oxygen.

$$V = \left| \sum_{t=t_{Start}}^{t_{End}} \left(c_{O_2,Out,t} - c_{O_2,In,t}\right) \cdot \left(\dot{V}_{N_2,t} + \dot{V}_{O_2,t}\right) \cdot \Delta t \right| - V_{empty} \qquad (5.1)$$

An approximate value for the absorbed oxygen volume can be calculated (as shown in equation 5.1) by integrating the difference of oxygen concentration between in- and output $c_{O_2,Out,t} - c_{O_2,In,t}$, multiplying it with the total gas flow $\dot{V}_{N_2,t} + \dot{V}_{O_2,t}$ and subtracting the dead volume $V_{empty}$. Although in the case of our discontinuous measurements, we have to sum it up instead of integrating.

Attempts can be made to correct the gas flow, but there are problems that cannot be fixed, like the leakage not being constant. The gas flow change from binding/emission cannot be calculated from the change in oxygen concentration, because when turning the oxygen input on or off, there is a large dead volume and the off-gas oxygen curve of an empty reactor is not a perfect step function but instead takes more than one minute to approach the input value.

A sample oxygen/temperature curve starting with CuO is shown in figure 37. The temperature (blue line) is rising during the first hour until it reaches 950°C (~15K/min). But before that, as soon as the temperature reaches 800°C, the copper oxide is being reduced and gives off oxygen into the offgas (orange line). The reduction took 2h and gave off 3.83L oxygen.

When the reduction was finished, the temperature was decreased to 900°C and the oxygen input raised from 0 to 22% (grey line), initiating the oxidation. The oxygen output slowly approaches the input, indicating a decreasing reaction speed. When this happens, the reaction is usually ended (and the operator proceeds with the next step or ends the experiment), as it is too impractical to continue at very low reaction speeds. With the reduction step, the reaction speed is high enough for the reaction to complete. The oxidation step, however, usually gets around halfway done. 3.41L of oxygen were absorbed over 1h.

For the next step, the oxygen input is once again turned off (and the temperature increased to 1 000°C) which initiates another reduction, which gave off 4.07L over 1h.

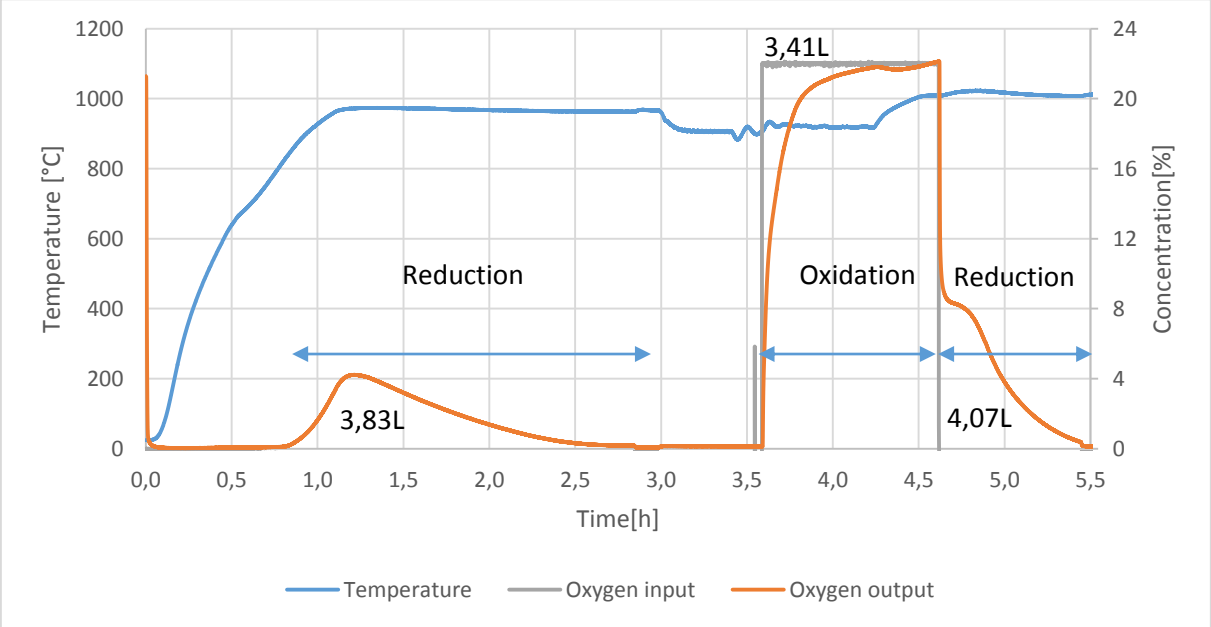



*Figure 37: Temperature and oxygen curve of an experiment*

Figure 38 shows the detailed curve that we get when we initiate oxidation. After half an hour, the reaction is nearly completed, as seen in the small difference between oxygen input and output concentration.

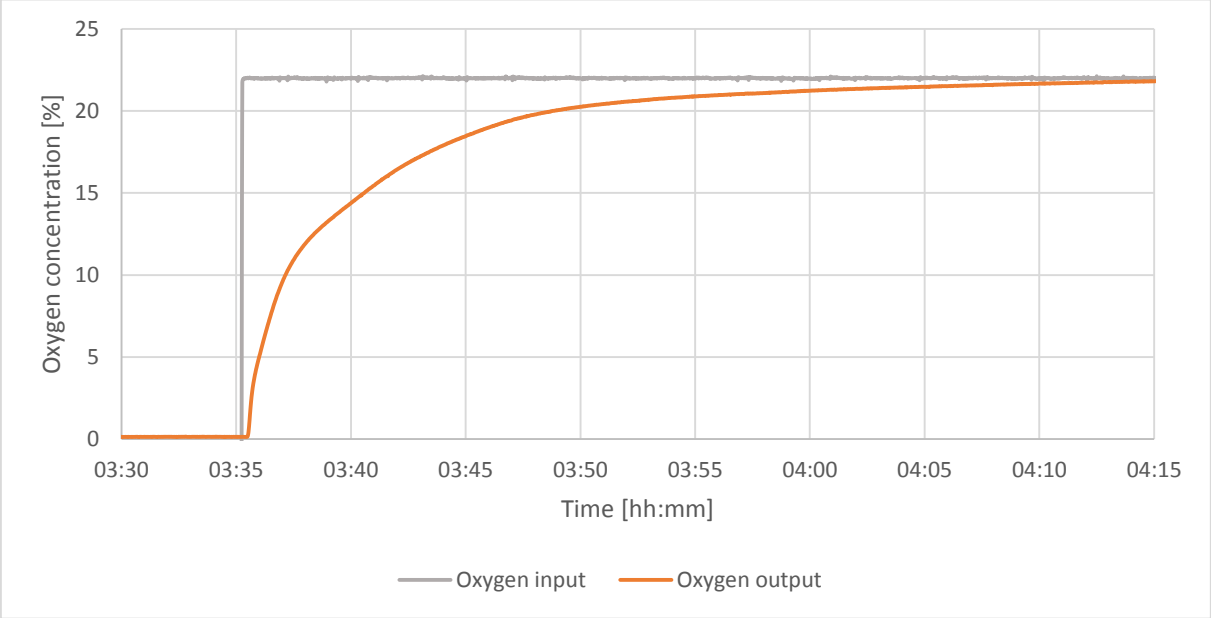



*Figure 38: Response curve to an oxygen step up (magnification of fig.29)*

### 5.2.3. Sintering

With the granulate (figure 39), there was some agglomeration during the charging process, which could however be destroyed by the application of light force. It was predominant in the charged form of the material, and might be avoided by a reduction of the charging temperature. However, the effect of temperature reduction on the charging process yield and duration would have to be examined. Another option would be the use of a rotary reactor.


*Figure 39: Copper oxide in the oxidized/discharged (top) and reduced/charged (bottom) state*

The copper oxide powder (figure 40) was found to completely sinter together, and could not be broken down with pure manual force. This disqualified the material for further experiments, as the mass transfer through the solid would have been to slow.

No sintering of the material to the reactor wall was detected.


*Figure 40: Copper oxide fine discharged powder (top) and charged solid block (bottom)*

### 5.2.4. Conclusion

The experiments were very successful and the charging process was successful with a >99% yield. The discharging process was also successful, but with a yield of only 65.7%. The charging yields and reaction durations are however consistent over multiple cycles, and agglomeration of the granulate was acceptable.

The copper oxide granulate thus shows signs of good cycle stability, but needs further testing. These tests should include multiple charge/discharge cycles with the same conditions to evaluate long-term degradation of the reactivity.

The copper oxide powder sintered together in the first experiment and thus disqualifies for use at 950°C, although a reduction of temperature could be examined.

Furthermore, the agglomeration at this particle size was acceptable and variation of the particle size and the particle size distribution should be examined.

## 5.3. Reactor Assessment

### 5.3.1. Strengths

The reactor can withstand temperatures up to 1050°C. This is far above the operating temperatures of most thermochemical heat storage materials studied in common research.

The reactor can be filled with up to 150cm³ of material, this amounts to 500g of copper oxide. The 50g used in these experiments only amount to 10% of the reactors capacity.

The set-up is modular and can be used for various purposes. Therefore, further parts can be built to replace the top, the reactor or to be added in between.

There are 5 thermocouples inside the reactor which can be bent to measure at different locations in the reactor. Thus, the temperature can be monitored across different heights or distances from the center.

### 5.3.2. Weaknesses

There is significant wear: the gaskets become brittle and the screws creep to the point where nuts have to be unscrewed with immense force or sawn off.

There is a significant dead volume which causes the gas analysis to be delayed and somewhat diffuse. This poses a problem, as the absorption or desorption of the reactive gas causes a change in gas flow. This has to be accounted for, but can only be done so if the gas analysis is somewhat simultaneous and if the source/sink of the reactive gas is known (desorption/adsorption or just release from/storage to a dead zone).

# 6. Future

Further tests should be run with copper oxide, as it shows promising results. The particle size should be reduced and the particle size distribution should be narrowed. This might increase the yield of the discharge reaction (currently approximately 66%) and reduce the reaction duration (currently >1h). It should be examined, what particle size the sintering becomes too intense at. Additionally, the degradation of the material and its effect on the yield and reaction duration after multiple cycles should be studied.

Furthermore, the reactor can be used to test other materials, such as cobalt oxide and manganese oxide. These have the added benefit that they react 100°C below copper oxide, which would reduce wear of the reactor.

To improve the evaluation of the reactions, the exit gas flow could be monitored by an MFC or a gas flow calibrator. With the data from the gas analyzer, this would allow an accurate calculation of the absorbed and desorbed oxygen.

Slight adjustments for the oxygen calibration could be made, as the oxygen content is 20,95% in dry air, but not in common humid air.

# Nomenclature

| Variable | Unit | Description |
|---|---|---|
| $Q_V$ | $\frac{J}{m^3}$ | Volumetric heat capacity |
| $c$<br>$C$ | $\frac{mol}{m^3}$ | Concentration |
| $\rho$ | $\frac{kg}{m^3}$ | Density |
| $T$ | $K$ | Temperature |
| $\nu_A$ | $1$ | Stoichiometric coefficient |
| $H$ | $J$ | Enthalpy |
| $M$ | $\frac{kg}{mol}$ | Molar mass |
| $\Delta_r H$ | $\frac{J}{mol}$ | Reaction enthalpy |
| $H_{m,i}$ | $\frac{J}{mol}$ | Molar enthalpy of species i |
| $\Delta_r G$ | $\frac{J}{mol}$ | Reaction Gibb's energy |
| $\Delta_r S$ | $\frac{J}{K \cdot mol}$ | Reaction entropy |
| $\xi$ | $1$ | Reaction coordinate |
| $p$ | $Pa$ | Pressure |
| $\mu_i$ | $\frac{J}{mol}$ | Chemical potential |
| $K$ | $1$ | Equilibrium constant |
| $\Delta_r H^{\ominus}$ | $\frac{J}{mol}$ | Standard reaction enthalpy |
| $R$ | $\frac{J}{K \cdot mol}$ | Gas constant |
| $p_C$ | $Pa$ | Partial pressure |
| $p^{\ominus}$ | $Pa$ | Standard pressure |
| $N_B$ | $mol$ | Amount of substance |
| $b$ | $1$ | Stoichiometric factor |
| $\rho_B$ | $\frac{mol}{m^3}$ | Molar density |
| $V$ | $m^3$ | Volume |

| Variable | Unit | Description |
|---|---|---|
| $r$<br>$R$ | $m$ | Radius |
| $t$ | $s$ | Time |
| $k''$ | $\frac{m}{s}$ | Reaction rate constant of a reaction at a surface |
| $\tau$ | $s$ | Reaction duration |
| $X_B$ | 1 | Conversion |
| $S_{ex}$ | $m^2$ | External surface |
| $k_g$ | $\frac{m}{s}$ | Mass transfer coefficient |
| $d_p$ | $m$ | Diameter |
| $y$ | 1 | Mole fraction |
| $D$ | $\frac{m^2}{s}$ | Diffusion coefficient |
| $Sc$ | 1 | Schmidt number |
| $Re$ | 1 | Reynolds number |
| $\mu$ | $\frac{kg}{m \cdot s}$ | Viscosity |
| $u$ | $\frac{m}{s}$ | Speed |
| $Q_A$ | $\frac{mol}{m^2 \cdot s}$ | Molar flux |
| $Q_{As}$ | $\frac{mol}{m^2 \cdot s}$ | Molar flux |
| $Q_{Ac}$ | $\frac{mol}{m^2 \cdot s}$ | Molar flux |
| $D_e$ | $\frac{m^2}{s}$ | Diffusion coefficient |
| $\eta$ | % | Yield |
| $m$ | $kg$ | Mass |
| $n_C$ | $mol$ | Amount of substance |
| $u$ | $V$ | Voltage |
| $n$ | 1 | Digital output |
| $\dot{V}$ | $\frac{m^3}{s}$ | Gas flow |

# Bibliography


[1] Kumar, Arun; Schei, T.; Ahenkorah, A.; Rodriguez, R. Caceres; Devernay, J. M.; Freitas, M. et al. (2011): Hydropower IPCC Special Report on Renewable Energy Sources and Climate Change Mitigation. Chapter 5: Cambridge University Press, Cambridge and New York

[2] Kaldellis, John K.; Zafirakis, D. (2011): The wind energy (r)evolution: A short review of a long history. In *Renewable Energy* 36 (7), pp. 1887–1901. Available online at http://www.sciencedirect.com/science/article/pii/S0960148111000085

[3] de Souza, Victor Barbosa; Cerqueira, Niander Aguiar; Silva, Priscila Dias (2015): PHOTOVOLTAIC ENERGY AS A SOLUTION FOR ENERGETIC CRISIS: ANALYSIS OF TECHNICAL FEASIBILITY OF ITS IMPLEMENTATION IN BUILDING AN INSTITUTION OF HIGHER EDUCATION IN ITAPERUNA-RJ CITY. In *REINPEC-Revista Interdisciplinar Pensamento Científico* 1 (1)

[4] Chan, C. W.; Ling-Chin, J.; Roskilly, A. P. (2013): A review of chemical heat pumps, thermodynamic cycles and thermal energy storage technologies for low grade heat utilisation. In *Applied Thermal Engineering* 50 (1), pp. 1257–1273. Available online at http://www.sciencedirect.com/science/article/pii/S1359431112004620

[5] Kearney, D.; Kelly, Bruce; Herrmann, Ulf; Cable, Robert; Pacheco, J.; Mahoney, R. et al. (2004): Engineering aspects of a molten salt heat transfer fluid in a trough solar field. In *Energy* 29 (5), pp. 861–870

[6] I. E.A. Statistics (2015): Excerpt from Electricity Information 2015. In *International Energy Agency*

[7] Eurostat (2015): Supply of electricity - monthly data. Available online at http://ec.europa.eu/eurostat/product?code=nrg_105m&language=en&mode=view, updated on 12/15/2015, checked on 1/7/2016

[8] Hamilton, James D. (2014): The Changing Face of World Oil Markets. In *National Bureau of Economic Research Working Paper Series* No. 20355. Available online at http://www.nber.org/papers/w20355

[9] Feiveson, Harold; Mian, Z.; Ramana, M. V.; Hippel, Fv (2011): Managing Spent Fuel from Nuclear Power Reactors Experience and Lessons from Around the World. In *International Panel on Fissile Materials*

[10] Zarfl, Christiane; Lumsdon, AlexanderE.; Berlekamp, Jürgen; Tydecks, Laura; Tockner, Klement (2015): A global boom in hydropower dam construction. In *Aquat Sci* 77 (1), pp. 161-170. Available online at http://dx.doi.org/10.1007/s00027-014-0377-0

