Electrical and Firmware Design of a Compact Multi-Channel Radiometer Prototype for Application on a Mars Weather Balloon

Thesis B.Sc. Technische Informatik

Luca Moritz Jungnickel
406963

10.04.2023

Supervisor: Prof. Dr.-Ing. Roland Thewes

Technische Universität Berlin
School of Electrical Engineering and Computer Science
Department of Energy and Automation Technology
Chair of Sensor and Actuator Systems

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den 10.04.2023

Luca Jungnickel

Abstract

This thesis presents the development of a radiometer prototype for a potential application on a Mars weather balloon in the context of a Mars exploration mission proposal by DLR (Deutsches Zentrum für Luft- und Raumfahrt - German Aerospace Center). The instrument measures radiative flux in three dedicated wavelength bands using thermopile sensors, the design is based on heritage from radiometers flown on previous space exploration missions. In this thesis, the design is adapted to the balloon's flight avionics and its electrical and data interfaces. The work focuses on the development of the electronics schematics design, the printed circuit board layout, and the custom firmware. Furthermore, integration, commissioning, as well as functional and performance testing are discussed. Performance of the instrument is validated and the offset temperature drift of the measured signals as well as the excellent noise performance of the prototype is demonstrated.

Zusammenfassung

Die folgende Abschlussarbeit präsentiert die Entwicklung eines Radiometers als Prototyp für eine mögliche Anwendung auf einem Mars-Wetterballon im Rahmen einer zukünftigen Marserkundigungsmission des DLR (Deutsches Zentrum für Luft- und Raumfahrt). Das Instrument misst Strahlungsfluss mithilfe von Thermosäulen in drei verschiedenen Wellenlängenbereichen. Das Design basiert auf bereits entwickelten Radiometern, welche auf vorherigen Missionen zur Erforschung des Weltraums eingesetzt wurden. In dieser Arbeit wurde das Design an die bestehende Flugavionik und deren elektronische sowie Datenschnittstellen des Wetterballons angepasst. Die Arbeit fokussiert sich auf die Entwicklung der Schaltungen, dem Leiterplattendesign sowie der Firmware des Messinstruments. Zusätzlich werden Integration, Inbetriebnahme, Funktions- und Performancetests, welche erstellt und durchgeführt wurden, vorgestellt. Die Performance des Messinstruments wird analysiert, der Offsetdrift der Messsignale sowie das hervorragende Rauschverhalten des Instrumentes werden hier aufgezeigt.

Contents

List of Figures xi

List of Tables xiii

List of Abbreviations xv

1. Introduction 1
- 1.1. General Overview . . . 1
- 1.2. Motivation and Objectives . . . 1
- 1.3. Goal of this Thesis . . . 2
- 1.4. Structure of this Document . . . 2

2. Theory and Approach 3
- 2.1. Fundamentals of Radiometry . . . 3
  - 2.1.1. Blackbody Radiation . . . 4
  - 2.1.2. Greybody Radiation . . . 4
  - 2.1.3. Planck's Law . . . 4
- 2.2. Platinum Resistance Thermometers . . . 5
- 2.3. Thermopile Sensors . . . 6
  - 2.3.1. Seebeck Effect . . . 6
  - 2.3.2. Thermocouple . . . 6
  - 2.3.3. Thermopile . . . 7
  - 2.3.4. Measurement Principle . . . 7
  - 2.3.5. Brightness Temperature Deviation . . . 8
  - 2.3.6. NETD and Performance . . . 9
  - 2.3.7. IPHT Thermopile Sensors . . . 9
- 2.4. Analog to Digital Conversion . . . 10
- 2.5. Signal Path and Conditioning . . . 11

3. Design and Development 13
- 3.1. Design Overview . . . 13
  - 3.1.1. Electrical Interface . . . 16
- 3.2. Mechanical Design . . . 16
  - 3.2.1. Temperature Stabilization of the Sensor Heads . . . 17
- 3.3. Schematics Design . . . 18
  - 3.3.1. Instrumentation Amplifier . . . 19
  - 3.3.2. Pt100 . . . 20
  - 3.3.3. Implementation of Testpoints . . . 21
- 3.4. Circuit Board Design . . . 21
  - 3.4.1. Handling of Sensitive Traces . . . 22
  - 3.4.2. Grounding and Shielding Options . . . 23

3.5. Firmware Design . . . 26
3.5.1. Overview . . . 26
3.5.2. Internal Oversampling . . . 27
3.5.3. Measurement Procedure . . . 27
3.5.4. ADC Configuration . . . 30
3.5.5. External Oversampling . . . 32
3.5.6. Error Handling . . . 33
3.5.7. Data Processing . . . 34
3.6. Startup Procedure . . . 34
3.7. Development Board for Radiometer . . . 36

4. Experiments and Results **37**
4.1. Measurement Configuration Test . . . 37
4.1.1. Interpretation of Measured Voltages . . . 40
4.1.2. Interpretation of Standard Deviation . . . 42
4.2. Temperature Drift . . . 44
4.2.1. Interpretation of Offset Channel Measurements . . . 46
4.2.2. Interpretation of Thermopile Channel Measurements . . . 47
4.2.3. Internal Pt100 Channels . . . 50

5. Summary and Future Work **53**
5.1. Summary . . . 53
5.2. Future Work . . . 53

Bibliography **55**

A. Appendix **57**
A.1. Data Experiment 1 - Measurement Configuration Test . . . 57
A.1.1. External Pt100 channel, replaced with a high tolerance, high TC 100 Ω resistor . . . 60
A.2. Data Experiment 2 - Temperature Drifts . . . 62
A.2.1. Offset Channel . . . 62
A.2.2. Thermopile 1 Channel (100 kΩ - 1 kΩ resistor combination) . . . 63
A.2.3. Thermopile 2 Channel (100 kΩ - 10 kΩ resistor combination) . . . 64
A.2.4. Thermopile 3 Channel (10 kΩ - 10 kΩ resistor combination) . . . 65
A.2.5. Pt100 of Thermopile 1 Channel (substituted with a ±0.1% precision 5 ppm / °C TC 100 Ω resistor) . . . 66
A.2.6. Pt100 of Thermopile 3 Channel (substituted with a ±0.1% precision 5 ppm / °C TC 100 Ω resistor) . . . 67
A.2.7. External Pt100 channel (substituted with a generic 100 Ω, high TC and high tolerance resistor) . . . 68
A.3. MableRad - Schematics and PCB Design . . . 69
A.3.1. Schematics . . . 69
A.3.2. L1 - Top Layer . . . 77
A.3.3. L2 - Ground Layer . . . 78
A.3.4. L3 - Ground and Power Layer . . . 79
A.3.5. L4 - Bottom Layer . . . 80

A.4. MableRad Development Board - Schematics and PCB Design 81
A.4.1. Schematics 81
A.4.2. L1 - Top Layer 82
A.4.3. L2 - Bottom Layer 83
A.5. Firmware Radiometer 84
A.5.1. Ads1258.h 84
A.5.2. Ads1258.cpp 96
A.5.3. Rad.h 106
A.5.4. Rad.cpp 108
A.5.5. Mable-rad-complete-test.cpp 110
A.5.6. Mable-rad-one-channel-test.cpp 113

List of Figures

2.1. Schematic of the mechanical design of a thermopile, modified from [10, p. 189 Figure 7.6] . . . 7
2.2. Block diagram of signal flow and signal amplification of the measurements taken in the developed radiometer . . . 11

3.1. System overview of the radiometer with its component groups and corresponding power rails . . . 15
3.2. System design and integration of the radiometer in the weather balloon . . . 17
3.3. View of the aluminium body . . . 18
3.4. Schematic of the external signal conditioning circuit . . . 19
3.5. Schematic of Pt100 measuring circuit . . . 20
3.6. Computer generated view of the radiometer . . . 23
3.7. Detail of circuit board trace design with three differential traces, surrounded by shielding traces . . . 24
3.8. Composition of internal oversampling of the ADC . . . 27
3.9. Flowchart of function Rad::init in radiometer driver . . . 29
3.10. Flowchart of function Rad::measure in radiometer driver . . . 29
3.11. Flowchart of function Ads1258::getSample in ADS1258 driver . . . 30
3.12. Flowchart of function Ads1258::getDataAndOversample in ADS1258 driver . . . 30
3.13. Radiometer with assembled sensor simulators . . . 35
3.14. Development Board for radiometer "Mable-Rad DevBoard" . . . 36

4.1. Measurement setup for the test of different measurement configurations, for the offset and external Pt100 channel . . . 38
4.2. Offset channel, measured continuously with a 0.1 s sample length . . . 39
4.3. $U(t)$ of Pt100 of thermopile 1 channel with highlighted stable temperature areas 45
4.4. Offset channel's temperature drift of average voltage and standard deviation . . . 46
4.5. Thermopile 3, assembled with 10 kΩ -10 kΩ high precision low TC resistors, its temperature drift of average voltage and standard deviation . . . 47
4.6. Thermopile 1 and 2, their temperature drift of average voltage and corresponding standard deviation . . . 48
4.7. Pt100 of thermopile 1, substituted with a 100 Ω resistor, its temperature drift of average voltage and corresponding standard deviation . . . 50
4.8. Pt100 of thermopile 3, substituted with a 100 Ω resistor, its temperature drift of average voltage and corresponding standard deviation . . . 50
4.9. External Pt100, substituted with a low precision high TC 100 Ω resistor, its temperature drift of average voltage and corresponding standard deviation . . . 51

A.1. Offset channel, measured continuously with a 0.1 s sample length . . . 57
A.2. Offset channel, measured continuously with a 1 s sample length . . . 58

A.3. Offset channel, channel is switched after measurement, 1 s sample length, 0 sample discard . . . 58
A.4. Offset channel, channel is switched after measurement, 1 s sample length, 32 sample discard . . . 59
A.5. External Pt100 channel, replaced with a 100 Ω resistor, measured continuously with a 0.1 s sample length . . . 60
A.6. External Pt100 channel, replaced with a 100 Ω resistor, measured continuously with a 1 s sample length . . . 60
A.7. External Pt100 channel, replaced with a 100 Ω resistor, channel is switched after measurement, 1 s sample length, 0 sample discard . . . 61
A.8. External Pt100 channel, replaced with a 100 Ω resistor, channel is switched after measurement, 1 s sample length, 32 sample discard . . . 61
A.9. Measured temperature drift, corresponding average voltage per temperature and corresponding standard deviation of offset channel, from -20 °C to 60 °C in steps of 20 °C . . . 62
A.10. Measured temperature drift, corresponding average voltage per temperature and corresponding standard deviation of the thermopile 1 channel, from -20 °C to 60 °C in steps of 20 °C . . . 63
A.11. Measured temperature drift, corresponding average voltage per temperature and corresponding standard deviation of the thermopile 2 channel, from -20 °C to 60 °C in steps of 20 °C . . . 64
A.12. Measured temperature drift, corresponding average voltage per temperature and corresponding standard deviation of the thermopile 3 channel, from -20 °C to 60 °C in steps of 20 °C . . . 65
A.13. Measured temperature drift, corresponding average voltage per temperature and corresponding standard deviation of the Pt100 of thermopile 1 channel, from -20 °C to 60 °C in steps of 20 °C . . . 66
A.14. Measured temperature drift, corresponding average voltage per temperature and corresponding standard deviation of the Pt100 of thermopile 3 channel, from -20 °C to 60 °C in steps of 20 °C . . . 67
A.15. Measured temperature drift, corresponding average voltage per temperature and corresponding standard deviation of the external Pt100 channel, from -20 °C to 60 °C in steps of 20 °C . . . 68