[11] Boccard, Nicolas (2009): Capacity factor of wind power realized values vs. estimates. In *Energy Policy* 37 (7), pp. 2679–2688. Available online at http://www.sciencedirect.com/science/article/pii/S030142150900144X

[12] IRENA (2014): Renewable Power Generation Costs in 2014: International Renewable Energy Agency

[13] Global Wind Energy Council (2015): Annual market update 2014: Global Wind Report. In *GWEC, Brussels*

[14] Chow, T. T. (2010): A review on photovoltaic/thermal hybrid solar technology. In *Applied Energy* 87 (2), pp. 365–379. Available online at http://www.sciencedirect.com/science/article/pii/S0306261909002761

[15] SolarPower Europe (2015): Global market outlook for solar power 2015-2019. In *Euoropean Photovoltaic Industry Association, Bruxelles, Tech. Rep*

[16] Kleijn, Rene; Van der Voet, Ester (2010): Resource constraints in a hydrogen economy based on renewable energy sources: An exploration. In *Renewable and Sustainable Energy Reviews* 14 (9), pp. 2784–2795

[17] Ren21, Renewables (2015): Global Status Report Key Findings, 2015

[18] California Independent System Operator: Fast Facts. Available online at https://www.caiso.com/Documents/FlexibleResourcesHelpRenewables_FastFacts.pdf, checked on 12/15/2015

[19] Whittingham, M. Stanley (2012): History, evolution, and future status of energy storage. In *Proceedings of the IEEE* 100 (Special Centennial Issue), pp. 1518–1534

[20] Bullough, Chris; Gatzen, Christoph; Jakiel, Christoph; Koller, Martin; Nowi, Andreas; Zunft, Stefan (Eds.) (2004): Advanced adiabatic compressed air energy storage for the integration of wind energy (22)

[21] Bolund, Björn; Bernhoff, Hans; Leijon, Mats (2007): Flywheel energy and power storage systems. In *Renewable and Sustainable Energy Reviews* 11 (2), pp. 235–258. Available online at http://www.sciencedirect.com/science/article/pii/S1364032105000146

[22] The Economist: Packing some power. In *The Economist*. Available online at http://www.economist.com/node/21548495?frsc=dg|a, checked on 12/15/2015

[23] Office of Energy Efficiency & Renewable Energy: Concentrating Solar Power Thermal Storage System Basics. Available online at http://energy.gov/eere/energybasics/articles/concentrating-solar-power-thermal-storage-system-basics, checked on 12/15/2015

[24] Sharma, S. D.; Sagara, Kazunobu (2005): Latent Heat Storage Materials and Systems: A Review. In *International Journal of Green Energy* 2 (1), pp. 1–56

[25] Pardo, P.; Deydier, A.; Anxionnaz-Minvielle, Z.; Rougé, S.; Cabassud, M.; Cognet, P. (2014): A review on high temperature thermochemical heat energy storage. In *Renewable and Sustainable Energy Reviews* 32, pp. 591–610. Available online at http://www.sciencedirect.com/science/article/pii/S1364032113008289

[26] Levenspiel, O., 1999. Chemical reaction engineering. *Industrial & engineering chemistry research*, *38*(11), pp.4140-4143.

[27] Protostack Pty Ltd: ATmega168A Pulse Width Modulation – PWM. Available online at http://www.protostack.com/blog/2011/06/atmega168a-pulse-width-modulation-pwm/, checked on 12/15/2015

# Annex A: Equipment Specifications

## MFCs

MKS 1259C and MKS 1579A

various types ranging from 1slm to 20slm

## Analyzer

ABB EL3020

## PLC

B&R CP1585

with modules

2x X20AO4622
3x X20AI4622
1x X20AT6402
1x X20AI8221

## Heating Mantle

## Steel

1.4841

## Gaskets

## Isolating Pad

Bauchinger

Keramikfasermatte 13

500x610x13mm  128 kg/m² Organisch gebunden für den Hochtemperaturbereich
Dauereinsatztemperatur (Prüfung nach EN 1094-7) bis 900°und Kurzzeitig bis 1260 °C
Schwindungsverhalten 2%

## Thermocouples

6 Stück TC Direct Art.-Nr. 405-338
1,5mm Durchmesser x 300mm Länge
Typ K Inconel 600 oxidationsbeständig bis 1100°C
Miniaturthermoelementstecker beständig bis 135°C

## Resistor

Arcol HS50 Series Aluminium Housed Axial Panel Mount Resistor, 6.8Ω ±5% 50W

## Hoses

**PUN 6X4 BLAU: Polyurethan-Schlauch 6 x 4 mm, blau**

**Verpackungseinheit:** 50 M
**Schlauch Ø außen x innen [mm]:** 6 x 4
**Farbe:** blau
**Betriebsdruck [bar]:** 14
**min. Biegeradius [mm]:** 15
**Gewicht:** 23g / mtr
**Zollwarennummer:** 39173900

**Vorteile:**
•kleiner Biegeradius durch besondere Flexibilität
•sehr gute Kälteflexibilität und Rückstelleigenschaften
•knick- und abriebfest
•beständig gegen aliphatische Kohlenwasserstoffe und die meisten Schmierstoffe
•alterungsbeständig in Sauerstoff
•schleppkettentauglich (min. Biegeradius entspricht 10 x Außendurchmesser)
**Werkstoff:** Polyester-Polyurethan
**Temperaturbereich:** -35°C bis max. +60°C
**Shore-Härte:** 97 A
**Rollenlänge:** 50 mtr.
**Richtwerte für die Druckausnutzung bei Temperaturbelastung:** bis +20°C: 100%, +30°C: 80%, +40°C: 68%, +50°C: 58%, +60°C: 50%

## Annex B: Drawings

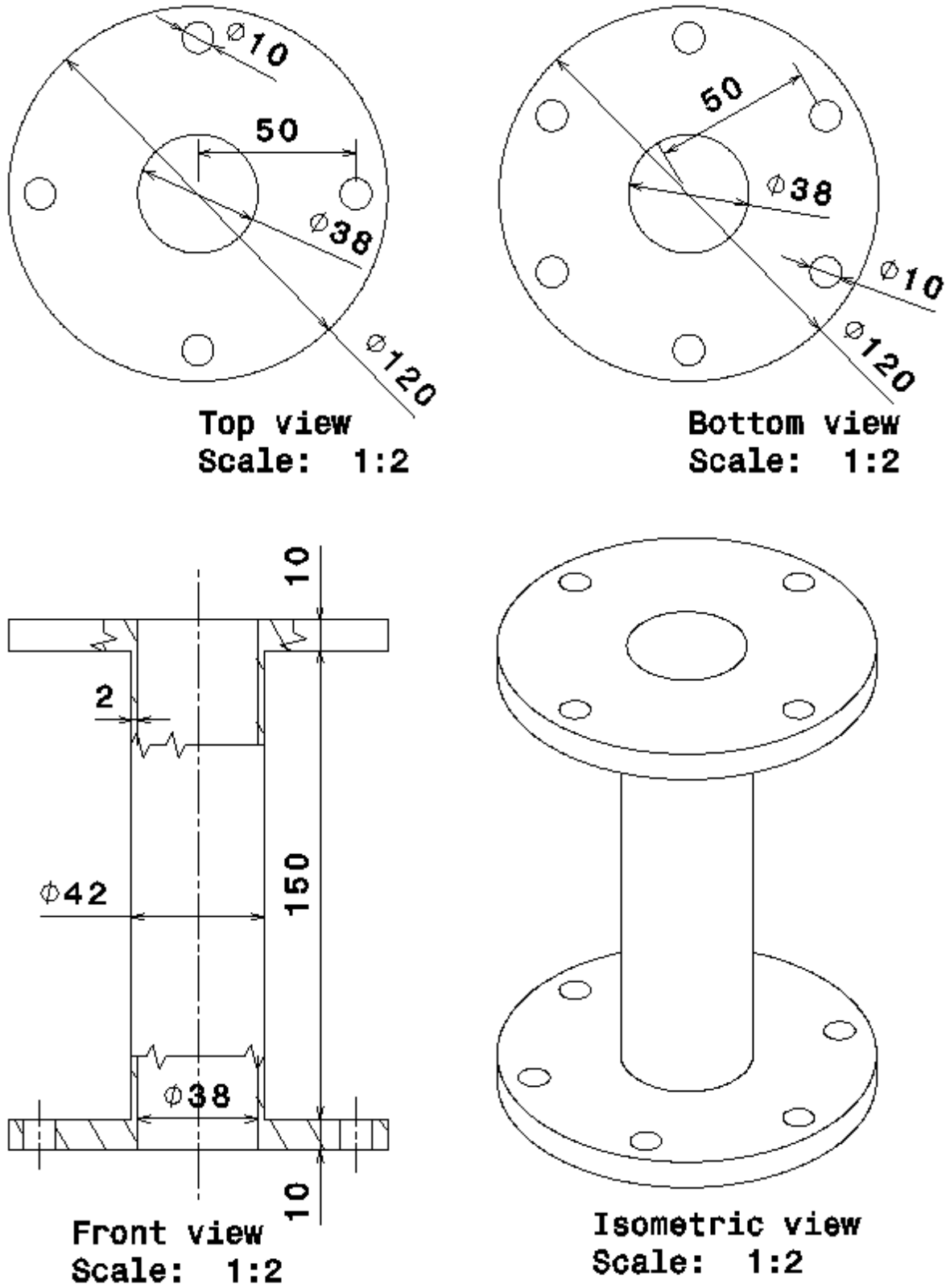



*Figure 41: Tubular reactor drawings*

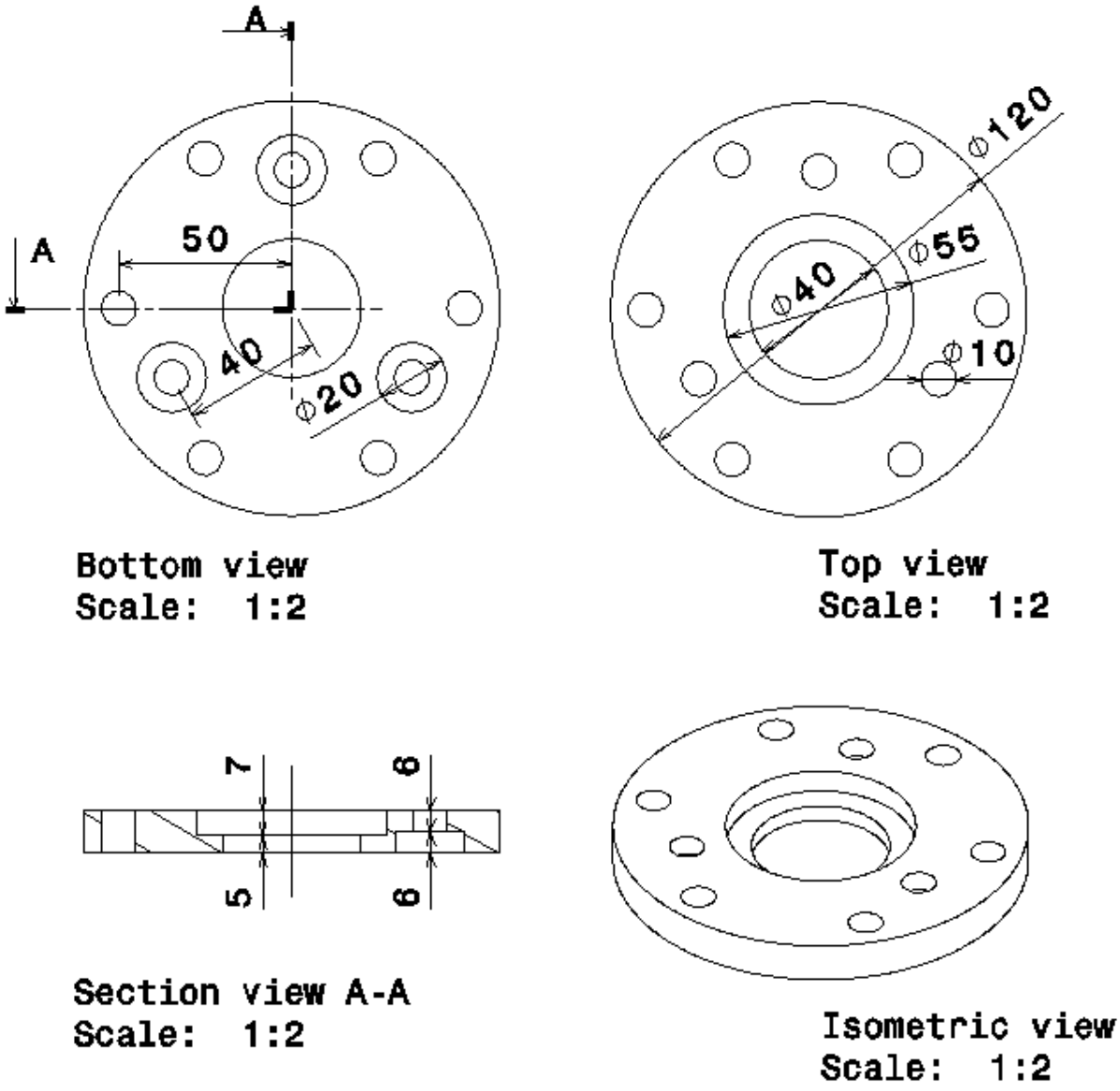



*Figure 42: Socket drawings*

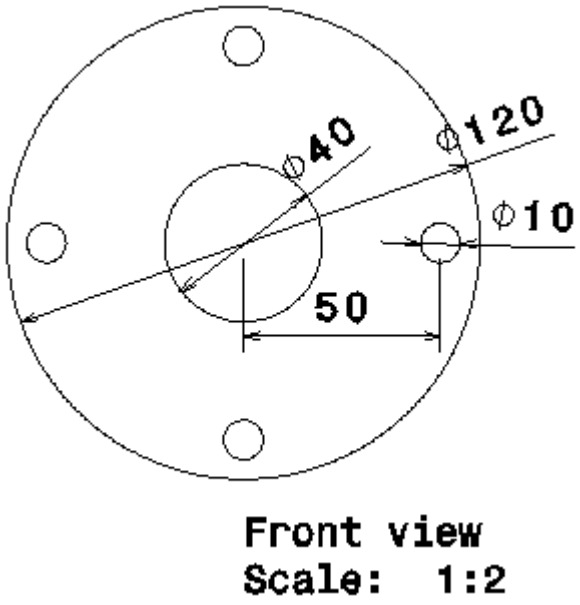



*Figure 43: Gasket between the lid and Reactor*

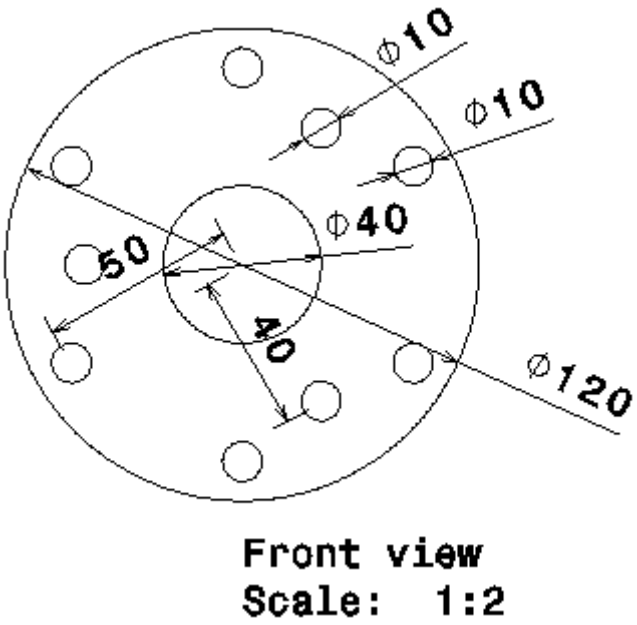



*Figure 44: Gasket between the reactor and socket*

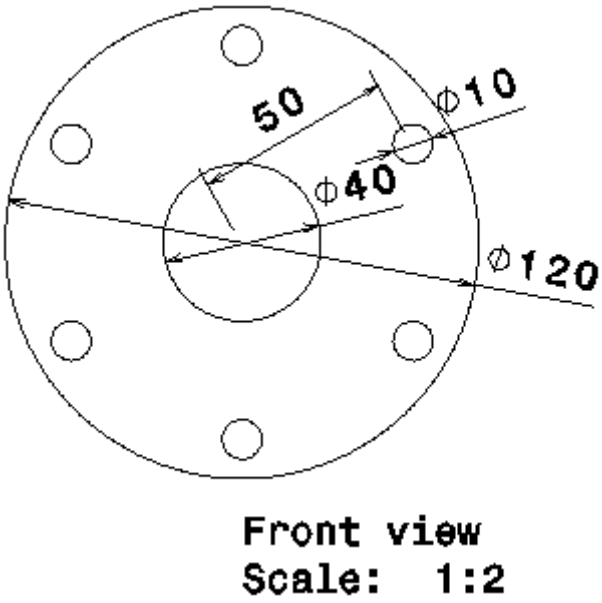



*Figure 45: Gasket between the socket and preheater*

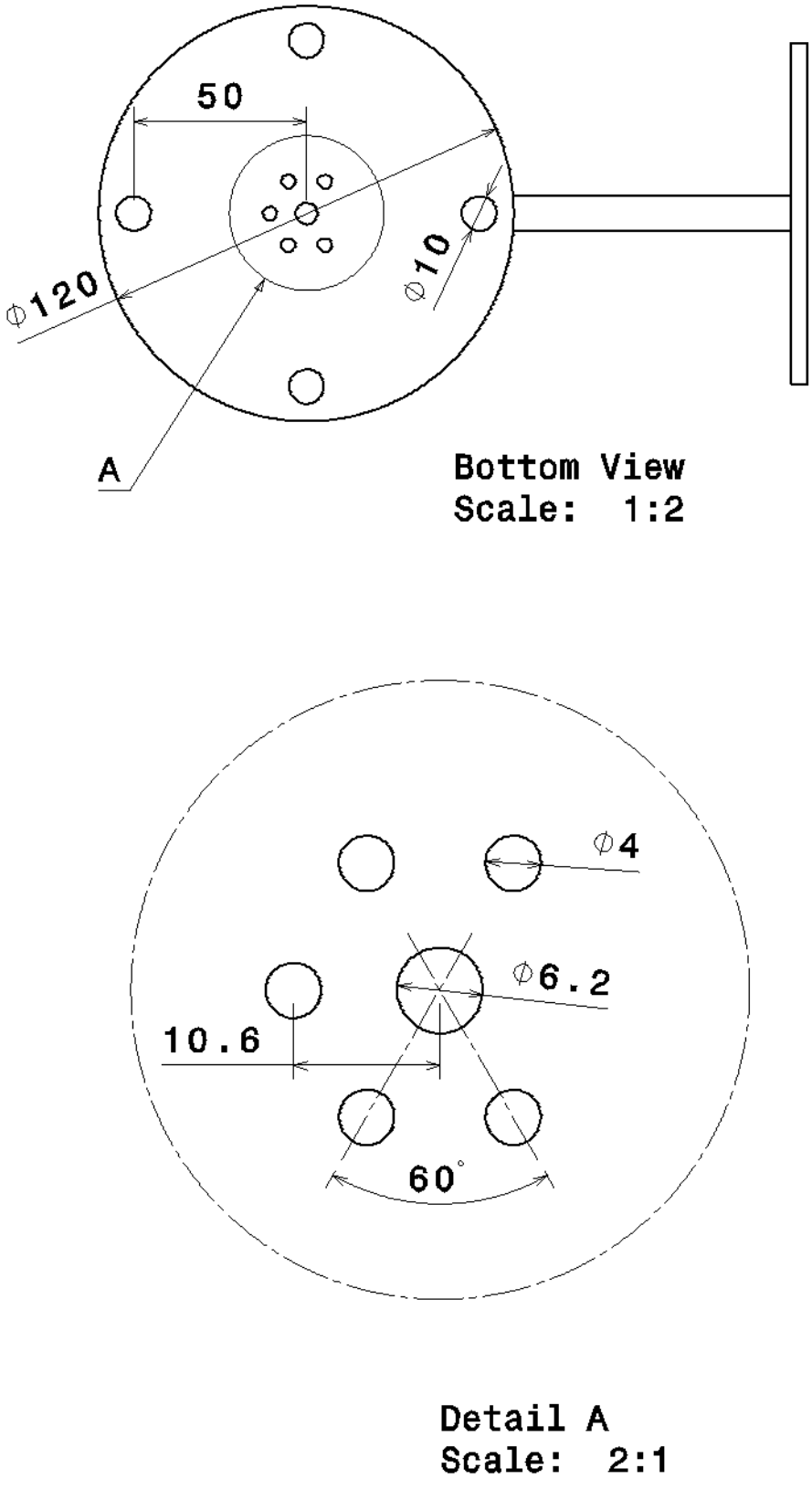



*Figure 46: Lid drawing bottom view and detail A*

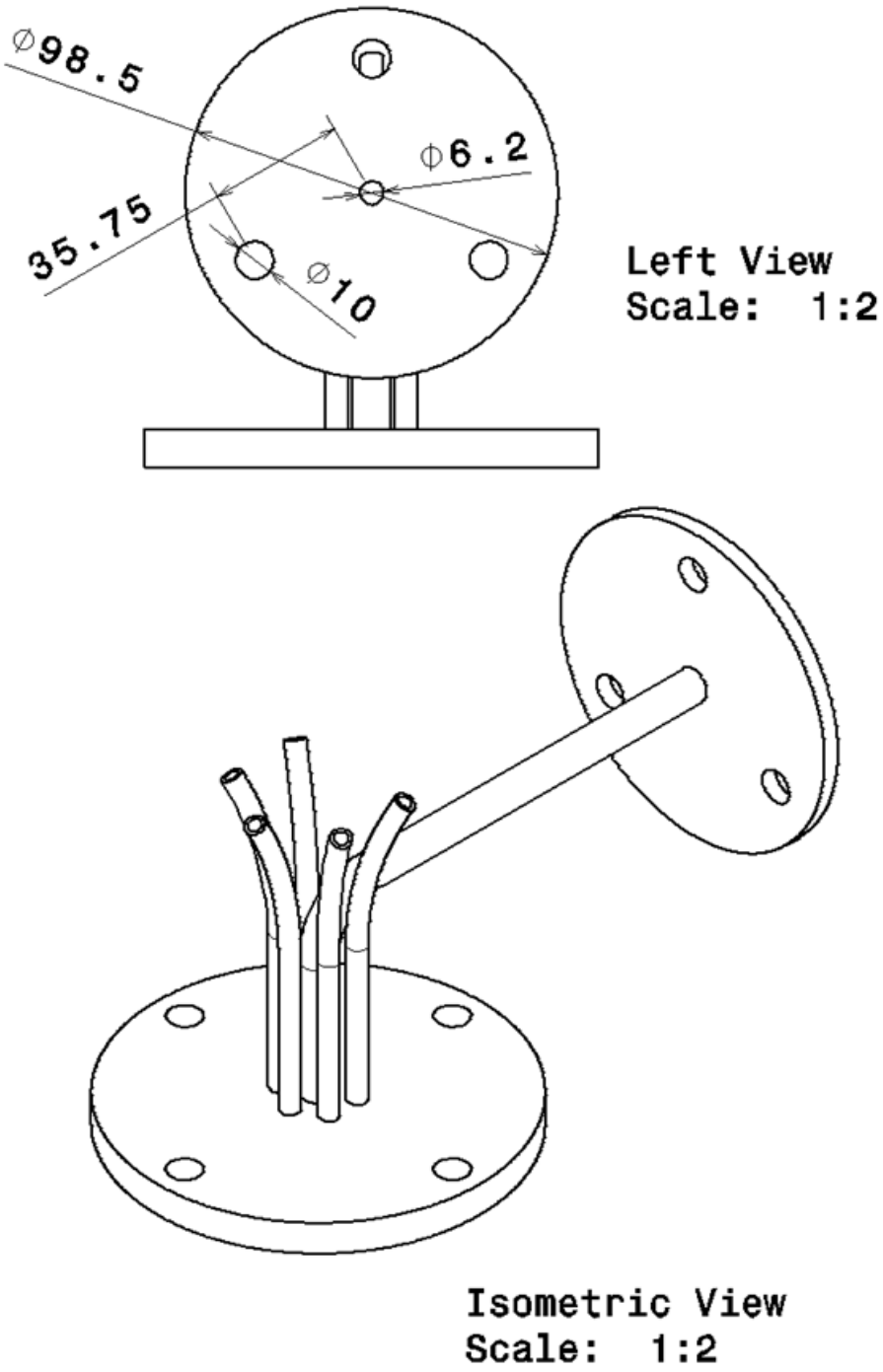



*Figure 47: Lid drawing left and isometric view*

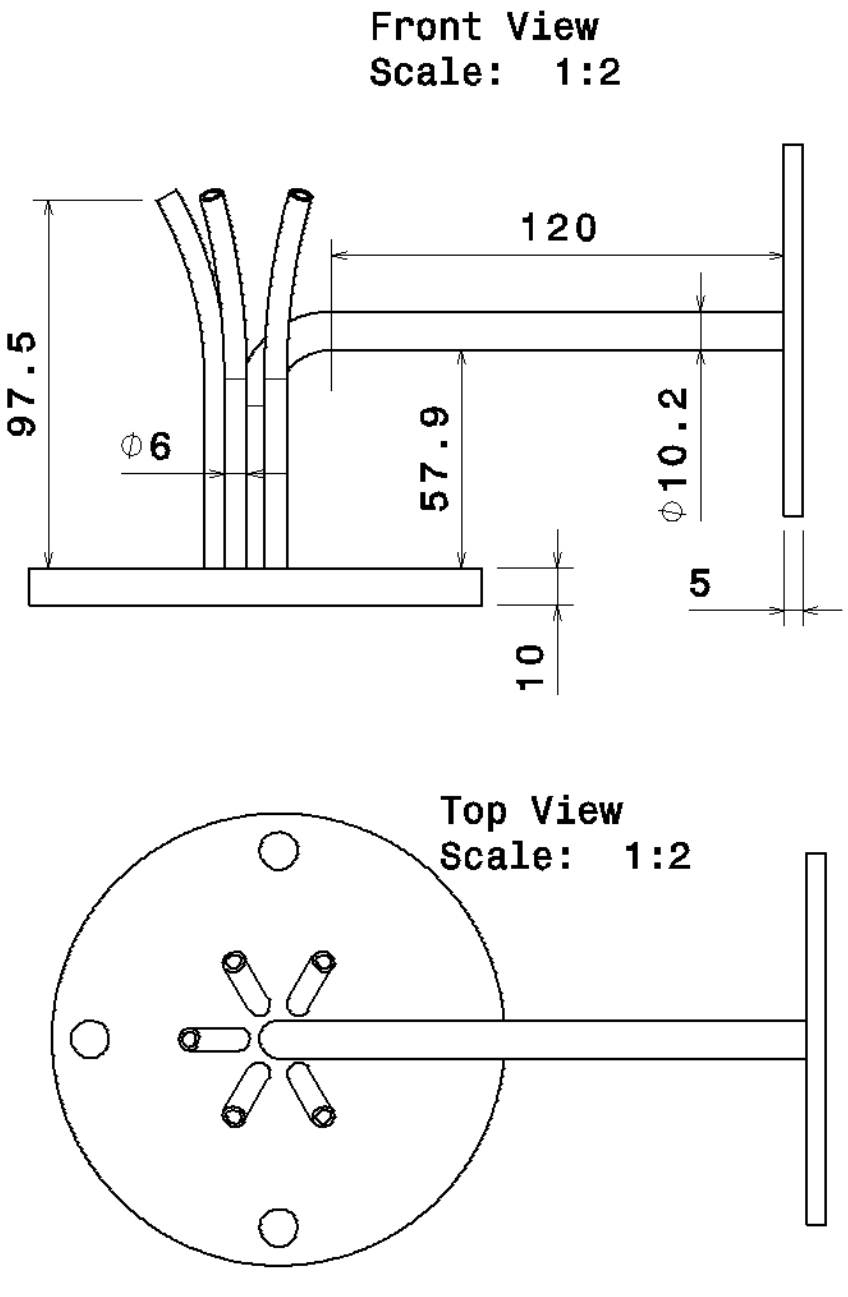


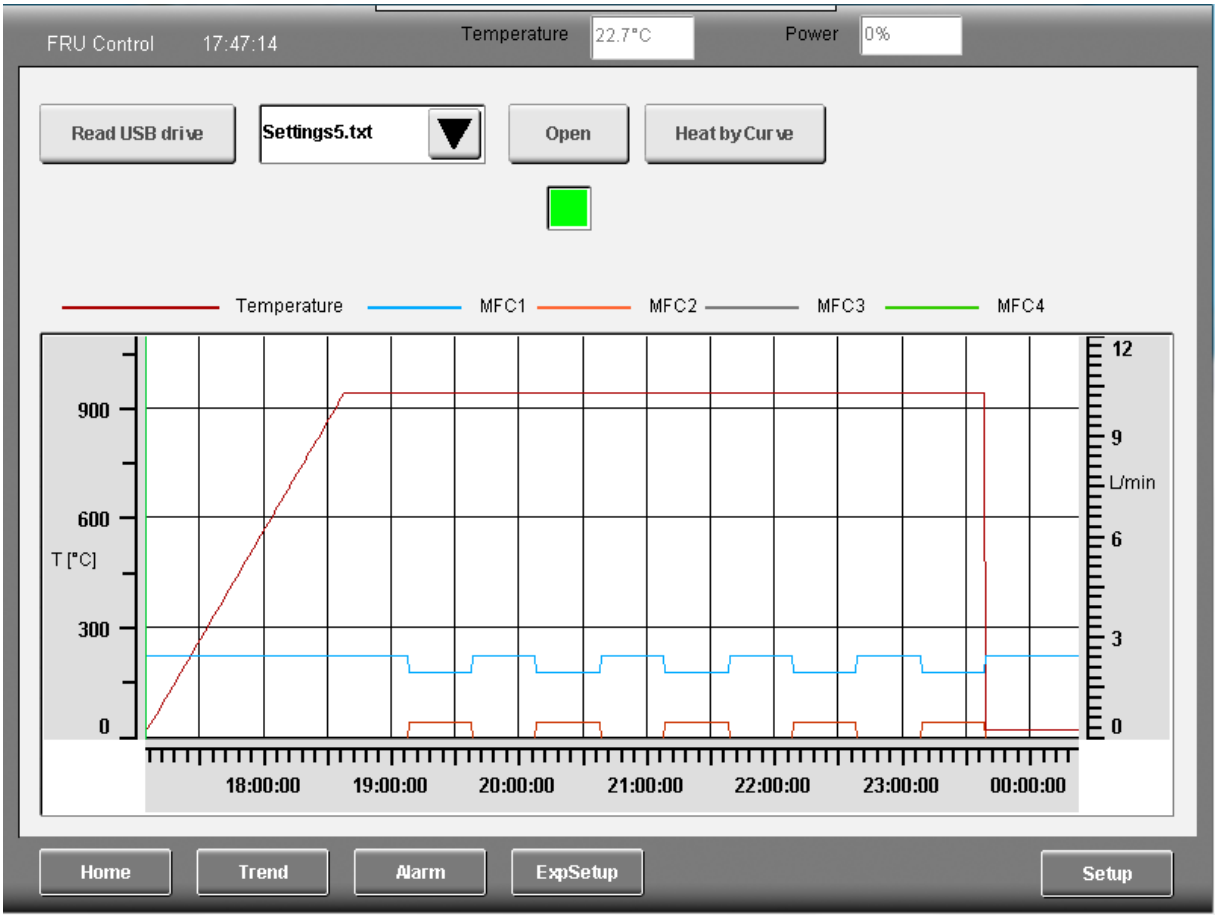

*Figure 48: Lid drawing front and top view*

# Annex C: T/MFC Program Example

## Code

| /Time | Temp | MFC1 | MFC2 | MFC3 | MFC4 |
|---|---|---|---|---|---|
| 0 | 25 | 2.5 | 0 | 0 | 0 |
| R10 | 950 | 2.5 | 0 | 0 | 0 |
| +1800 | 950 | 2.5 | 0 | 0 | 0 |
| +1 | 950 | 2 | 0.5 | 0 | 0 |
| +1800 | 950 | 2 | 0.5 | 0 | 0 |
| +1 | 950 | 2.5 | 0 | 0 | 0 |
| REP4x4 | | | | | |
| +1 | 25 | 2.5 | 0 | 0 | 0 |
| +2700 | 25 | 2.5 | 0 | 0 | 0 |
| +1 | 25 | 2.5 | 0 | 0 | 0 |

## Resulting Curve



# Annex D: Code

[Comments are marked green]