List of Tables

2.1. Filter characteristics of used sensors and their corresponding sensitivities 10

3.1. Channel numbers on the ADC and it's corresponding measurements 14
3.2. Peripheral connection of radiometer with the flight computer 16
3.3. Chosen circuit board layers and their functions 22
3.4. 0 Ω resistor placement options for setting the signal shielding track voltage of the radiometer 24
3.5. Resistor placement options for choosing the grounding concept of the radiometer 25
3.6. Register configuration for the used ADC (ADS1258-ep) 31

4.1. μ_{real} and σ_{real} of offset channel and external Pt100 channel for different measurement configurations 40
4.2. Overview of relevant component groups and their worst case error for analysing the high deviation of measured to expected voltage in the external Pt100 channel 41
4.3. Overview of channel's type of assembly for temperature drift test 44

List of Abbreviations

DLR	Deutsches Zentrum für Luft- und Raumfahrt
NASA	National Aeronautics and Space Administration
JAXA	Japan Aerospace Exploration Agency
FPGA	Field Programmable Gate Array
RTD	Resistance Temperature Detector
NETD	Noise Equivalent Temperature Difference
IPHT	Leibniz Institute for Photonic Technology
ADC	Analog-to-digital converter
Mable-Rad	Mars Atmospheric Boundary Layer Radiometer
SPI	Serial Peripheral Interface
PCB	Printed circuit board
SMD	Surface-mounted device
CPU	Central processing unit
GPIO	general-purpose input/output
CSV	Comma-separated values
TC	Temperature coefficient

1. Introduction

1.1. General Overview

The following bachelor thesis is the result of five months of work at the "Deutsches Zentrum für Luft- und Raumfahrt" (DLR) in the Institute of Planetary Research in Berlin, which offered this thesis topic for students in STEM programs. The objective was to develop electronics and firmware for a prototype three-channel radiometer with application on a Mars weather balloon. A study conducted by Witte et al. [1] in 2022 identified scientific goals how a weather balloon on Mars could contribute to the latest goals of the exploration of the planet. This proposed Mars weather balloon would contribute to the goals "Understand processes of climate of Mars", and "Prepare for human exploration" identified by the Mars Exploration Program Advisory Group (MEPAG) in 2018 [1, p. 2], [2].
Currently, a demonstrator weather balloon is developed at the DLR Institute of Space Systems in Bremen. The radiometer developed in this thesis will be a payload for the balloon system prototype. It is based on previous developed radiometers at the Institute for Planetary Research in Berlin, and will be presented in this thesis.

1.2. Motivation and Objectives

The scientific goals of the proposal balloon mission is to gather weather data from Mars, especially of the planetary boundary layer [1, p. 3]. This atmospheric layer is the connecting layer between the planet's surface and its atmosphere [1, p. 1].
Gathering of weather data is important for improving current martian weather models, like the Mars Weather Research and Forecasting model (MarsWRF, [3]) and the Mars Weather Research and Forecasting model (MRAMS, [4]). Among other applications, these models were used for the Perseverance rover surface and science operation planning by NASA [5]. The gathered data from a weather balloon drifting through the atmospheric boundary layer would contribute to fill a data gap between spatial coverage from global orbiters, and local observations by landers, rovers and drones [1, pp. 17–18].
Currently it is believed, that topographic peaks and larger formations on Mars are leading to turbulence winds, which are a potential hazard for entry and landing of space systems [1, p. 1]. Therefore, previous landings stayed clear of these topographies [1, p. 1], [6]. In the future it will be necessary to land closer to these formations, in order to access further scientifically interesting areas. To enable such landings, a deeper understanding of weather phenomena at those sites is needed, which would be possible in more detail with additional weather data [1, p. 2].

Besides the radiometer, other scientific instruments fly alongside on the weather balloon [1, pp. 4–5]. Namely, these are pressure, temperature and humidity sensors for gathering of general weather data. Additionally, a triad of accelerometers and gyros sense the vertical wind speeds and enable turbulence detection. A nadir-looking camera for event monitoring and post-flight trajectory reconstruction is added to the payload as well. For position tracking and communication, an UHF radio transceiver is part of the flight avionics as well.

1.3. Goal of this Thesis

The DLR Institute of Planetary Research in Berlin is experienced in developing radiometers and contributed such instruments to several space exploration missions. The last launched radiometer was with NASA's InSight Mission to Mars in 2018 [7]. Other missions include the Hayabusa 2 mission to asteroid 162173 Ryugu by JAXA (Japan Aerospace Exploration Agency) [8], which was launched in 2014, and JAXA's upcoming MMX (Martian Moons eXploration) mission [9]. The radiometer developed for this thesis heavily draws from these previous missions for faster development, since these designs are well known and tested.
Notable differences to previous radiometers are the adaption to the size and weight constraints of this Mars weather balloon. Because of that, it was not possible to integrate an active temperature stabilization with the help of a heating system, a passive cooling system is used instead. Additionally, a microcontroller is used as the flight computer for this system, whereas previous radiometers were controlled by a Field Programmable Gate Array (FPGA).
Own contributions are therefore the design of a radiometer with the given design constraints, the development of a driver for the deployed microcontroller and it's adaption to the given avionic system. Also, a development board for programming, testing and measuring of data from the radiometer was developed.
Scientific goals from data collection of this three-channel radiometer are to measure the radiation in different wavelength ranges. These are the wavelengths of 15 µm, in order to derive the average temperature of the air column, 9.35 µm, which correlates to the aerosol (dust) maximum and is a proxy for the opacity of the atmosphere, and 7.94 µm, which is used to derive the surface temperature of Mars [1, pp. 5–6].

1.4. Structure of this Document

The document is divided into five chapters. First a brief overview over the proposed mission and scientific goals is given in Chapter 1. The following theory part in Chapter 2 introduces the necessary background information which will be needed for understanding how all components of the radiometer work. After that, Chapter 3 goes into detail how the radiometer was developed and why specific design decisions were made. Chapter 4 presents and explains all measurements taken to characterize the radiometer and evaluates the results. Finally, a summary and outlook of the work is given and further work is discussed in Chapter 5.

2. Theory and Approach

2.1. Fundamentals of Radiometry

The term radiation refers to the propagation of energy in form of waves through space [10, p. 1]. This includes electromagnetic radiation, such as infrared and visible light, particle radiation, for example alpha radiation, acoustic radiation like sound or seismic waves which are propagating through a medium, and gravitational radiation, but the term radiometry is generally only used to describe electromagnetic radiation [11]. Whenever the term radiation is used here, it refers to electromagnetic radiation.
Radiometry describes and measures electromagnetic radiation [10, p. 1]. To get an overview of the fundamental quantities in radiometry and their relationships, the most important ones are listed below: [10, pp. 8–11]

Radiant Flux Φ is defined as the flow of radiant energy per unit time

$$\Phi = \frac{dQ}{dt} \qquad \left[\mathrm{W} = \frac{\mathrm{J}}{\mathrm{s}}\right] \tag{2.1}$$

where Q is the amount of energy emitted, transmitted, reflected or received per unit time t.
Irradiance E defines the area density of radiant flux

$$E = \frac{d\Phi}{ds_0} \qquad \left[\frac{\mathrm{W}}{\mathrm{m}^2}\right] \tag{2.2}$$

when ds_0 is the surface of an element through which the radiation passes.
Radiant intensity I defines the solid angle density of radiant flux

$$I = \frac{d\Phi}{d\omega} \qquad \left[\frac{\mathrm{W}}{\mathrm{sr}}\right] \tag{2.3}$$

when $d\omega$ is the solid angle of incident radiation.
Finally, radiance B is the radiant flux per unit solid angle and per unit of projected area

$$B = \frac{d^2\Phi}{d\omega\, ds_0} \qquad \left[\frac{\mathrm{W}}{\mathrm{sr}\cdot\mathrm{m}^2}\right] \tag{2.4}$$

For completion and since Eq. 2.4 for radiance B is based on them, Eq. 2.3 for radiant intensity I and Eq. 2.2 are also given, but won't be used in further calculations. In literature, radiance is sometimes also written as L, which is equivalent to B.

2.1.1. Blackbody Radiation

A blackbody is defined as a perfect emitter of electromagnetic radiation [10, p. 63]. Incident radiation pass into it, so it has zero reflectance, and all of this radiation is absorbed, which means it has zero transmittance. This applies for all angles and wavelengths of incoming radiation. Therefore, an ideal blackbody has a reflectance of $R = 0$ and an absorbance of $A = 1$. To the human eye, a blackbody looks perfectly black at room temperature [10, p. 64].
All bodies above a temperature of absolute zero emits radiation - the hotter the body, the more radiation is emitted. The total quantity of this emitted radiant flux from an object at temperature T is expressed by the Stefan-Boltzmann law: [10, p. 63]

$$\Phi_{bb} = \sigma T^4 \tag{2.5}$$

where Φ_{bb} is the emitted radiant flux, σ is the Stefan-Boltzmann constant (not to be confused with the Boltzmann constant) and T the temperature of the body in kelvin. The Stefan-Boltzmann constant is $\sigma = 5.670374 \cdot 10^{-8}\ \mathrm{W \cdot m^{-2} \cdot K^{-4}}$ [12, p. 44].

2.1.2. Greybody Radiation

Blackbodies are a theoretical concept, due to it's perfect absorbance of 1 and reflectance of 0 and don't exist in reality. To understand the concept of greybodies, one has to first introduce the quantity emittance. Emittance ε is the ratio of emitted flux from a surface to the flux which would be emitted by a perfect blackbody emitter at the same temperature [10, p. 132]. This quantity is always between 0 and 1. A blackbody has an emissivity of 1 [10, p. 133].
In general, emissivity is a function of wavelength. If a surface is not a perfect blackbody, but has a constant emittance, which is independent of wavelength, this surface is called a greybody [10, p. 133]. On the other hand, when the emittance of a body depends on wavelength, the surface is called a nonblackbody. In reality, a greybody doesn't have to appear grey like it's name suggests, but the body can appear in another color than black, because not all incoming radiation is absorbed and some is reflected. For a greybody, the previously introduced Stefan-Boltzmann Law changes to: [10, p. 133]

$$\Phi_{gb} = \varepsilon \sigma T^4 \tag{2.6}$$

To give an example of magnitude of ε, white paint has an emittance from 4 µm to 16 µm of 0.9, while black asphalt has one of 0.93 in the same wavelength range [10, p. 134].
The surface of Mars can be seen as a greybody, which emissivity varies, depending on the composition of the surface. Therefore, the concept of greybodies is crucial to understand why care has to be taken when interpreting a measured brightness temperature in terms of the kinetic temperature, as pointed out in Grott et al. paper about the MASCOT radiometer [8, p. 10].

2.1.3. Planck's Law

With the help of the Stefan-Boltzmann law, it is possible to determine the entire emitted radiant flux of a blackbody over all wavelengths. To calculate the emitted radiant flux within certain wavelengths, the usage of Planck's law is needed.
When the temperature of a body changes, the spectral distribution of it's emitted flux is also shifted. This is known as Planck's law, or Planck's blackbody spectral radiation law. It pre-

dicts the spectral radiance B of a blackbody at a given temperature [10, p. 64]. This relation is given by:

$$B(T,\lambda) = \frac{2hc^2}{\lambda^5(e^{\frac{hc}{\lambda kT}} - 1)} \tag{2.7}$$

where h is Planck's constant ($h = 6.626070 \cdot 10^{-34}\ \mathrm{J \cdot Hz^{-1}}$) [12, p. 44], c the speed of light ($c = 2.9979246 \cdot 10^8\ \mathrm{m \cdot s^{-1}}$ [12, p. 44]) and k is Boltzmann's constant with $k = 1.380649 \cdot 10^{-23}\ \mathrm{J \cdot K^{-1}}$ [12, p. 44]. T is the temperature of the object, λ the wavelength of the emitted energy. This law is used for calibration of the radiometer, as well as to calculate the theoretical value of the generated voltage at the thermopiles (see Eq. 2.16).