## altControl (controls gas flow and heating)

PROGRAM _INIT

```
setValue     := 200;                    //Room temperature * 10 as initial set value
start        := FALSE;                  //PID-Controller off
tuningRequest := LCPID_TUNE_REQU_OFF;

(* Parameters for PID tuning *)
LCPIDTune_0.Y_min        := -32763;
LCPIDTune_0.Y_max        := 32763;
LCPIDTune_0.Y0           := 0;
LCPIDTune_0.Y1           := 10000;
LCPIDTune_0.X0           := 250;
LCPIDTune_0.X_min        := -10000;
LCPIDTune_0.X_max        := 10000;
LCPIDTune_0.P_manualAdjust := 0;
LCPIDTune_0.I_manualAdjust := 0;
LCPIDTune_0.D_manualAdjust := 0;
LCPIDTune_0.pAddPar      := ADR(addParameter);
LCPIDTune_0.pOptions_osc  := ADR(oscOptions);
LCPIDTune_0.pOptions_step  := ADR(stepOptions);

(* Parameters for PID controller *)
LCPID_0.A       := 0;
LCPID_0.Y_man    := 0;
LCPID_0.Y_fbk    := 0;
LCPID_0.hold_I   := FALSE;
LCPID_0.out_mode := LCPID_OUT_MODE_AUTO;


LCPWM_0.max_value   := 32767;
LCPWM_0.min_value   := 0;
LCPWM_0.t_min_pulse := 10;
LCPWM_0.t_period    := 100;

LCPIDpara_1.enable := 0;
LCPIDpara_1.enter := 0;
```

```
        //Settings
        start := 0;
        LCPIDpara_1.enable := 1;
        LCPIDpara_1.enter := 1;
        LCPIDpara_1.Kp := 175.7461;
        LCPIDpara_1.Tn := 423450;
        LCPIDpara_1.Tv := 105862.5;
        LCPIDpara_1.Tf := 10586.25;
        LCPIDpara_1.calc_mode := 1;
        LCPIDpara_1.d_mode := 2;
        LCPIDpara_1.fbk_mode := 1;
        LCPIDpara_1.Kw := 1;
        LCPIDpara_1.Y_max := 32763;
        LCPIDpara_1.Y_min := -32763;
        LCPIDpara_1();
        LCPIDpara_1.enter := 0;
        LCPID_0.ident    := LCPIDpara_1.ident;
                (* ident of PIDTune -> provides parameters (Kp, Tn, Tv, ...) *)
        setValue := gTemp[0];
        calValue := 3000;
        calValuer := INT_TO_REAL(calValue)/10;

        oscOptions.osc_minAmplitude := 50;
        ControlSet := 0;
END_PROGRAM


PROGRAM _CYCLIC

        IF MFCsetnew THEN               //adopts MFC set values from VNC-viewer
                FOR i := 0 TO 2 BY 1 DO
                MFCsetL[i] := MFCsetVNC[i];
                END_FOR;
                MFCsetnew := 0;
        END_IF;

        CASE GasSet OF                  //settings for gas flow
                0:      //Gas off
                        MFCset[0] := 0;
                        MFCset[1] := 0;
                        MFCset[2] := 0;


                1:      //set constant value
```

```
                    MFCset[0] := REAL_TO_INT((MFCsetL[0]+MFCzero[0]) / gasCoef[gasChoice[0]]
/ MFCCoef[MFCChoice[0]]*3277);
                    MFCset[1] := REAL_TO_INT((MFCsetL[1]+MFCzero[1]) / gasCoef[gasChoice[1]]
/ MFCCoef[MFCChoice[1]]*3277);
                    MFCset[2] := REAL_TO_INT((MFCsetL[2]+MFCzero[2]) / gasCoef[gasChoice[2]]
/ MFCCoef[MFCChoice[2]]*3277);


        2:      //initialize curve

                FOR i := 0 TO ZeilenPos-1 BY 1 DO
                        MFC1Table[i].x := x[i];
                        MFC1Table[i].y := MFC1[i];
                        MFC2Table[i].x := x[i];
                        MFC2Table[i].y := MFC2[i];
                        MFC3Table[i].x := x[i];
                        MFC3Table[i].y := MFC3[i];
                END_FOR;

                charCurveLCCurveByPoints[1].NoOfPoints := ZeilenPos;
                charCurveLCCurveByPoints[2].NoOfPoints := ZeilenPos;
                charCurveLCCurveByPoints[3].NoOfPoints := ZeilenPos;
                charCurveLCCurveByPoints[1].ptr_table := ADR(MFC1Table);
                charCurveLCCurveByPoints[2].ptr_table := ADR(MFC2Table);
                charCurveLCCurveByPoints[3].ptr_table := ADR(MFC3Table);
                GasSet := 3;
                HeatSet := 3;

        3:      //execute curve

                refLCCounter();
                charCurveLCCurveByPoints[1].x := UDINT_TO_INT(refLCCounter.seccnt);
                charCurveLCCurveByPoints[2].x := UDINT_TO_INT(refLCCounter.seccnt);
                charCurveLCCurveByPoints[3].x := UDINT_TO_INT(refLCCounter.seccnt);
                charCurveLCCurveByPoints[1]();
                charCurveLCCurveByPoints[2]();
                charCurveLCCurveByPoints[3]();
                MFCset[0] :=
REAL_TO_INT((INT_TO_REAL(charCurveLCCurveByPoints[1].y)+MFCzero[0]*100) /
gasCoef[gasChoice[0]] / MFCCoef[MFCChoice[0]]*32.77);
                MFCset[1] :=
REAL_TO_INT((INT_TO_REAL(charCurveLCCurveByPoints[2].y)+MFCzero[1]*100) /
gasCoef[gasChoice[1]] / MFCCoef[MFCChoice[1]]*32.77);
                MFCset[2] :=
REAL_TO_INT((INT_TO_REAL(charCurveLCCurveByPoints[3].y)+MFCzero[2]*100) /
gasCoef[gasChoice[2]] / MFCCoef[MFCChoice[2]]*32.77);
```

```
END_CASE;

CASE HeatSet OF                    //settings for heating
        0:
                start := 0;

        1:      //Tune

                tuningRequest := 1;
                start := 1;
                LCPID_0.ident := LCPIDTune_0.ident;
                LCPIDTune_0.enable   := 1;
                LCPIDTune_0.okToStep := LCPIDTune_0.rdyToStep;
                LCPIDTune_0.request  := tuningRequest;
                LCPIDTune_0.basetime := baseLCCounter.mscnt;
                LCPIDTune_0();                  (* LCPIDTune function block call *)
                IF LCPIDTune_0.state = LCPID_TUNE_STATE_FINISHED THEN
                        tuningRequest := LCPID_TUNE_REQU_OFF;
                        HeatSet := 0;
                        start := 0;
                END_IF;

                calValue := REAL_TO_INT(calValuer*10);
                setValue := calValue;

        2:      //heat to constant value

                start := 1;
                setValue := REAL_TO_INT(boxValueR*10);
                LCPIDpara_1();
                LCPID_0.ident := LCPIDpara_1.ident;

        3:      //initialize curve
                FOR i := 0 TO ZeilenPos-1 BY 1 DO
                        yTable[i].x := x[i];
                        yTable[i].y := y[i];
                END_FOR;

                charCurveLCCurveByPoints[0].NoOfPoints := ZeilenPos;
                charCurveLCCurveByPoints[0].ptr_table := ADR(yTable);
                HeatSet := 4;

        4:      //execute curve

                start := 1;
```

```
                refLCCounter();
                charCurveLCCurveByPoints[0].x := UDINT_TO_INT(refLCCounter.seccnt);
                charCurveLCCurveByPoints[0]();
                setValue := charCurveLCCurveByPoints[0].y * 10;
                LCPIDpara_1();
                LCPID_0.ident := LCPIDpara_1.ident;

END_CASE;

IF changePara THEN                          //adopts PID parameters from VNC-viewer

        LCPIDpara_1.enter := TRUE;
        changePara := 0;


        LCPIDpara_1.Tf := LCPIDpara_1.Tv/10;
        LCPIDpara_1();

END_IF;

////////////////////////////////////////////////////


IF bMFCzero THEN                            //sets measured MFC flow to zero
        IF MFCsetL[0] = 0 AND MFCsetL[1] = 0 AND MFCsetL[2] = 0 THEN
        FOR i := 0 TO 2 BY 1 DO
                MFCzero[i] := vMFCis[i] * gasCoef[gasChoice[i]] * MFCCoef[MFCChoice[i]];
                END_FOR;
        END_IF;

        bMFCzero := 0;
END_IF;

//PID controller


                baseLCCounter();
                setValue1 := (setValue)/10;
LCPID_0.enable   := start;
actValue := gTemp[k-1];
LCPID_0.W       := setValue;
LCPID_0.X       := actValue;
LCPID_0.basetime := baseLCCounter.mscnt;
LCPID_0();                          (* LCPID function block call *)
manipulatedVar := LCPID_0.Y;
```

```
        IF bAverage AND j < 1200 THEN                   //initiates averaging of the power over 60s
                sumPower := sumPower + manipulatedVar;
                j := j + 1;
        END_IF;

        IF j = 1200 AND HeatSet <> 0 THEN                       //sets average power as set power
                avgPower := DINT_TO_INT(sumPower/1200);
                manipulatedVar := avgPower;
                bHolding := 1;
        ELSE
                bHolding := 0;
        END_IF;

        IF bAverage = 0 THEN                            //makes set power PID controlled again
                j := 0;
                sumPower := 0;
        END_IF;



        LCPWM_0.enable     := start;
        LCPWM_0.x       := manipulatedVar;
        LCPWM_0.basetime := baseLCCounter.mscnt;
        LCPWM_0();              (* LCPWM function block call *)
        pulse := LCPWM_0.pulse;
        IF pulse = TRUE THEN                            //PWM-signal for the relay
                gVoltage1 := 16387;
        ELSE
                gVoltage1 := 0;
        END_IF;

        IF manipulatedVar < 0 THEN      //air cooling if the output from the PID controller is below 0
                MFCsetL[3] := -INT_TO_REAL(manipulatedVar)/32767*20;
                MFCset[3] := REAL_TO_INT((MFCsetL[3]+MFCzero[3]) / gasCoef[gasChoice[3]] /
MFCCoef[MFCChoice[3]]*3277);
        ELSE
                MFCsetL[3] := 0;
                MFCset[3] := 0;
        END_IF;


        Power := REAL_TO_INT(INT_TO_REAL(LCPID_0.Y)/32767*100);


        IF LCPIDTune_0.state = 50 THEN          //tuning ends at 50, autotune ends and autotune is
turned off
```

```
            tuningRequest := 0;
        END_IF;


        FOR i := 0 TO 3 BY 1 DO                         //sets the control voltage for the MFCs
        vMFCis[i] := INT_TO_REAL(MFCis[i]) /3277;
        MFCLis[i] := vMFCis[i] * gasCoef[gasChoice[i]] * MFCCoef[MFCChoice[i]]-MFCzero[i];
        END_FOR;
        MFCLis[3] := vMFCis[3] * gasCoef[gasChoice[3]] * MFCCoef[2]-MFCzero[3];

        FOR i := 0 TO 5 BY 1 DO                                                 //checks
whether any thermocouple shows more than 1050°C, in which case it turns off heating. Disconnected
thermocouples (showing 3276.7°C) are ignored
            IF gTempR[i] > 1050 AND gTempR[i] <> 3276.7 THEN
            gVoltage1 := 0;
            END_IF;
        END_FOR;



END_PROGRAM

PROGRAM _EXIT




END_PROGRAM
```