2.2. Platinum Resistance Thermometers

To determine the temperature of a surface with the measurements taken by a radiometer, one has to know the temperature of the cold junctions of the thermopile. This is done with the help of resistance thermometers, also known as a Resistance Temperature Detector (RTD). As explained by Chris Dames [13, pp. 1782–1785] in more depth, a resistance thermometer is a sensing element, which converts an ambient temperature to an electrical resistance. It is based on the fact that the resistance of a conductor depends on its temperature. This resistance can be measured. For this purpose, the resistor is connected to a current source, and the voltage drop across the resistor can then be measured. With the help of Ohm's law and the known flowing current, the resistance can be calculated. For a Pt100 sensor, as used in this radiometer design, the measured resistance can be converted to a temperature with the help of the Callendar–Van Dusen equation. The equation applies only to platinum resistor thermometers.
For temperatures below 0 °C, this equation is:

$$R(T) = R(0)[1 + A \cdot T + B \cdot T^2 + (T - 100)C \cdot T^3] \tag{2.8}$$

For temperatures above 0 °C, a simplified form can be used:

$$R(T) = R(0)[1 + A \cdot T + B \cdot T^2] \tag{2.9}$$

Here, T is the temperature of the platinum resistor, $R(0)$ is the nominal resistance at 0 °C, A, B and C are universal constants which are determined experimentally. To convert a resistance into a temperature, one has to invert the given formulas. For temperatures above 0 °C, these leads to:

$$T(R) = \frac{-A \pm \sqrt{A^2 - 4B(1 - \frac{R}{R(0)})}}{2B} \tag{2.10}$$

For temperatures below 0 °C, the inversion of the cubic function cannot be performed analytically. Here precalculated values from the manufacturer or an interpolation table should be used.

2.3. Thermopile Sensors

2.3.1. Seebeck Effect

The Seebeck effect is the basic physical principle on which measurements using thermopiles are based (see Paul and Paul for details [14, pp. 659–661]). According to the Seebeck effect a voltage is generated when two different electrical conductor materials form an electrical circuit. These materials have to have different temperatures. The resulting generated voltage can then be calculated by

$$U = \int_{T_1}^{T_2} (S_B(T) - S_A(T))\, dT \tag{2.11}$$

when T_1 and T_2 are the temperatures of the contacts between the materials, S_B and S_A are called Seebeck coefficients. These are material constants with the dimension of $[\mathrm{U} \cdot \mathrm{dT}^{-1}]$ and temperature dependent. If the temperature differences between the two metals are small, it can be assumed that S_A and S_B are constant, and the Eq. can be simplified to

$$U = (S_B - S_A) \cdot (T_2 - T_1) = (S_B - S_A) \cdot \Delta T \tag{2.12}$$

As can be seen, at small temperature differences the generated voltage is proportional to the temperature difference. Different combinations of materials lead to different sensitivities, and in thermopiles a high sensitivity is desired. This way, the thermopile is more sensitive and the SNR is high [10, pp. 187–188].

2.3.2. Thermocouple

A thermocouple is a sensor which converts a temperature difference into electrical energy by means of the Seebeck effect. It consists of two different metals forming an electrical junction. If electromagnetic energy is absorbed by an object, it is converted into heat. Due to the Seebeck effect, this generates a voltage difference proportional to the temperature difference of the two metals. To measure heat temperature, one junction has to be kept at a known reference temperature. This junction is called the passive or cold junction. The other junction, called active or hot junction, changes it's temperature due to changes of the incoming radiation [10, pp. 187–188].
The passive junction has to be protected from incoming radiation, while the active junction exchanges radiation with the outside. Since the sensitivity of an object to radiation is inversely proportional to it's mass, any smaller objects are heated up more quickly by the same amount of energy, the active junction should have low mass. This way, it responds quickly to changes in the flux level, which is desired. This increases the response time of a thermocouple. On the other hand, the passive junction should be as insensitive as possible to outer radiation and temperature changes. To make it insensitive to outer radiation, it has to be shielded from it. To make the passive junction even more insensitive to ambient temperature changes, it should have a high mass and a large area [10, pp. 187–188], and active or passive temperature stabilization can be used to improve sensor performance.
Passive stabilization is discussed in Chapter 3.2.1, where the insensitivity is improved by adding an aluminium body around the used sensors to increase the thermal mass.

2.3.3. Thermopile

Single thermocouples have a relatively low sensitivity. To increase the sensitivity and thus the signal to noise ratio of the measurement, thermocouples are placed in series, which is then called a thermopile. The generated voltages by each thermocouple is added up according to Kirchhoff's law and generates a higher voltage, which is easier to measure. Since all thermocouples are generating approximately the same voltage because of their proximity to each other, the resulting voltage increases roughly linear as a function of the number of used thermocouples [10, p. 188]. The thermopile sensor used in the MARA radiometer for the Hayabusa 2 mission for example, integrated 72 thermocouples with a Seebeck coefficient of 135 µV / K each [8, p. 8].

As shown in Figure 2.1, one possible mechanical design configuration for arranging the thermocouples consists of the two junctions connected in series in multiple points. To insulate the passive junctions, the chamber between hot and cold junction is filled with a noble gas or vacuum. Insulation is needed to increase the insulation of the passive junctions from the active junctions and decrease heat transfer between the two [10, p. 189]. The active junctions, which absorb the radiation, are coated with a black material with low spectral selectivity. This increases the linearity of the sensor. It has contact to an insulating transparent window, which is made of glass or another material. Ideally, this material has zero reflectance over the used spectral range. Additionally, an optical filter is added, which rejects radiation outside the spectral range, in which the thermopile should measure [10, p. 188].

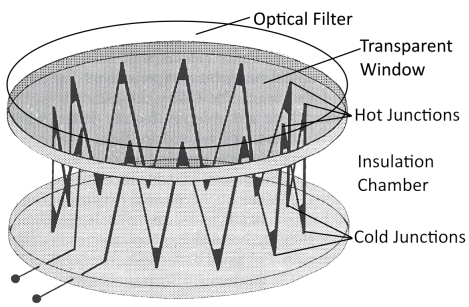


Figure 2.1.: Schematic of the mechanical design of a thermopile, modified from [10, p. 189 Figure 7.6]

A thermopile consists of multiple electrical connections between the hot and cold surface of the sensor. Each connection between the two junctions forms one thermocouple, therefore multiple thermocouples in series are used. This is increasing the sensitivity of the thermopile sensor. The incoming radiation is absorbed by the hot junctions, which are connected to a transparent window. Both junction types are insulated from each other with the insulation chamber. An optical filter is added on top the sensor to reject radiation outside the spectral range.

2.3.4. Measurement Principle

Since the temperature difference between the hot and cold junctions are small in a thermopile, Eq. 2.12 applies and the generated voltage is proportional to the temperature differences at the junctions. As derived by Grott et al. [8, pp. 7–9], the generated voltage of a thermopile sensor

can be calculated as follows:
The amount of radiation received by the sensor is given by

$$\Phi_{in} = \pi \nu \int_{\lambda_1}^{\lambda_2} H(\lambda) \varepsilon B(T, \lambda)\, d\lambda \tag{2.13}$$

where ν is the solid angle (the viewfactor of the sensor), λ_1 and λ_2 are the wavelengths boundaries, $H(\lambda)$ is the instrument throughput, ε is the emissivity of the surface in the field of view, T the temperature of the surface and $B(T, \lambda)$ is the Planck function. $H(\lambda)$ is given by

$$H(\lambda) = \tau(\lambda) \varepsilon_{abs}(\lambda) \tag{2.14}$$

which is the product of absorbing characteristics of the sensing element $\varepsilon_{abs}(\lambda)$ and the filter transmission $\tau(\lambda)$. Since the instrument itself re-radiates at the temperature of the instrument T_{ref}, when reflections inside the sensor are ignored, the net flux on the sensor can be calculated by:

$$\Phi_{net} = \pi \nu \int_{\lambda_1}^{\lambda_2} H(\lambda)(\varepsilon B(T, \lambda) - B(T_{ref}, \lambda))\, d\lambda \tag{2.15}$$

As can be seen, emission from the sensor $B(T_{ref}, \lambda)$ leads to a subtraction from the emitted flux of the surface. Note that no thermal interactions with the filter window have been considered, and it is assumed that the filter will perfectly block radiation outside its transmission window $\tau(\lambda)$.
The generated voltage at a thermopile can now be derived by including the sensor characteristics:

$$U = S A_d \pi \nu \int_{\lambda_1}^{\lambda_2} H(\lambda)(\varepsilon B(T, \lambda) - B(T_{ref}, \lambda))\, d\lambda \tag{2.16}$$

here S is the sensor sensitivity in [U / W] and A_d the absorber size of the sensor.

2.3.5. Brightness Temperature Deviation

To determine the brightness temperature of a surface, the emitted radiated electromagnetic energy is measured by a thermopile. To then derive a brightness temperature from that measurement, the internal temperature of the instrument has to be measured as well. As seen in Eq. 2.12, only a temperature difference between the junctions can be measured by a thermopile. Therefore it is needed to know the temperature of the cold junction. This is achieved by a dedicated temperature sensor, like a Pt100, which is placed close to the cold junction. Each thermopile sensor has therefore an integrated temperature sensor in its housing. In the present radiometer design, there is an additional resistance thermometer in the form of a Pt100 integrated in the aluminium body surrounding the sensors. This measures the temperature of the thermally coupled aluminium body, which acts to stabilize the cold junction temperatures.

As seen in Eq. 2.16, it is possible to calculate the voltage expected by the thermopile. This voltage can be compared to the measured voltage from the radiometer to determine the surface brightness of an object. In practice, it is more useful to calibrate the device and compare the measured data with the calibrated values, rather than the theoretically calculated ones. The reason for this is that there many parameters of the calculation, i.e., the sensor sensitivity S, the absorber size A_d and the instrumentation throughput $H(\lambda)$ which introduce a degree of uncertainty and variation.

2.3.6. NETD and Performance

The Noise Equivalent Temperature Difference, short NETD, is a measure for the thermal sensitivity of an infrared sensor. It is measured in [K]. The higher the NETD, the lower the sensor's resolution of smaller temperature differences. The NETD can be calculated as follows: [15, pp. 1441–1442]

$$NETD(T) = \frac{U_{Noise}(T)}{S(T)} = \frac{U_{Noise}(T)}{\frac{dU(T)}{dT}} \quad [K] \tag{2.17}$$

Here, T is the temperature of the object which is measured, $S(T)$ is the temperature dependant sensitivity, $U(T)$ describes the measured voltage and $U_{Noise}(T)$ is the noise of the measurement. It combines noise from the electronics as well as thermal noise. The NETD is a good measure for comparing different radiometer performances. To given an example of the magnitude of a NETD, the MARA radiometer for the Hayabusa 2 mission has a NETD for it's > 3 µm channel at T > 123 K of 0.1 K [8, p. 7].