## FileHandling (reads files and saves measurements)

```
PROGRAM _INIT

        Select := 0;

        IF Select = 0 THEN
            strcpy( ADR(Handling.Data.Device), ADR('LOCAL_DEVICE') );
            strcpy( ADR(Handling.Data.Parameter), ADR('"/DEVICE=C:') );
        ELSIF Select = 1 THEN
            strcpy( ADR(Handling.Data.Device), ADR('STICK') );
            strcpy( ADR(Handling.Data.Parameter), ADR('"/DEVICE=IF4.ST1') );
        END_IF

        strcpy( ADR(Handling.Data.NewFileName), ADR('Results.txt') );
        Handling.Data.Step := 0;
```

```
        gMeasure := 0;
        gRead := 0;
        gCreate := 0;

        byErrorLevel := 0;
        dwCounter      := 0;
        bOK := FALSE;
        strDevice      := 'HARDDISK';
        MFCList[0] := 'A (10 slm)';
        MFCList[1] := 'B (10 slm)';
        MFCList[2] := 'C (20 slm)';
        MFCList[3] := 'D (20 slm)';
        MFCList[4] := 'E (1 slm)';
        MFCList[5] := 'F (1 slm)';
        gasList[0] := 'Air';
        gasList[1] := 'N2';
        gasList[2] := 'O2';
        gasList[3] := 'CO2';
        gasCoef[0] := 1;
        gasCoef[1] := 1;
        gasCoef[2] := 1;
        gasCoef[3] := 0.7;
        MFCCoef[0] := 2;                    // L/min/V
        MFCCoef[1] := 1.75;
        MFCCoef[2] := 3.96;
        MFCCoef[3] := 3.73;
        MFCCoef[4] := 0.218;
        MFCCoef[5] := 0.25;

        DTGetTime_0.enable := 1;
        DTGetTime_0();
        startDT := DTGetTime_0.DT1;
        DTSGetTime_0.enable := 1;
        DTSGetTime_0.pDTStructure := ADR(DTStructure_0);
        DTSGetTime_0();
        sDatum := INT_TO_STRING(DTStructure_0.year);
        strcat(ADR(sDatum),ADR('-'));
        sMonth := INT_TO_STRING(DTStructure_0.month);
        strcat(ADR(sDatum),ADR(sMonth));
        strcat(ADR(sDatum),ADR('-'));
        sDay := INT_TO_STRING(DTStructure_0.day);
        strcat(ADR(sDatum),ADR(sDay));
        strcpy( ADR(Handling.Data.FileName), ADR(sDatum) );
        strcat(ADR(Handling.Data.FileName),ADR('.txt'));
        Handling.Data.NewFileName := Handling.Data.FileName;
END_PROGRAM
PROGRAM _CYCLIC

        Zoom := USINT_TO_REAL(iZoom);
```

```
Scroll := USINT_TO_REAL(iScroll)/100;

locZahl3 := gTest;
lRead := gRead;

        gStep := Handling.Data.Step;

CASE Handling.Data.Step OF

    0:  (* link (create) a file device *)
            Handling.Functionblock.DevLink_0.enable := 1;
            Handling.Functionblock.DevLink_0.pDevice := ADR(Handling.Data.Device);
            Handling.Functionblock.DevLink_0.pParam := ADR(Handling.Data.Parameter);

            IF Handling.Functionblock.DevLink_0.status = 0 THEN
                Handling.Data.Step := 1;
            ELSIF Handling.Functionblock.DevLink_0.status = ERR_FUB_BUSY THEN
            ELSIF Handling.Functionblock.DevLink_0.status = fiERR_SYSTEM THEN
                Error := FileIoGetSysError();
                Handling.Data.Step := 255;
            ELSE
                Handling.Data.Step := 255;
            END_IF

    1:  (* Wait and command step *)
            i := 0;
            bMeasuring := 0;
            Handling.Data.NewFileName := Handling.Data.FileName;
            IF gCreate = 1 THEN
                (* Create a new file with the name from the variable
"Handling.Data.FileName" *)
                Handling.Data.Step := 10;  (* next Step*)

            ELSIF gMeasure = 1 THEN
                (* writes the data from variable "Handling.Data.WriteData"
                  in the file with the name from the variable
"Handling.Data.FileName" *)
                Handling.Data.Step := 10;  (* next Step*)

            ELSIF gRead = 1 THEN
                (* read data to variable "Handling.Data.ReadData" from the file with
the name of the variable "Handling.Data.FileName" *)
                Handling.Data.Step := 30;  (* next Step*)

            ELSIF Handling.Command.bReadExFile = 1 THEN
                (* read data to variable "Handling.Data.ReadData" from the file with
the name of the variable "Handling.Data.FileName" *)
                Handling.Data.Step := 40;  (* next Step*)
```

```
                ELSIF Handling.Command.bCopyFile = 1 THEN
                        (* creates a copy with the name from the variable
"Handling.Data.NewFileName"
                          from the file with the name from the variable
"Handling.Data.FileName" *)
                        Handling.Data.Step := 50;  (* next Step*)

                ELSIF Handling.Command.bRenameFile = 1 THEN
                        (* renames the the file with the name from the variable
"Handling.Data.FileName"
                          to the name from the variable "Handling.Data.NewFileName" *)
                        Handling.Data.Step := 60;  (* next Step*)


                ELSIF Handling.Command.bDeleteFile = 1 THEN
                        (* deletes the file with the name from the variable
"Handling.Data.FileName" *)
                        Handling.Data.Step := 70;  (* next Step*)

                ELSIF ReadDir = 1 THEN
                        (* deletes the file with the name from the variable
"Handling.Data.FileName" *)
                        Handling.Data.Step := 81;  (* next Step*)

                END_IF

        2:
                DevUnlink_0.enable := 1;
                DevUnlink_0.handle := Handling.Functionblock.DevLink_0.handle;
                DevUnlink_0();
                IF DevUnlink_0.status <> 65535 THEN
                        Handling.Data.Step := 0;
                END_IF;



    (*********************************************************************
*********************)
        10:  (* create a new File with the selected name  *)
                Handling.Functionblock.FileCreate_0.enable := 1;
                Handling.Functionblock.FileCreate_0.pDevice := ADR(Handling.Data.Device);
(* name of the linked device *)
                Handling.Functionblock.FileCreate_0.pFile := ADR(Handling.Data.FileName);
(* name of the file *)
                Handling.Functionblock.FileCreate_0;  (* call the function*)

                IF Handling.Functionblock.FileCreate_0.status = 0 THEN  (* FileCreate
successful *)
                        Handling.Data.Step := 11;  (* next Step*)
```

```
                    ELSIF Handling.Functionblock.FileCreate_0.status = ERR_FUB_BUSY THEN  (*
FileCreate not finished -> redo *)
                        (* Busy *)
                    ELSE  (* Goto Error Step *)
                        //Handling.Data.Step := 21;
                    END_IF

                    IF Handling.Functionblock.FileCreate_0.status = 0 THEN  (* FileCopy
successful *)
                        Handling.Data.Step := 11;  (* next Step*)
                        gCreate := 0;  (* clear command *)
                    ELSIF Handling.Functionblock.FileCreate_0.status = 20705 OR
Handling.Functionblock.FileCreate_0.status = 20700 THEN  (* FileCopy not finished -> redo *)
                        i := i +1;
                        strcpy(ADR(Handling.Data.FileName), ADR(sDatum));
                        strcat(ADR(Handling.Data.FileName),ADR('-'));
                        si := UDINT_TO_STRING(i);
                        strcat(ADR(Handling.Data.FileName),ADR(si));
                        strcat(ADR(Handling.Data.FileName),ADR('.txt'));
                    END_IF

                    (*locInfo.enable := TRUE;
                    locInfo.pDevice := ADR(Handling.Data.Device);
                    locInfo.pName := ADR(Handling.Data.FileName);
                    locInfo.pInfo := ADR(locInfo2);
                    REPEAT
                        locInfo;
                        UNTIL
                        locInfo.status <> 65535
                    END_REPEAT;*)

                11:     (* close file, because of limited number of available file handles on the
system *)
                    Handling.Functionblock.FileClose_0.enable := 1;
                    Handling.Functionblock.FileClose_0.ident :=
Handling.Functionblock.FileCreate_0.ident; (* ident for FileCreate-functionblock*)
                    Handling.Functionblock.FileClose_0;  (* call the function*)

                    IF Handling.Functionblock.FileClose_0.status = 0 THEN  (* FileClose successful
*)
                        Handling.Data.Step := 20;  (* next Step*)
                        gCreate := 0;  (* clear command *)
                    ELSIF Handling.Functionblock.FileClose_0.status = ERR_FUB_BUSY THEN  (*
FileClose not finished -> redo *)
                        (* Busy *)
                    ELSE  (* Goto Error Step *)
                        Handling.Data.Step := 20;
                    END_IF
```

```
        (*************************************************************************
********************)
                20:     (* open file for write access *)
                        Handling.Functionblock.FileOpen_0.enable := 1;
                        Handling.Functionblock.FileOpen_0.pDevice := ADR(Handling.Data.Device);
(* name of the linked device *)
                        Handling.Functionblock.FileOpen_0.pFile :=  ADR(Handling.Data.FileName);
(* name of the file *)
                        Handling.Functionblock.FileOpen_0.mode := fiWRITE_ONLY; (* write access
*)
                        REPEAT
                                Handling.Functionblock.FileOpen_0;  (* call the function*)
                                UNTIL
                                Handling.Functionblock.FileOpen_0.status <> 65535
                        END_REPEAT

                        (*IF Handling.Functionblock.FileOpen_0.status = 20708 THEN
                                Handling.Data.Step := 10;
                        END_IF*)

                        locInfo.enable := TRUE;
                        locInfo.pDevice := ADR(Handling.Data.Device);
                        locInfo.pName := ADR(Handling.Data.FileName);
                        locInfo.pInfo := ADR(locInfo2);
                        REPEAT
                                locInfo;
                                UNTIL
                                locInfo.status <> 65535
                        END_REPEAT;


                        IF Handling.Functionblock.FileOpen_0.status = 0 THEN  (* FileOpen successful
*)
                                IF locInfo.status = 0 THEN
                                        Handling.Data.Step := 23;  (* next Step*)
                                        END_IF;
                        ELSIF Handling.Functionblock.FileOpen_0.status = ERR_FUB_BUSY THEN  (*
FileOpen not finished -> redo *)
                                (* Busy *)
                        ELSE  (* Goto Error Step *)
                                //Handling.Data.Step := 255;
                        END_IF


                21: (* write data into file *)
                        IF i MOD 10 = 0 THEN            //da diese Routine alle 100ms ausgeführt
wird, wird diese Funktion nur jedes zehnte Mal durchgeführt um insgesamt jede Sekunde einen
Messwert aufzuschreiben
```

```
Handling.Data.WriteData := '';
//locTemp4 := INT_TO_STRING(gTemp[0]);
//status_gettime := clock_ms();//RTC_gettime(ADR(rtc_gettime));
DTGetTime_0.enable := 1;
DTGetTime_0();
ascDT(DTGetTime_0.DT1,ADR(lTime),25);
strcat(ADR(lTime), ADR(lSp));
(*lMinute := USINT_TO_STRING(rtc_gettime.minute);
lSecond := USINT_TO_STRING(rtc_gettime.second);
lTime := USINT_TO_STRING(rtc_gettime.hour);
strcat(ADR(lTime), ADR(lSep));
strcat(ADR(lTime),ADR(lMinute));
strcat(ADR(lTime), ADR(lSep));
strcat(ADR(lTime),ADR(lSecond));
strcat(ADR(lTime), ADR(lSp));*)
locString1 := '$R$N';
locTemp4 := REAL_TO_STRING(gTempR[0]);
strcat(ADR(lTime), ADR(locTemp4));
strcat( ADR(lTime), ADR(lSp) );


locTemp4 := REAL_TO_STRING(gTempR[1]);
strcat(ADR(lTime), ADR(locTemp4));
strcat(ADR(lTime), ADR(lSp));

locTemp4 := REAL_TO_STRING(gTempR[2]);
strcat(ADR(lTime), ADR(locTemp4));
strcat(ADR(lTime), ADR(lSp));

locTemp4 := REAL_TO_STRING(gTempR[3]);
strcat(ADR(lTime), ADR(locTemp4));
strcat(ADR(lTime), ADR(lSp));

locTemp4 := REAL_TO_STRING(gTempR[4]);
strcat(ADR(lTime), ADR(locTemp4));
strcat(ADR(lTime), ADR(lSp));

locTemp4 := REAL_TO_STRING(gTempR[5]);
strcat(ADR(lTime), ADR(locTemp4));
strcat(ADR(lTime), ADR(lSp));

locTemp4 := REAL_TO_STRING(MFCsetL[0]);
strcat(ADR(lTime), ADR(locTemp4));
strcat(ADR(lTime), ADR(lSp));

locTemp4 := REAL_TO_STRING(MFCsetL[1]);
strcat(ADR(lTime), ADR(locTemp4));
strcat(ADR(lTime), ADR(lSp));
```

```
                        //strcat( ADR(lTime), ADR(locString1) );
                        strcpy( ADR(Handling.Data.WriteData), ADR(lTime) );
                        //strcat( ADR(Handling.Data.WriteData), ADR(MFCList[0]) );
                        //strcat( ADR(Handling.Data.WriteData), ADR(MFCList[1]) );
                        //strcat( ADR(Handling.Data.WriteData), ADR(MFCList[2]) );
                        //strcat( ADR(Handling.Data.WriteData), ADR(MFCList[3]) );
                        Handling.Functionblock.FileWrite_0.enable := 1;
                        Handling.Functionblock.FileWrite_0.ident :=
Handling.Functionblock.FileOpen_0.ident;
                        Handling.Functionblock.FileWrite_0.offset := locInfo2.size;
                        Handling.Functionblock.FileWrite_0.pSrc := ADR(Handling.Data.WriteData);
                                Handling.Functionblock.FileWrite_0.len := strlen(
ADR(Handling.Data.WriteData) );

                        END_IF;
                        IF i MOD 10 = 5 THEN                    //Da die Daten zu lang sind um sie auf
einmal zu schreiben, wird hier nun mit 500ms Versetzung der zweite Teil geschrieben
                                lTime := '';
                                Handling.Data.WriteData := '';
                                //locTemp4 := INT_TO_STRING(gTemp[0]);
                                //status_gettime := clock_ms();//RTC_gettime(ADR(rtc_gettime));
                                //DTGetTime_0.enable := 1;
                                //DTGetTime_0();
                                //ascDT(DTGetTime_0.DT1,ADR(lTime),25);
                                //strcat(ADR(lTime), ADR(lSp));
                        (*lMinute := USINT_TO_STRING(rtc_gettime.minute);
                        lSecond := USINT_TO_STRING(rtc_gettime.second);
                        lTime := USINT_TO_STRING(rtc_gettime.hour);
                        strcat(ADR(lTime), ADR(lSep));
                        strcat(ADR(lTime),ADR(lMinute));
                        strcat(ADR(lTime), ADR(lSep));
                        strcat(ADR(lTime),ADR(lSecond));
                        strcat(ADR(lTime), ADR(lSp));*)
                                locString1 := '$R$N';
                                locTemp4 := REAL_TO_STRING(MFCsetL[2]);
                                strcpy(ADR(lTime), ADR(locTemp4));
                                strcat(ADR(lTime), ADR(lSp));
                        (*
                        locTemp4 := INT_TO_STRING(MFCset[3]);
                        strcat(ADR(lTime), ADR(locTemp4));
                        strcat(ADR(lTime), ADR(lSp));
                        *)
                                locTemp4 := REAL_TO_STRING(MFCLis[0]);
                                strcat(ADR(lTime), ADR(locTemp4));
                                strcat(ADR(lTime), ADR(lSp));

                                locTemp4 := REAL_TO_STRING(MFCLis[1]);
                                strcat(ADR(lTime), ADR(locTemp4));
```

```
                                strcat(ADR(lTime), ADR(lSp));

                                locTemp4 := REAL_TO_STRING(MFCLis[2]);
                                strcat(ADR(lTime), ADR(locTemp4));
                                strcat(ADR(lTime), ADR(lSp));

                                locTemp4 := REAL_TO_STRING(MFCLis[3]);
                                strcat(ADR(lTime), ADR(locTemp4));
                                strcat(ADR(lTime), ADR(lSp));

                                locTemp4 := REAL_TO_STRING(offgasR[0]);
                                strcat(ADR(lTime), ADR(locTemp4));
                                strcat(ADR(lTime), ADR(lSp));

                                locTemp4 := REAL_TO_STRING(setValue);
                                strcat(ADR(lTime), ADR(locTemp4));
                                strcat(ADR(lTime), ADR(lSp));

                                locTemp4 := REAL_TO_STRING(Power);
                                strcat(ADR(lTime), ADR(locTemp4));
                                strcat(ADR(lTime), ADR(lSp));

                                strcat( ADR(lTime), ADR(locString1) );
                                strcpy( ADR(Handling.Data.WriteData), ADR(lTime) );
                                Handling.Functionblock.FileWrite_0.enable := 1;
                                Handling.Functionblock.FileWrite_0.ident :=
Handling.Functionblock.FileOpen_0.ident;
                                Handling.Functionblock.FileWrite_0.offset := locInfo2.size;
                        //chooses the writing position so the data is attached tot he end oft he file instead of
overwriting something
                                Handling.Functionblock.FileWrite_0.pSrc :=
ADR(Handling.Data.WriteData);
                                Handling.Functionblock.FileWrite_0.len := strlen(
ADR(Handling.Data.WriteData) );

                        END_IF;

                                IF i MOD 5 = 0 OR Handling.Functionblock.FileWrite_0.status =
ERR_FUB_BUSY THEN
                                Handling.Functionblock.FileWrite_0;  (* call the function*)

                                IF Handling.Functionblock.FileWrite_0.status = 0 THEN  (* FileWrite
successful *)
                                        locInfo2.size := locInfo2.size + strlen(
ADR(Handling.Data.WriteData));                  //position advances by the written data length
                                END_IF;

                                END_IF;
```

```
                IF Handling.Functionblock.FileWrite_0.status = 0 THEN  (* FileWrite
successful *)

                        bMeasuring := 1;

                        IF gMeasure = FALSE THEN
                                Handling.Data.Step := 22;
                        END_IF;

                        IF Handling.Functionblock.FileWrite_0.status <> 0 AND
Handling.Functionblock.FileWrite_0.status <> 65535 THEN
                                bMeasuring := 0;
                        END_IF;

                ELSIF Handling.Functionblock.FileWrite_0.status = ERR_FUB_BUSY THEN  (*
FileWrite not finished -> redo *)
                        (* Busy *)
                END_IF

                i := i +1;

        22:     (* close file, because of limited number of available file handles on the
system *)
                bMeasuring := 0;
                Handling.Functionblock.FileClose_0.enable := 1;
                Handling.Functionblock.FileClose_0.ident :=
Handling.Functionblock.FileOpen_0.ident; (* ident for FileCreate-functionblock*)
                Handling.Functionblock.FileClose_0;  (* call the function*)

                IF Handling.Functionblock.FileClose_0.status = 0 THEN  (* FileClose successful
*)
                        Handling.Data.Step := 1;  (* next Step*)
                        Handling.Command.bWriteFile := 0;  (* clear command *)
                ELSIF Handling.Functionblock.FileClose_0.status = ERR_FUB_BUSY THEN  (*
FileClose not finished -> redo *)
                        (* Busy *)
                ELSE  (* Goto Error Step *)
                        Handling.Data.Step := 255;
                END_IF

        23:     //writes the first part oft he header, the entire thing does not fit into one
string

                Handling.Functionblock.FileWrite_0.enable := 1;
                Handling.Functionblock.FileWrite_0.ident :=
Handling.Functionblock.FileOpen_0.ident;
                Handling.Functionblock.FileWrite_0.offset := locInfo2.size;
                Handling.Data.WriteData :='Wochentag Monat Tag Zeit
Jahr;T1;T2;T3;T4;T5;T6;MFC1-Einstellung;MFC2-Einstellung;MFC3-Einstellung';
```

```
                    Handling.Functionblock.FileWrite_0.pSrc := ADR(Handling.Data.WriteData);
                    Handling.Functionblock.FileWrite_0.len := strlen(
ADR(Handling.Data.WriteData) );
                    Handling.Functionblock.FileWrite_0();
                    IF Handling.Functionblock.FileWrite_0.status = 0 THEN
                        REPEAT
                            locInfo;
                            UNTIL
                            locInfo.status <> 65535
                        END_REPEAT;
                        Handling.Data.Step := 24;
                    END_IF;

            24:     //writes the second part oft he header

                    Handling.Functionblock.FileWrite_0.enable := 1;
                    Handling.Functionblock.FileWrite_0.ident :=
Handling.Functionblock.FileOpen_0.ident;
                    Handling.Functionblock.FileWrite_0.offset := locInfo2.size;
                    Handling.Data.WriteData :=';MFC1-Durchfluss;MFC2-Durchfluss;MFC3-
Durchfluss;MFC4-Durchfluss(Ausgang);Sauerstoffkonzentration$R$N';
                    Handling.Functionblock.FileWrite_0.pSrc := ADR(Handling.Data.WriteData);
                    Handling.Functionblock.FileWrite_0.len := strlen(
ADR(Handling.Data.WriteData) );
                    Handling.Functionblock.FileWrite_0();
                    IF Handling.Functionblock.FileWrite_0.status = 0 THEN
                        REPEAT
                            locInfo;
                            UNTIL
                            locInfo.status <> 65535
                        END_REPEAT;
                        Handling.Data.Step := 21;
                    END_IF;
            i:=0;

        (*************************************************************************
*******************)
            30:     (* open file for read access *)
                    LesePos := 0;
                    ZeilenPos := 0;
                    strcpy(ADR(sSettingsFile),ADR(sPath));

                    IF sPath <> '' THEN
                        strcat(ADR(sSettingsFile),ADR('/'));
                        //Handling.Data.Step := 34;
                    END_IF;

                    strcat(ADR(sSettingsFile),ADR(sFileName[iFileChoice]));
                    //sSettingsFile := sFileName[iFileChoice];
```

```
                        Handling.Functionblock.FileOpen_0.enable := 1;
                        Handling.Functionblock.FileOpen_0.pDevice := ADR(gDevice);  (* name of the
linked device *)
                        Handling.Functionblock.FileOpen_0.pFile :=
ADR(sSettingsFile);//'Settings.txt');  (* name of the file *)
                        Handling.Functionblock.FileOpen_0.mode := fiREAD_ONLY;  (* read access *)
                        Handling.Functionblock.FileOpen_0;  (* call the function*)

                        IF Handling.Functionblock.FileOpen_0.status = 0 THEN  (* FileOpen successful
*)
                                Handling.Data.Step := 31;  (* next Step*)
                        ELSIF Handling.Functionblock.FileOpen_0.status = ERR_FUB_BUSY THEN  (*
FileOpen not finished -> redo *)
                                (* Busy *)
                        ELSE  (* Goto Error Step *)
                                //Handling.Data.Step := 255;
                        END_IF

                31: (* read data from file *)
                        Handling.Functionblock.FileRead_0.enable := 1;
                        Handling.Functionblock.FileRead_0.ident :=
Handling.Functionblock.FileOpen_0.ident;  (* ident of the previous "FileOpen" *)
                        Handling.Functionblock.FileRead_0.offset := LesePos;  (* start at the
beginning of the file *)
                        Handling.Functionblock.FileRead_0.pDest := ADR(sDatenzeile);  (* formerly
Handling.Data.ReadData adress of the destination of reaaded datas *)
                        Handling.Functionblock.FileRead_0.len := SIZEOF(sDatenzeile);  (* formerly
Handling.Data.ReadData lenght of data, which should be readed *)
                        Handling.Functionblock.FileRead_0;  (* call the function*)

                        IF Handling.Functionblock.FileRead_0.status = 0 THEN

                        AbsatzPos := FIND(sDatenzeile,'$R$N');
                        LesePos := LesePos+AbsatzPos+1;
                        IF LEFT(sDatenzeile,4) = 'EXIT' THEN
                                Handling.Data.Step := 33;
                        ELSIF AbsatzPos = 0 THEN
                                        Handling.Data.Step := 33;
                        ELSIF LEFT(sDatenzeile,1) = '/' THEN
                        ELSIF LEFT(sDatenzeile,3) = 'REP' THEN
                                RepZeilenZahl := STRING_TO_INT(MID(sDatenzeile,1,4));
                                RepZyklenZahl := STRING_TO_INT(MID(sDatenzeile,1,6));
                                FOR RepZyklus := 0 TO RepZyklenZahl-1 BY 1 DO
                                        FOR RepZeile := 0 TO RepZeilenZahl-1 BY 1 DO
                                                x[ZeilenPos+RepZyklus*RepZeilenZahl+RepZeile] :=
x[ZeilenPos+RepZyklus*RepZeilenZahl+RepZeile-1] + x[ZeilenPos-RepZeilenZahl+RepZeile] -
x[ZeilenPos-RepZeilenZahl+RepZeile-1];
y[ZeilenPos+RepZyklus*RepZeilenZahl+RepZeile] := y[ZeilenPos-RepZeilenZahl+RepZeile];
MFC1[ZeilenPos+RepZyklus*RepZeilenZahl+RepZeile] := MFC1[ZeilenPos-RepZeilenZahl+RepZeile];
```

```
MFC2[ZeilenPos+RepZyklus*RepZeilenZahl+RepZeile] := MFC2[ZeilenPos-RepZeilenZahl+RepZeile];
MFC3[ZeilenPos+RepZyklus*RepZeilenZahl+RepZeile] := MFC3[ZeilenPos-RepZeilenZahl+RepZeile];
MFC4[ZeilenPos+RepZyklus*RepZeilenZahl+RepZeile] := MFC4[ZeilenPos-RepZeilenZahl+RepZeile];
                                        END_FOR;
                                END_FOR;
                                ZeilenPos := ZeilenPos+RepZyklenZahl*RepZeilenZahl;
                        ELSE
                                        TabPos := FIND(sDatenzeile,'$t');

                                        IF LEFT(sDatenzeile,1) = 'R' OR LEFT(sDatenzeile,1) = 'r' THEN
                                                bRate := 1;
                                                Rate := STRING_TO_REAL(MID(sDatenzeile,TabPos -
2,2));
                                        ELSIF LEFT(sDatenzeile,1) = '+' THEN
                                                TimeIncrement :=
STRING_TO_INT(MID(sDatenzeile,TabPos - 2,2));
                                                x[ZeilenPos] := x[ZeilenPos -1] + TimeIncrement;
                                        ELSE
                                        x[ZeilenPos] := STRING_TO_INT(LEFT(sDatenzeile,TabPos-1));
                                        END_IF;

                                        sDatenzeile := DELETE(sDatenzeile,TabPos,1);


                                TabPos := FIND(sDatenzeile,'$t');
                                y[ZeilenPos] := STRING_TO_INT(LEFT(sDatenzeile,TabPos-1));
                                sDatenzeile := DELETE(sDatenzeile,TabPos,1);

                                TabPos := FIND(sDatenzeile,'$t');
        MFC1[ZeilenPos] := REAL_TO_INT(STRING_TO_REAL(LEFT(sDatenzeile,TabPos-1))*100);
                                sDatenzeile := DELETE(sDatenzeile,TabPos,1);

                                TabPos := FIND(sDatenzeile,'$t');
        MFC2[ZeilenPos] := REAL_TO_INT(STRING_TO_REAL(LEFT(sDatenzeile,TabPos-1))*100);
                                sDatenzeile := DELETE(sDatenzeile,TabPos,1);

                                TabPos := FIND(sDatenzeile,'$t');
MFC3[ZeilenPos] := REAL_TO_INT(STRING_TO_REAL(LEFT(sDatenzeile,TabPos-1))*100);
                                sDatenzeile := DELETE(sDatenzeile,TabPos,1);

                                TabPos := FIND(sDatenzeile,'$r$n');
MFC4[ZeilenPos] := REAL_TO_INT(STRING_TO_REAL(LEFT(sDatenzeile,TabPos-1))*100);
                                        //sDatenzeile := DELETE(sDatenzeile,TabPos,1);
                                IF bRate = 1 THEN
x[ZeilenPos] := x[ZeilenPos -1] + ABS(REAL_TO_INT(INT_TO_REAL(y[ZeilenPos] - y[ZeilenPos-1])*60 /
Rate));
                                        bRate := 0;
                                END_IF;
```

```
                    ZeilenPos := ZeilenPos+1;
                END_IF;
                END_IF;
                EndTime := x[ZeilenPos -1];
                //IF Handling.Command.bWriteFile = 0 THEN
                //      Handling.Data.Step := 32;
                //END_IF;


            IF Handling.Functionblock.FileRead_0.status = 0 THEN  (* FileRead successful *)
                    sDatenzeile := '';
ELSIF Handling.Functionblock.FileRead_0.status = ERR_FUB_BUSY THEN  (* FileRead not finished ->
redo *)
                    (* Busy *)
                ELSE  (* Goto Error Step *)
                    //Handling.Data.Step := 255;
                END_IF

32:     (* close file, because of limited number of available file handles on the system *)
                bHasRead := 1;
                Handling.Functionblock.FileClose_0.enable := 1;
Handling.Functionblock.FileClose_0.ident := Handling.Functionblock.FileOpen_0.ident; (* ident for
FileCreate-functionblock*)
                Handling.Functionblock.FileClose_0;  (* call the function*)

        IF Handling.Functionblock.FileClose_0.status = 0 THEN  (* FileClose successful *)
                    Handling.Data.Step := 1;  (* next Step*)
                    gRead := 0;  (* clear command *)
ELSIF Handling.Functionblock.FileClose_0.status = ERR_FUB_BUSY THEN  (* FileClose not finished ->
redo *)
                    (* Busy *)
                ELSE  (* Goto Error Step *)
                    Handling.Data.Step := 255;
                END_IF

        33:             //Interpolation of the T/MFC-program fort he GUI
                SampleNumber := 676;
                SampleTime := INT_TO_REAL(EndTime) / SampleNumber;
                SampleTimeMS := REAL_TO_DINT(SampleTime * 1000);
                ZeilenPos1 := 0;
                SampleCount := 0;
        WHILE (x[ZeilenPos1] <> 0 OR ZeilenPos1 <= 1) AND SampleCount < SampleNumber DO
                IF x[ZeilenPos1 + 1] > REAL_TO_INT(SampleCount * SampleTime) THEN
SampleValue[SampleCount] := y[ZeilenPos1] + (y[ZeilenPos1 + 1]-y[ZeilenPos1]) *
(REAL_TO_INT(SampleCount * SampleTime) - x[ZeilenPos1])/(x[ZeilenPos1 + 1]-x[ZeilenPos1]);

SampleValue1[SampleCount] := MFC1[ZeilenPos1] + (MFC1[ZeilenPos1 + 1]-MFC1[ZeilenPos1]) *
(REAL_TO_INT(SampleCount * SampleTime) - x[ZeilenPos1])/(x[ZeilenPos1 + 1]-x[ZeilenPos1]);
```

```
SampleValue2[SampleCount] := MFC2[ZeilenPos1] + (MFC2[ZeilenPos1 + 1]-MFC2[ZeilenPos1]) *
(REAL_TO_INT(SampleCount * SampleTime) - x[ZeilenPos1])/(x[ZeilenPos1 + 1]-x[ZeilenPos1]);

SampleValue3[SampleCount] := MFC3[ZeilenPos1] + (MFC3[ZeilenPos1 + 1]-MFC3[ZeilenPos1]) *
(REAL_TO_INT(SampleCount * SampleTime) - x[ZeilenPos1])/(x[ZeilenPos1 + 1]-x[ZeilenPos1]);

SampleValue4[SampleCount] := MFC4[ZeilenPos1] + (MFC4[ZeilenPos1 + 1]-MFC4[ZeilenPos1]) *
(REAL_TO_INT(SampleCount * SampleTime) - x[ZeilenPos1])/(x[ZeilenPos1 + 1]-x[ZeilenPos1]);

SampleValue1R[SampleCount] := INT_TO_REAL(SampleValue1[SampleCount])/100;

SampleValue2R[SampleCount] := INT_TO_REAL(SampleValue2[SampleCount])/100;

SampleValue3R[SampleCount] := INT_TO_REAL(SampleValue3[SampleCount])/100;

SampleValue4R[SampleCount] := INT_TO_REAL(SampleValue4[SampleCount])/100;
                                        SampleCount := SampleCount + 1;
                                ELSE
                                        ZeilenPos1 := ZeilenPos1 + 1;
                                END_IF;
                        END_WHILE;


                        Handling.Data.Step := 32;

                34:
                        IF DirOpen_0.ident <> 0 THEN
                                DirClose_0.enable := 1;
                                DirClose_0.ident := DirOpen_0.ident;
                                DirClose_0();
                        ELSE
                                Handling.Data.Step := 35;
                        END_IF;
                        IF DirClose_0.status = 0 THEN
                                Handling.Data.Step := 35;
                        END_IF;

                35:

                        DirOpen_0.enable := 1;
                        DirOpen_0.pDevice := ADR(gDevice);
                        DirOpen_0.pName := ADR(sPath);
                        DirOpen_0();

                        IF DirOpen_0.status = 0 THEN
                                Handling.Data.Step := 30;
                                sPathOpened := sPath;
                        END_IF;
```

```
50: (* copy a file *)
        Handling.Command.bCopyFile := 0;
        Handling.Functionblock.FileCopy_0.enable := 1;
        Handling.Functionblock.FileCopy_0.option := fiRECURSIVE;
        Handling.Functionblock.FileCopy_0.pSrcDev := ADR(Handling.Data.Device);
(* name of a linked device *)
        Handling.Functionblock.FileCopy_0.pSrc := ADR(Handling.Data.FileName);  (*
name of the source file *)
        Handling.Functionblock.FileCopy_0.pDestDev := ADR(gDevice);  (* name of a
linked device *)
        strcpy(ADR(NewFilePath), ADR('Results\'));
        strcat(ADR(NewFilePath), ADR(Handling.Data.NewFileName));

        Handling.Functionblock.FileCopy_0.pDest := ADR(NewFilePath);  (* name of
the new file*)
        Handling.Functionblock.FileCopy_0;  (* call the function*)

        IF Handling.Functionblock.FileCopy_0.status = 0 THEN  (* FileCopy successful
*)
            Handling.Data.Step := 1;  (* next Step*)
            bCopied := 1;
        ELSIF Handling.Functionblock.FileCopy_0.status = ERR_FUB_BUSY THEN  (*
FileCopy not finished -> redo *)
            (* Busy *)
        ELSIF Handling.Functionblock.FileCopy_0.status = 20705 (*OR
Handling.Functionblock.FileCopy_0.status = 20700*) THEN  (* FileCopy not finished -> redo *)
            i := i +1;
            strcpy(ADR(Handling.Data.NewFileName), ADR(sDatum));
            strcat(ADR(Handling.Data.NewFileName),ADR('-'));
            si := UDINT_TO_STRING(i);
            strcat(ADR(Handling.Data.NewFileName),ADR(si));
            strcat(ADR(Handling.Data.NewFileName),ADR('.txt'));
        ELSIF Handling.Functionblock.FileCopy_0.status = 20700 THEN
            Handling.Data.Step := 51;
        ELSE  (* Goto Error Step *)
            //Handling.Data.Step := 255;
        END_IF

51:         //erzeugt den Ordner 'Results' auf dem USB-Stick (falls nicht
vorhanden), in den die Ergebnisse dann kopiert werden
        DirCreate_0.enable := 1;
        DirCreate_0.pDevice := ADR(gDevice);
        DirCreate_0.pName := ADR('Results');
        DirCreate_0();
        IF DirCreate_0.status = 0 THEN
            Handling.Data.Step := 50;
        END_IF;

60: (* rename a file *)
```

```
                        Handling.Functionblock.FileRename_0.enable := 1;
                        Handling.Functionblock.FileRename_0.pDevice :=
ADR(Handling.Data.Device);  (* name of the linked device *)
                        Handling.Functionblock.FileRename_0.pName :=
ADR(Handling.Data.FileName);  (* actual file name *)
                        Handling.Functionblock.FileRename_0.pNewName :=
ADR(Handling.Data.NewFileName);  (* new file name *)
                        Handling.Functionblock.FileRename_0;  (* call the function*)

                        IF Handling.Functionblock.FileRename_0.status = 0 THEN  (* FileRename
successful *)
                                Handling.Data.Step := 1;  (* next Step*)
                                Handling.Command.bRenameFile := 0;  (* clear command *)
                        ELSIF Handling.Functionblock.FileRename_0.status = ERR_FUB_BUSY THEN
(* FileRename not finished -> redo *)
                                (* Busy *)
                        ELSE  (* Goto Error Step *)
                                Handling.Data.Step := 255;
                        END_IF
                        Zoom := 1;
                        Scroll := 1;


                80: (**** Error step ****)
                        bOK := FALSE;
                81: (**** Get directory info ****)

                        IF sFileName[iFileChoice] = '.' OR sFileName[iFileChoice] = '..' AND bPath = 1
THEN
                                DRead.pPath := 0;
                                sPath := '';
                                bPath := 0;
ELSIF FIND(sFileName[iFileChoice],'.txt') = 0 AND bOK = 1 AND DRead.pPath <> 0 AND bPath = 1 THEN
                                strcat(ADR(sPath), ADR('\'));
                                strcat(ADR(sPath), ADR(sFileName[iFileChoice]));
                                DRead.pPath     := ADR(sPath);
                                bPath := 0;

ELSIF FIND(sFileName[iFileChoice],'.txt') = 0 AND bOK = 1 AND bPath = 1 THEN //
ReadData[iFileChoice].Filelength = 0
                                //bTest := 1;
                                sPath := sFileName[iFileChoice];
                                DRead.pPath     := ADR(sPath);
                                bPath := 0;
                        END_IF;

                        ReadDir := 0;
                        (* Initialize info structure *)
```

```
		DInfo.enable	:= 1;
		DInfo.pDevice	:= ADR(gDevice);
		IF sPath <> '' THEN
			DInfo.pPath	:= ADR(sPath);
		ELSE
			DInfo.pPath	:= 0;
		END_IF;

		(* Call FBK *)
		DInfo();
		(* Get FBK output information *)
		wStatus		:= DInfo.status;
		dwDirNum	:= DInfo.dirnum;
		dwFileNum	:= DInfo.filenum;
		(* Verify status *)
		IF (wStatus = 0) THEN
			Handling.Data.Step := 82;
			(* Verify number of files found *)
		ELSE
			IF (wStatus <> 65535) THEN
				byErrorLevel := 1;
				//Handling.Data.Step := 1;
				IF (wStatus = 20799) THEN
					wError := FileIoGetSysError();
				END_IF
			END_IF
		END_IF

dwCounter := 0;
82: (**** Get file info ****)
		bPath := 1;
		(* Verify counter variable *)
		IF (dwCounter < 10) THEN//dwFileNum+dwDirNum) THEN
			(* Initialize read directory structure *)
			DRead.enable	:= 1;
			DRead.pDevice	:= ADR(gDevice);
			//DRead.pPath	:= ADR('FRU');
			DRead.entry	:= dwCounter;
			DRead.option	:= fiBOTH;
			DRead.pData	:= ADR(ReadData[dwCounter]);
			DRead.data_len	:= SIZEOF(ReadData[0]);
			(* Call FBK *)
			DRead();

			(* Get status *)
			wStatus := DRead.status;
			(* Verify status *)
			IF (wStatus = 0) THEN
```

```
                                        charCounter := 0;


        strcpy(ADR(sFileName[dwCounter]),ADR(ReadData[dwCounter].Filename[charCounter]));
                                                charCounter := charCounter + 1;
                                        IF dwCounter >= dwDirNum + dwFileNum THEN
                                                strcpy(ADR(sFileName[dwCounter]),ADR(''));
                                        END_IF;


                                        dwCounter := dwCounter + 1;
                                ELSE
                                        IF wStatus = 20702 THEN
                                                strcpy(ADR(sFileName[dwCounter]),ADR(''));
                                                dwCounter := dwCounter + 1;
                                                bOK := 1;
                                        ELSIF (wStatus <> 65535) THEN
                                                byErrorLevel := 2;
                                                IF (wStatus = 20799) THEN
                                                        wError := FileIoGetSysError();
                                                END_IF
                                        END_IF
                                END_IF
                        ELSE
                                bOK     := TRUE;
                                Handling.Data.Step := 1;
                        END_IF



        END_CASE




END_PROGRAM
```