2.3.7. IPHT Thermopile Sensors

For the present radiometer design, already chosen sensors for the HP^3 radiometer were used. HP^3 was launched in 2018 as part of the InSight mission [7] and used sensors provided by the Institute of Photonic Technology (IPHT) located in Jena. Mueller et al. describe the calibration of the HP^3 radiometer in detail [16]. The HP^3 radiometer integrates a total of six thermopile sensors, here each sensor type is integrated twice with individual fields of view. Due to size and weight constraints for the Mars balloon radiometer, it was chosen to use each sensor type only once, so in total three sensors are integrated in this design.

Each sensor has the same underlying foundation. These are TS-72 thermopile sensors from the IPHT. Differences are then achieved with choosing a different filter window for each sensor. Due to the different filters, each sensor has then a different sensitivity over the spectral range, and generates a different voltage when observing the same surface. This is illustrated in Table 2.1. The material combination chosen for the hot and cold junction is a pair of Bismuth and Antimony ($Bi_{0.87}Sb_{0.13}$ / Sb), which generates a voltage of 135 µV / K per thermocouple [7, p. 23]. 72 of these thermocouples are combined, so the sensitivity is 9.72 mV / K.

The sensors are sealed using the noble gas krypton, which protects and insulates the junctions from each other. Krypton is chosen because of its relatively high molecular weight, which reduces thermal conductivity and thus increases the temperature difference between hot and cold junctions. This increases the signal-to-noise ratio. Additionally, each sensor has an integrated platinum resistance thermometer (Pt100) with a nominal resistance of 100 Ω, to measure the temperature of the cold junction. The integrated Pt100 is placed as close as possible to the cold junction, and is thermally coupled to it. The three filter windows are different band pass filters. When exposed to an irradiance of 38 W / m^2 they generate a thermal voltage as seen in Table 2.1.

Sensor Number	Filter Range	Generated Voltage
1	8 µm - 14 µm	0.82 mV
2	8 µm - 10 µm	0.37 mV
3	15 µm - 19 µm	0.05 mV

Signal voltage generated for the different thermopile sensors, when exposed to an irradiance of 38 W / m^2.

Table 2.1.: Filter characteristics of used sensors and their corresponding sensitivities

Knowing the sensor response is important when choosing the gain of the signal amplification, as well as interpreting measured results from the radiometer. As shown in Table 2.1, the largest signal will be generated by the 8 µm - 14 µm filter, which will also provide the highest Signal-to-Noise ratio.

2.4. Analog to Digital Conversion

Since the detection of radiation with thermopiles generate an analog signal, this signal has to be digitized. This is done with an analog-to-digital converter (short: ADC). Many different kinds of ADCs exist, here a delta-sigma converter is chosen that has also been used in previous radiometer developments, like the upcoming MMX mission [9]. Because of that, the implementation and behaviour of the ADC is known, and choosing the same ADC contributed to a faster development time and known performance of this radiometer.

In analog-to-digital conversion, there is always a trade off between the precision and the number of measurements per time. The higher the number of measurements per time (also known as the sampling rate), the lower it's measurement uncertainty, and vice versa. For radiometric measurements, it is expected that the measured temperatures are not changing within seconds, since atmospheric temperature changes are generally slow. For this reason the focus is more on precision rather than on a high sampling rate. A low sampling rate of 1 s per channel was chosen, increasing the precision of the measurements and reducing the noise. This increase can be observed in Chapter 4.1.1, where a 0.1 s sample length and 1 s sample length of the same measurement configuration is compared. Since the used ADC doesn't support 0.1 s or higher sample lengths, oversampling on the firmware side was implemented to achieve these rates.

2.5. Signal Path and Conditioning

To increase the resolution of the measured signal, it is important to amplify the measured signal from a thermopile into a range which matches the conversion range of the ADC. As seen in Table 2.1, the generated voltage of a thermopile is small. The analog-to-digital converter has a measuring range of -2.5 V - 2.5 V, so the signal should be amplified by an external instrumentation amplifier, to fit it's range. All thermopiles are measured differentially, therefore the instrumentation amplifier must also operate differentially and amplify this difference by a constant factor.

The signal flow is presented in Figure 2.2. The ADC has an internal multiplexer, which selects the sensor which should be measured. After multiplexing, the signal is routed again to the outside of the ADC and into the external signal amplifier. This amplifier has a constant gain. After that, the amplified signal is routed to the ADC and is digitized.

Care has to be taken not to distort the measured signals and insert non-linearities. Additionally, the amplification circuit should be small and light, adding only the smallest possible number of additional circuit elements, since each additional circuit introduces another source of uncertainty into the measurement chain.

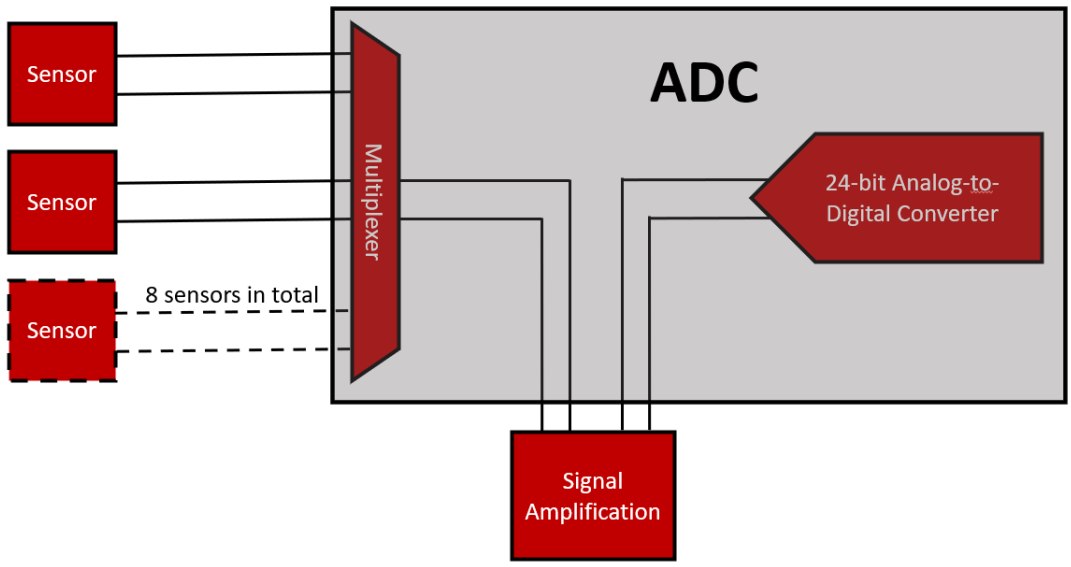


Figure 2.2.: Block diagram of signal flow and signal amplification of the measurements taken in the developed radiometer

Each sensor is measured differentially, the multiplexer in the ADC choses which sensor is measured. The differential signal is routed to the external amplifier, amplified by it and routed back to the ADC, where it is digitized.