## main (converts some commonly used variables)

```
PROGRAM _INIT

END_PROGRAM

PROGRAM _CYCLIC

        //Overheating protection
locTemp1 := INT_TO_REAL(gTemp[0]) / 10;
        IF locTemp1 > 1050 THEN
```

```
            gOverheat := TRUE;
            gVoltage1 := 0;
        END_IF;
        IF locTemp1 < 900 THEN
            gOverheat := FALSE;
        END_IF;
        FOR i:= 0 TO 5 BY 1 DO
            gTempR[i] := INT_TO_REAL(gTemp[i])/10;
        END_FOR;


        //Thermocouple display for GUI, shows a broken connection
        FOR i:= 0 TO 5 BY 1 DO
        IF gTemp[i] = 32767 THEN
            Tcon[i] := 0;
        ELSE
            Tcon[i] := 1;
            END_IF;
        END_FOR;

        //Header GUI
        sTemp := REAL_TO_STRING(gTempR[k-1]);
        strcat(ADR(sTemp),ADR('°C'));
        sPower := INT_TO_STRING(Power);
        strcat(ADR(sPower),ADR('%'));

        //Offgas concentration conversion

        offgasR[0] := (INT_TO_REAL(offgas[0]))*25/32767; //Oxygen [%]
        offgasR[1] := (INT_TO_REAL(offgas[1]))*10000/32767; //CO [ppm]
        offgasR[2] := (INT_TO_REAL(offgas[2]))*20/32767; //CO2 [%]

        //pressure calculation if a pressure cell should be attached

        pressureIsR[0] := (INT_TO_REAL(pressureIs[0])*0.0006104-4)/16*50;
        bMFCready := 1;
        FOR i := 0 TO 2 BY 1 DO
            IF ABS(MFCLis[i]-MFCsetL[i]) > 0.1 AND MFCis[i] <> 32767 THEN
                bMFCready := 0;
            END_IF;
        END_FOR;


END_PROGRAM
PROGRAM _EXIT



END_PROGRAM
```

## read_data (connects to USB drives)

```
//checks for connected devices and stores them in gDevice and gPath

PROGRAM _INIT


	step := WAIT;


END_PROGRAM


PROGRAM _CYCLIC

CASE step OF

	WAIT:

		IF gCheckDevice = TRUE THEN
			step := CREATE_NODE_ID_LIST;  (*start FUBs below*)
		ELSE
			step := WAIT;
		END_IF;
			IF usb_data_buffer[0].ifName <> '' THEN
				DeviceConnection := 1;
				gConnect := 1;
			ELSE
				DeviceConnection := 0;
			END_IF;



	CREATE_NODE_ID_LIST:  (*Library AsUSB - Functionblock USBNodeListGet()*)

			UsbNodeListGet_0.enable := 1;
UsbNodeListGet_0.pBuffer := ADR(node_id_buffer);  (*pointer to buffer - UDINT array is assigned*)
UsbNodeListGet_0.bufferSize := SIZEOF(node_id_buffer);  (*size of node-id-buffer-array*)
UsbNodeListGet_0.filterInterfaceClass := asusb_CLASS_MASS_STORAGE;  (*filter on mass storage
devices is set*)
			UsbNodeListGet_0.filterInterfaceSubClass := 0;  (*no filer is set*)
			UsbNodeListGet_0;

			IF UsbNodeListGet_0.status = 0 THEN
	step := READ_DEVICE_DATA;  (*FUB worked correctly => next step*)
			ELSIF UsbNodeListGet_0.status = ERR_FUB_BUSY THEN
```

```
        step := CREATE_NODE_ID_LIST;  (*FUB work asynchron => called until status is not BUSY*)
                ELSE
                        //step := ERROR_CASE;  (*error occured*)
                END_IF;



        READ_DEVICE_DATA:  (*Library AsUSB - Functionblock USBNodeGet()*)

                UsbNodeGet_0.enable := 1;
UsbNodeGet_0.nodeId := node_id_buffer[node];  (*specific node is read out of node_id_buffer*)
UsbNodeGet_0.pBuffer := ADR(usb_data_buffer[node]);  (*data of specific node get stored in
usb_data_buffer*)
UsbNodeGet_0.bufferSize := SIZEOF (usb_data_buffer[node]);  (*size of specific node is read out
usb_data_buffer*)
                UsbNodeGet_0;

                IF UsbNodeGet_0.status = 0 THEN  (*FUB worked correctly*)
                        node := node + 1;  (*next node to be read out of buffer*)
                        IF node = UsbNodeListGet_0.listNodes THEN  (*last existing node is reached*)
                                node := 0;
                                step := GET_DESCRIPTOR;  (*all nodes are read out of buffer*)
                        END_IF;

                ELSIF UsbNodeGet_0.status = ERR_FUB_BUSY THEN
                        step := READ_DEVICE_DATA;  (*FUB work asynchron => called until status
isn't BUSY*)
                ELSE
                        step := ERROR_CASE;  (*error occured*)
                END_IF;



        GET_DESCRIPTOR:  (*Library AsUSB - Functionblock USBDescriptorGet()*)

                UsbDescriptorGet_0.enable := 1;
UsbDescriptorGet_0.nodeId := node_id_buffer[node];  (*specific node is read out of
node_id_buffer*)
UsbDescriptorGet_0.requestType := 0;  (*Request for device*)
UsbDescriptorGet_0.descriptorType := 1;  (*Determines the device descriptor*)
UsbDescriptorGet_0.languageId := 0;  (*for device and configuration descriptors*)
UsbDescriptorGet_0.pBuffer := ADR(device_descriptor[node]);  (*descriptor-data of specific node get
stored in device_descriptor-buffer*)
UsbDescriptorGet_0.bufferSize := SIZEOF(device_descriptor[node]);  (*size of specific node is read
out device_descriptor-buffer*)
                UsbDescriptorGet_0;

                IF UsbDescriptorGet_0.status = 0 THEN  (*FUB worked correctly*)
```

```
                node := node + 1;  (*next node to be read out of buffer*)
                IF node = UsbNodeListGet_0.listNodes THEN  (*last existing node is reached*)
                        node := 0;
                        step := CREATE_FILE_DEVICE;  (*all nodes are read out of buffer*)
                END_IF;

        ELSIF UsbDescriptorGet_0.status = ERR_FUB_BUSY THEN
                step := GET_DESCRIPTOR;  (*FUB work asynchron => called until status isn't
BUSY*)
        ELSE
                step := ERROR_CASE;  (*error occured*)
        END_IF;



    CREATE_FILE_DEVICE:  (*Library FileIO - Functionblock DevLink() - create file out of data just
from 1. USB*)

        strcpy(ADR(device_name), ADR('DEVICE1'));  (*fixed Device-Name get copied to
device_name-Variable*)

        strcpy(ADR(device_param), ADR('/DEVICE='));  (*first part of parameter get copied to
device_param-Variable*)
        strcat(ADR(device_param), ADR(usb_data_buffer[0].ifName));  (*second part get
added to device_param-Variable*)



        DevLink_0.enable := 1;
        DevLink_0.pDevice := ADR(device_name);  (*Devicename is assigned*)
        DevLink_0.pParam := ADR(device_param);  (*the path of the Device is assigned*)
        DevLink_0;

        IF DevLink_0.status = 0 THEN
                step := DIRECTORY_INFO;  (*FUB worked correctly => next step*)
        ELSIF DevLink_0.status = ERR_FUB_BUSY THEN
                step := CREATE_FILE_DEVICE;  (*FUB work asynchron => called until status
isn't BUSY*)
        ELSE
                step := ERROR_CASE;  (*error occured*)
        END_IF;
                strcpy(ADR(gDevice), ADR(device_name));
        strcpy(ADR(gPath), ADR(device_param));


    DIRECTORY_INFO:  (*Library FileIO - Functionblock DirInfo()*)

        DirInfo_0.enable := 1;
        DirInfo_0.pDevice := ADR(device_name);  (*Devicename is assigned*)
```

```
		DirInfo_0.pPath := 0;  (*no path is needed => value = 0*)
		DirInfo_0;

		IF DirInfo_0.status = 0 THEN
			step := DIRECTORY_READ;  (*FUB worked correctly => next step*)
		ELSIF DirInfo_0.status = ERR_FUB_BUSY THEN
			step := DIRECTORY_INFO;  (*FUB work asynchron => called until status isn't
BUSY*)
		ELSE
			step := ERROR_CASE;  (*error occured*)
		END_IF;



	DIRECTORY_READ:  (*Library FileIO - Functionblock DirRead()*)

		DirRead_0.enable := 1;
		DirRead_0.pDevice := ADR(device_name);  (*Devicename is assigned*)
		DirRead_0.pPath := 0;  (*no path is needed => value = 0*)
		DirRead_0.entry := 1;  (*only the first entry is read out*)
		DirRead_0.option := fiBOTH;  (*Collect information from files and directories*)
		DirRead_0.pData := ADR(device_data);  (*device_data-variable with
fiDIR_READ_DATA type is assigned to data-buffer*)
		DirRead_0.data_len := SIZEOF(device_data); (*size of device_data-variable is read
out*)
		DirRead_0;

		IF DirRead_0.status = 0 THEN
			step := FINISH; //UNLINK_DEVICE;  (*FUB worked correctly => next step*)

		ELSIF DirRead_0.status = ERR_FUB_BUSY THEN
			step := DIRECTORY_READ;  (*FUB work asynchron => called until status isn't
BUSY*)
		ELSE
			step := ERROR_CASE;  (*error occured*)
		END_IF;



	UNLINK_DEVICE:  (*Library FileIO - Functionblock DevUnlink()*)

		DevUnlink_0.enable := 1;
		DevUnlink_0.handle := DevLink_0.handle;  (*handle from DevLink is assigned to cut
the connection to specific file-device*)
		DevUnlink_0;

		IF DevUnlink_0.status = 0 THEN
			step := FINISH;  (*FUB worked correctly => next step*)
```

```
                ELSIF DevUnlink_0.status = ERR_FUB_BUSY THEN
                        step := UNLINK_DEVICE;  (*FUB work asynchron => called until status isn't
BUSY*)
                ELSE
                        step := ERROR_CASE;  (*error occured*)
                END_IF;




        FINISH:  (*successfully finished*)

                gCheckDevice := FALSE;
                step := WAIT;  (*back to beginning - wait for start_reading_usb_data to be set*)



        ERROR_CASE:   (*error-handling*)


END_CASE;

END_PROGRAM
```

# Variables

## altControl

| | Typ | Retain | W... | Beschreibung [1] |
|---|---|---|---|---|
| actValue | INT | ☐ | | derzeitige Temperatur |
| addParameter | lcpid_tune_addpar_typ | ☐ | | |
| avgPower | INT | ☐ | | über 60s gemittelte Leistung die gehalten werden kann (starten mit bAverage) |
| baseLCCounter | LCCounter | ☐ | | Zähler für PID-Regler und Autotune |
| bAverage | BOOL | ☐ | | initiiert Mittelung der Leistung, anschließend wird die mittlere Leistung gehalten |
| bHolding | INT | ☐ | | zeigt an ob die mittlere Leistung gehalten wird |
| bMFCzero | BOOL | ☐ | | veranlasst die Nullsetzung der MFCs 0-2 |
| boxValueR | REAL | ☐ | | im Fenster eingestellte Temperatur die bei konstanter Betriebsweise zur Solltemperatur wird |
| calValue | INT | ☐ | | Wert bei dem die Kalibration durchgeführt wird (x10) |
| calValuer | REAL | ☐ | | Wert bei dem die Kalibration durchgeführt wird |
| changePara | BOOL | ☐ | | veranlasst die Änderung der PID-Parameter |
| charCurveLCCurveByPoints | LCCurveByPoints[0..4] | ☐ | | |
| ControlSet | USINT | ☐ | | außer Verwendung |
| GasSet | USINT | ☐ | | Stellzustand des MFC-Betriebes |
| HeatSet | USINT | ☐ | | Stellzustand des Heizungsbetriebes |
| i | INT | ☐ | | Index |
| j | UINT | ☐ | | Jott nicht 'i', Index |
| LCPIDpara_1 | LCPIDpara | ☐ | | Parameter für den PID-Regler |
| LCPIDTune_0 | LCPIDTune | ☐ | | |
| LCPID_0 | LCPID | ☐ | | |
| LCPWM_0 | LCPWM | ☐ | | Pulswellenmodulator, wandelt eine Sollleistung in eine periodische Rechtecksfunktion die diskret auf 0 oder 100% springt |
| manipulatedVar | INT | ☐ | | vom PID-Regler berechnete Sollleistung |
| MFC1Table | lcCurveByPoints_TabEntry_type[0..255] | ☐ | | MFC-Durchfluss für das Programm |
| MFC2Table | lcCurveByPoints_TabEntry_type[0..255] | ☐ | | MFC-Durchfluss für das Programm |
| MFC3Table | lcCurveByPoints_TabEntry_type[0..255] | ☐ | | MFC-Durchfluss für das Programm |
| MFC4Table | lcCurveByPoints_TabEntry_type[0..255] | ☐ | | nicht in Verwendung |
| MFCsetnew | BOOL | ☐ | | veranlasst Übernahme der Gas-Solldurchflüsse aus dem VNC-Fenster, bis dahin werden die alten Werte weiterverwendet |
| MFCsetVNC | REAL[0..2] | ☐ | | Gas-Solldurchflüsse aus dem VNC-Fenster, müssen mit MFCsetnew übernommen werden |
| MFCzero | REAL[0..3] | ☑ | | Nullwerte der MFCs |
| oscOptions | lcpid_tune_osc_options_typ | ☐ | | |
| PID_Kp | LREAL[0..5] | ☐ | | PID-Proportionalverstärkung für alle 6 Thermoelemente (nicht vollständig implementiert) |
| PID_Tn | LREAL[0..5] | ☐ | | PID-Integralverstärkung für alle 6 Thermoelemente (nicht vollständig implementiert), kleinere Werte verstärken mehr |
| PID_Tv | LREAL[0..5] | ☐ | | PID-Differentialverstärkung für alle 6 Thermoelemente (nicht vollständig implementiert) |
| pulse | BOOL | ☐ | | zeigt an ob der PWM bei 0 oder 100% ist |
| refLCCounter | LCCounter | ☐ | | misst die genaue Zeit für den PID-Regler |
| setValue1 | INT | ☐ | | Solltemperatur in °C |
| start | BOOL | ☐ | | PID-Einschalter |
| stepOptions | lcpid_tune_step_options_typ | ☐ | | |
| sumPower | DINT | ☐ | | Summe der Leistungen in % für die Leistungsmittelung |
| tuningRequest | UDINT | ☐ | | Einstellung des Autotune (0: aus, 1: cyclisch, 2: Sprungfunktion) |
| yTable | lcCurveByPoints_TabEntry_type[0..255] | ☐ | | Tabelle mit den Temperaturwerten der eventuell programmierten Temperaturkurve |

## FileHandling

| | Typ | Retain | Wert | Beschreibung [1] |
|---|---|---|---|---|
| bCopied | INT | ☐ | 0 | gibt an, ob die Textdatei mit den Messwerten auf den USB-Stick kopiert wurde, löst das grüne Licht im VNC aus |
| bHasRead | INT | ☐ | 0 | |
| bMeasuring | INT | ☐ | 0 | gibt an, ob Messwerte aufgezeichnet werden, löst das entsprechende grüne Licht im VNC aus |
| bOK | BOOL | ☐ | | |
| bPath | USINT | ☐ | 1 | |
| bRate | BOOL | ☐ | | |
| byErrorLevel | USINT | ☐ | | |
| charCounter | USINT | ☐ | | |
| charFileName | STRING[10] | ☐ | | |
| DesReset | BOOL | ☐ | | experimentell |
| DevUnlink_0 | DevUnlink | ☐ | | |
| DInfo | DirInfo | ☐ | | |
| DirClose_0 | DirClose | ☐ | | |
| DirCreate_0 | DirCreate | ☐ | | |
| DirOpen_0 | DirOpen | ☐ | | |
| DRead | DirRead | ☐ | | |
| DTGetTime_0 | DTGetTime | ☐ | | |
| DTSGetTime_0 | DTStructureGetTi... | ☐ | | |
| DTStructure_0 | DTStructure | ☐ | | |
| dwCounter | UDINT | ☐ | | |
| dwDirNum | UDINT | ☐ | | |
| dwFileNum | UDINT | ☐ | | |
| EndTime | INT | ☐ | | |
| Error | UINT | ☐ | | return value from FileIoGetSysError() |
| Handling | FileHandling_typ | ☐ | (0) | gesamte Filehandling-Struktur mit allen Funktionen |
| i | UDINT | ☐ | | |
| iFileChoice | USINT | ☐ | | Nummer der gewählten Datei in der Drop-Down-Liste |
| iScroll | USINT | ☐ | | Bildlauf der Zeitaches der Temperatur- und Gas-Graphen (6min pro Punkt) |

| | | | | |
|---|---|---|---|---|
| iZoom | USINT | ☐ | | Vergrößerung der Zeitachse der Temperatur- und Gas-Graphen |
| LesePos | INT | ☐ | 0 | Leseposition, in Byte vom Anfang der Textdatei |
| lList1 | USINT | ☐ | 3 | unwichtig |
| lList2 | USINT | ☐ | 1 | das wichtige |
| lMinute | STRING[80] | ☐ | | |
| locInfo | FileInfo | ☐ | | |
| locInfo2 | fiFILE_INFO | ☐ | | locInfo2.size ist die Schreibposition in der Messdatei |
| locString1 | STRING[80] | ☐ | | enthält den Zeilenseparator '$R$N' |
| locTemp4 | STRING[80] | ☐ | '0' | Variable, in der die zu speichernden Zahlenwerte als String kurz gespeichert werden, bevor sie in lTime angehängt werden |
| locZahl3 | USINT | ☐ | 0 | |
| lRead | BOOL | ☐ | | |
| lSecond | STRING[80] | ☐ | | |
| lSep | STRING[80] | ☐ | ';' | Separator/Trennzeichen |
| lSp | STRING[80] | ☐ | ' ' | Space/Abstandhalter |
| lTime | STRING[80] | ☐ | | String, in dem die in die Textdatei zu speichernden Daten stückweise angehängt werden bevor sie in die Textdatei kopiert werden |
| MaxOxy | REAL | ☐ | | experimentell |
| NewFilePath | STRING[40] | ☐ | | Dateiname auf dem USB-Stick für die kopierte Messdatendatei |
| OxyAds | REAL | ☐ | | experimentell |
| OxyDes | REAL | ☐ | | experimentell |
| OxyVolume | REAL | ☐ | | experimentell |
| Rate | REAL | ☐ | | gibt an, ob der Zeitpunkt im T/MFC-Programm über die Aufheizrate zu berechnen ist |
| ReadData | fiDIR_READ_DA... | ☐ | | |
| ReadDir | BOOL | ☐ | | |
| RepZeile | INT | ☐ | 0 | |
| RepZeilenZahl | INT | ☐ | 0 | gibt an, wieviel Zeilen des Programmes zu wiederholen sind |
| RepZyklenZahl | INT | ☐ | 0 | Wieoft die Zeilen zu wiederholen sind |
| RepZyklus | INT | ☐ | 0 | Beschreibt beim wiederholten Schreiben der Zeilen, beim wievielten Zyklus das Programm ist |

| | | | | |
|---|---|---|---|---|
| rtc_gettime | RTCtime_typ | ☐ | | |
| SampleCount | INT | ☐ | | für Voranzeige der Temperatur-/Gasprogramme im Graphen |
| SampleNumber | INT | ☐ | | für Voranzeige der Temperatur-/Gasprogramme im Graphen |
| SampleTime | REAL | ☐ | | für Voranzeige der Temperatur-/Gasprogramme im Graphen |
| SampleTimeMS | DINT | ☐ | | für Voranzeige der Temperatur-/Gasprogramme im Graphen |
| SampleValue | INT[0..675] | ☐ | | für Voranzeige der Temperatur-/Gasprogramme im Graphen |
| SampleValue1 | INT[0..675] | ☐ | | für Voranzeige der Temperatur-/Gasprogramme im Graphen |
| SampleValue1R | REAL[0..675] | ☐ | | für Voranzeige der Temperatur-/Gasprogramme im Graphen |
| SampleValue2 | INT[0..675] | ☐ | | für Voranzeige der Temperatur-/Gasprogramme im Graphen |
| SampleValue2R | REAL[0..675] | ☐ | | für Voranzeige der Temperatur-/Gasprogramme im Graphen |
| SampleValue3 | INT[0..675] | ☐ | | für Voranzeige der Temperatur-/Gasprogramme im Graphen |
| SampleValue3R | REAL[0..675] | ☐ | | für Voranzeige der Temperatur-/Gasprogramme im Graphen |
| SampleValue4 | INT[0..675] | ☐ | | für Voranzeige der Temperatur-/Gasprogramme im Graphen |
| SampleValue4R | REAL[0..675] | ☐ | | für Voranzeige der Temperatur-/Gasprogramme im Graphen |
| Scroll | REAL | ☐ | 1.0 | Scrollwert der in VNC links von der Lieste angezeigt wird |
| sDatenzeile | STRING[255] | ☐ | | |
| sDatum | STRING[20] | ☐ | | |
| sDay | STRING[10] | ☐ | | |
| Select | USINT | ☐ | | variable to select the file-device |
| sFileName | STRING[100][0..... | ☐ | | |
| si | STRING[5] | ☐ | | |
| sMonth | STRING[10] | ☐ | | |
| SorpLen | REAL | ☐ | | experimentell |
| sPath | STRING[80] | ☐ | | |
| sPathOpened | STRING[80] | ☐ | | |
| sSettingsFile | STRING[40] | ☐ | | |
| startDT | DATE_AND_TIME | ☐ | | |

| | | | | |
|---|---|---|---|---|
| strDevice | STRING[80] | ☐ | | |
| TabPos | INT | ☐ | 0 | Position des nächsten Tabulators |
| TimeIncrement | INT | ☐ | | |
| wError | UINT | ☐ | | |
| wStatus | UINT | ☐ | | |
| ZeilenPos1 | INT | ☐ | | |
| Zoom | REAL | ☐ | 1.0 | Zoomwert der in VNC links von der Lieste angezeigt wird |

## read_data

| | Typ | Konstante | Retain | Wert | Beschreibung [1] |
|---|---|---|---|---|---|
| CREATE_FILE_DEVICE | USINT | ☑ | ☐ | 5 | Function step: Create file out of data from USB device |
| CREATE_NODE_ID_LIST | USINT | ☑ | ☐ | 2 | Function step: Create a list of Node-IDs from all active USB devices |
| DeviceConnection | USINT | ☐ | ☐ | 0 | |
| device_data | fiDIR_READ_DA... | ☐ | ☐ | (0) | Buffer to store device data |
| device_descriptor | usbDeviceDescr_... | ☐ | ☐ | | Buffer to store descriptor data of device |
| device_name | STRING[80] | ☐ | ☐ | | Name of the USB device - optional |
| device_param | STRING[80] | ☐ | ☐ | | Connection path of USB device |
| DevLink_0 | DevLink | ☐ | ☐ | | Type of FUB DevLink |
| DevUnlink_0 | DevUnlink | ☐ | ☐ | | Type of FUB DevUnlink |
| DIRECTORY_INFO | USINT | ☑ | ☐ | 6 | Function step: Read out the Info about the connected File Device |
| DIRECTORY_READ | USINT | ☑ | ☐ | 7 | Function step: Read out to data from linked File Device |
| DirInfo_0 | DirInfo | ☐ | ☐ | | Type of FUB DirInfo |
| ERROR_CASE | USINT | ☑ | ☐ | 1 | Function step: Jump to this step in every case of Error occuring in the Function chain |
| FINISH | USINT | ☑ | ☐ | 8 | Function step: Function chain is finished - back to WAIT |
| GET_DESCRIPTOR | USINT | ☑ | ☐ | 4 | Function step: Read out the descriptor data from the Node-IDs |
| locString2 | STRING[80] | ☐ | ☐ | | |
| locZahl6 | USINT | ☐ | ☐ | 0 | |
| node | USINT | ☐ | ☐ | | Node number |
| node_id_buffer | UDINT[0..4] | ☐ | ☐ | | Different nodeIDs saved in array-elements |
| READ_DEVICE_DATA | USINT | ☑ | ☐ | 3 | Function step: Read out the specific data from the Node-IDs |
| start_reading_usb_data | BOOL | ☐ | ☐ | | If TRUE - Function chain gets started |
| step | USINT | ☐ | ☐ | | Determines the Function step |
| UNLINK_DEVICE | USINT | ☑ | ☐ | 9 | Function step: Unlink the File Device from specific path |
| UsbDescriptorGet_0 | UsbDescriptorGet | ☐ | ☐ | | Type of FUB UsbDescriptorGet |
| UsbNodeGet_0 | UsbNodeGet | ☐ | ☐ | | Type of FUB UsbNodeGet |
| UsbNodeListGet_0 | UsbNodeListGet | ☐ | ☐ | | Type of FUB UsbNodeListGet |
| usb_data_buffer | usbNode_typ[0..4] | ☐ | ☐ | | Defined data_typ of AsUSB - Library |
| WAIT | USINT | ☑ | ☐ | 0 | Function step: Wait for starting the Function chain |

## Global

| | Typ | Konstante | Retain | Wert | Beschreibung [1] |
|---|---|---|---|---|---|
| bMFCready | USINT | ☐ | ☐ | | zeigt an ob alle Durchflüsse unter 0,1 von der Vorgabe abweichen; in der Aufwärmphase zeigen MF |
| DirRead_0 | DirRead | ☐ | ☐ | | Type of FUB DirRead |
| gasChoice | INT[0..3] | ☐ | ☑ | | speichert welches Gas an welchem MFC angeschlossen ist |
| gasCoef | REAL[0..3] | ☐ | ☐ | | speichert die Gaskoeffizienten für die Gase |
| gasList | STRING[8][0..3] | ☐ | ☐ | | Liste der möglichen Gase als Text |
| gCheckDevice | BOOL | ☐ | ☐ | FALSE | startet in read_data die Suche nach einem USB-Stick |
| gConnect | BOOL | ☐ | ☐ | FALSE | zeigt an ob ein USB-Stick gefunden wurde |
| gCreate | BOOL | ☐ | ☐ | FALSE | veranlasst das Erzeugen einer neuen Datei für die Messergebnisse |
| gDevice | STRING[80] | ☐ | ☐ | | interner Name des USB-Sticks |
| gMeasure | BOOL | ☐ | ☐ | FALSE | zeigt an ob Daten aufgenommen werden |
| gOverheat | BOOL | ☐ | ☐ | FALSE | zeigt an ob einer der Messfühler über 1050°C misst (jedoch nicht 3276,7 da dies einen Kabelbruch b |
| gPath | STRING[80] | ☐ | ☐ | | interner Name für den Pfad des USB-Sticks |
| gRead | BOOL | ☐ | ☐ | FALSE | veranlasst das Auslesen des Temperatur-/Gasprogrammes |
| gStep | USINT | ☐ | ☐ | 0 | Kopie von Handling.Data.Step, zeigt was FileHandling gerade macht (255 bei Fehler) |
| gTemp | INT[0..5] | ☐ | ☐ | 6(0) | 10faches der Temperatur der Thermoelemente |
| gTempR | REAL[0..5] | ☐ | ☐ | 6(0.0) | korrekte Temperatur der Thermoelemente |
| gTest | USINT | ☐ | ☐ | 0 | |
| gTest2 | USINT | ☐ | ☐ | | |
| gVoltage1 | INT | ☐ | ☐ | 0 | Spannung für das Relay der Heizung |
| gVoltage2 | INT | ☐ | ☐ | 0 | |
| k | USINT | ☐ | ☑ | | |
| MFC1 | INT[0..254] | ☐ | ☐ | | Solldurchfluss als Eckpunkte der Programmkurve zur jeweiligen Zeit in x |
| MFC2 | INT[0..254] | ☐ | ☐ | | Solldurchfluss als Eckpunkte der Programmkurve zur jeweiligen Zeit in x |
| MFC3 | INT[0..254] | ☐ | ☐ | | Solldurchfluss als Eckpunkte der Programmkurve zur jeweiligen Zeit in x |
| MFC4 | INT[0..254] | ☐ | ☐ | | Solldurchfluss als Eckpunkte der Programmkurve zur jeweiligen Zeit in x |
| MFCChoice | INT[0..3] | ☐ | ☑ | | gibt an, welches MFC an das jeweilige Kabel angesteckt ist |
| MFCCoef | REAL[0..5] | ☐ | ☐ | | Koeffizienten der jeweiligen MFCs |
| MFCis | INT[0..3] | ☐ | ☐ | | MFC-Istdurchfluss als Spannung |
| MFCLis | REAL[0..3] | ☐ | ☐ | | MFC-Istdurchfluss in Liter |
| MFCList | STRING[10][0..5] | ☐ | ☐ | | Liste der MFCs als Text |
| MFCset | INT[0..3] | ☐ | ☐ | | MFC-Stellwert als Spannung |
| MFCsetL | REAL[0..3] | ☐ | ☐ | | MFC-Stellwert in Liter |
| offgas | INT[0..7] | ☐ | ☐ | | Abgaskonzentration als Spannung |
| offgasR | REAL[0..7] | ☐ | ☐ | | Abgaskonzentration (0: O2[%], 1: CO[ppm], 2: CO2[%], 3-7: - ) |
| Power | INT | ☐ | ☐ | | relative Leistung der Heizung in Prozent |
| pressureIs | INT[0..1] | ☐ | ☐ | | Druck, welcher über die Druckmessdosen gemessen werden kann, als Spannung |
| pressureIsR | REAL[0..1] | ☐ | ☐ | | korrekter Druck |
| setValue | INT | ☐ | ☐ | | Solltemperatur |
| Tcon | INT[0..5] | ☐ | ☐ | | zeigt an ob die Thermoelemente verbunden sind (Kabelbruch bei T=3276,7) |
| vMFCis | REAL[0..3] | ☐ | ☐ | | MFC-Istwert als Spannung |
| vMFCset | REAL[0..3] | ☐ | ☐ | 4(0.0) | MFC-Stellwert als Spannung |
| x | INT[0..254] | ☐ | ☐ | | Zeitkoordinaten der Eckpunkte der T/MFC-Kurve in s |
| y | INT[0..254] | ☐ | ☐ | | Solltemperaturen des Programmes zur jeweiligen Zeit in x |
| ZeilenPos | INT | ☐ | ☐ | 0 | Position der derzeitigen Zeile in der Datenabfolge |

## main

| | Typ | Konstante | Retain | Wert | Beschreibung [1] |
|---|---|---|---|---|---|
| sTemp | STRING[7] | ☐ | ☐ | | |
| sPower | STRING[5] | ☐ | ☐ | | |