For the instrumentation amplifier a circuit with two operational amplifiers was chosen. Important criteria for choosing a fitting operational amplifier for this appliaction are low bias currents and a low offset voltage. The chosen operational amplifier, the ISL28227, is also available as a radiation hardened device and the design can easily be adapted for space application. In this radiometer prototype, the industrial equivalent of the same operational amplifier is used, but it

can be easily replaced with it's corresponding space graded variant.

It should be noted that the research to find this operational amplifier is not part of this thesis and the part comes from the heritage radiometer designs which are used. In Chapter 3.3.1 the implemented amplification circuit is explained in detail.

3. Design and Development

3.1. Design Overview

In this chapter, the design and development of the radiometer for the Mars weather balloon is presented and explained in detail. The radiometer, also called Mable-Rad (standing for Mars Atmospheric Boundary Layer Radiometer), is one payload of the weather balloon besides others. More details about that can be found in its concept paper by Witte et al. [1]. The flight computer consists of one single microcontroller, the ADuCRF101 [17], which controls all of its attached scientific instruments. The radiometer is commanded through the flight computer and all firmware is executed on the microcontroller. The radiometer does not store any code. All EEE (electrical, electronic and electro-mechanical) components of the radiometer are selected, so that there is an equivalent space graded or military graded version, which is suitable for space application. In the prototype developed here commercial equivalents were used because it will not be deployed to space, therefore, more expensive radiation-hardened components are not necessary. This design decision does not affect or alter the performance of the radiometer in comparison to their space graded variants.

The developed radiometer is designed around its analog-digital converter, the ADS1258ep by Texas Instruments. The "ep" in its name stands for enhanced product which has better properties, like a higher temperature range, than its base component ADS1258. Both chips have a similar functionality. In this thesis, the name ADS1258 will be further used. The ADC is available in a relatively small package, a HTQFP 48 pin package [18, pp. 50–52], which is around 10 mm x 10 mm big. This allows a good degree of miniaturisation, but is still suitable for hand soldering. That way, changes of the prototype are easier to make in workshops of the institute. The chosen ADC has 16 single-ended channels which are multiplexed before the measurements are taken. Differential measuring, which is used here, reduces the amount of available channels to 8. The designed radiometer houses three thermopile sensors, each measuring flux in a different wavelength range. Each sensor has one thermopile and one resistance thermometer Pt100 to measure the temperature of its cold junction. An additional external offset circuit is used to correct for electrical offsets when a measurement is taken. With this additional offset measurement, the uncertainty of the radiometer measurements can be increased. A resistance thermometer, integrated into the aluminium body surrounding the sensors is also measured by the ADC. In total, 8 differential measurements are performed, which occupy all 16 channels of the used ADC. An overview of all channels is given in Table 3.1. Here, the left column shows the combination of channels used by the ADC. The column on the right describes the corresponding measurement. It becomes visible that the channel assignment is set by the PCB layout, therefore, matching thermopile and Pt100 channels are not always situated next to each other. This does not affect measurements. Only another assignment must be made in the firmware when matching the channel number combination with its measurement function.

A system overview is given in Figure 3.1, which shows all component groups, their connec-

tions and their used power rails. For presentational reasons, the visible connections do not show the real number of electrical connections. The measurements are all done differentially, therefore two connections are needed for each measurement. The signal amplifier has a total of four connections: two inputs and two outputs. The communication between ADC and microcontroller takes place via the Serial Peripheral Interface (SPI) and additional signal lines. The peripheral connection can be seen in more detail in Table 3.2 in the next sub chapter.

Channel Number Combination on ADC	Description
0, 1	Thermopile 1
2, 3	Pt100 of Thermopile 3
4, 5	Thermopile 3
6, 7	External Pt100
8, 9	External Offset
10, 11	Thermopile 2
12, 13	Pt100 of Thermopile 2
14, 15	Pt100 of Thermopile 1

Table 3.1.: Channel numbers on the ADC and it's corresponding measurements

The radiometer is powered by two external supply voltages coming from the flight computer: 3.3 V and 5 V. The 3.3 V power rail only supplies the digital core of the ADC, whereas the 5 V power rail supplies the analog part. In more detail, all operational amplifiers, the voltage reference and the analog core of the ADC are powered by this 5 V rail. To improve the voltage stability of both power rails, each is filtered by a passive LC filter before being available as a voltage for the radiometer. In addition, the 5 V rail powers a voltage reference, which generates a precise and stable 2.5 V. Furthermore, this reference is used across several parts of the radiometer. For digitization, this voltage is used by the ADC for comparing it with the voltage of the measurement. The ADC measures a voltage and returns a number proportional to the voltage of the voltage reference. Therefore, the voltage reference has to be very stable and precise, since in the later calculation a constant 2.5 V of the voltage reference are assumed. Another task of the voltage reference is to provide a bias voltage to the thermopiles and provide a measurement current for the Pt100. These circuits need to have a voltage level for it's bias level, here it is best practice to use the same voltage the ADC is comparing the measurement with. This is presented in Chapter 3.3.2 about the used Pt100 circuitry in more detail.

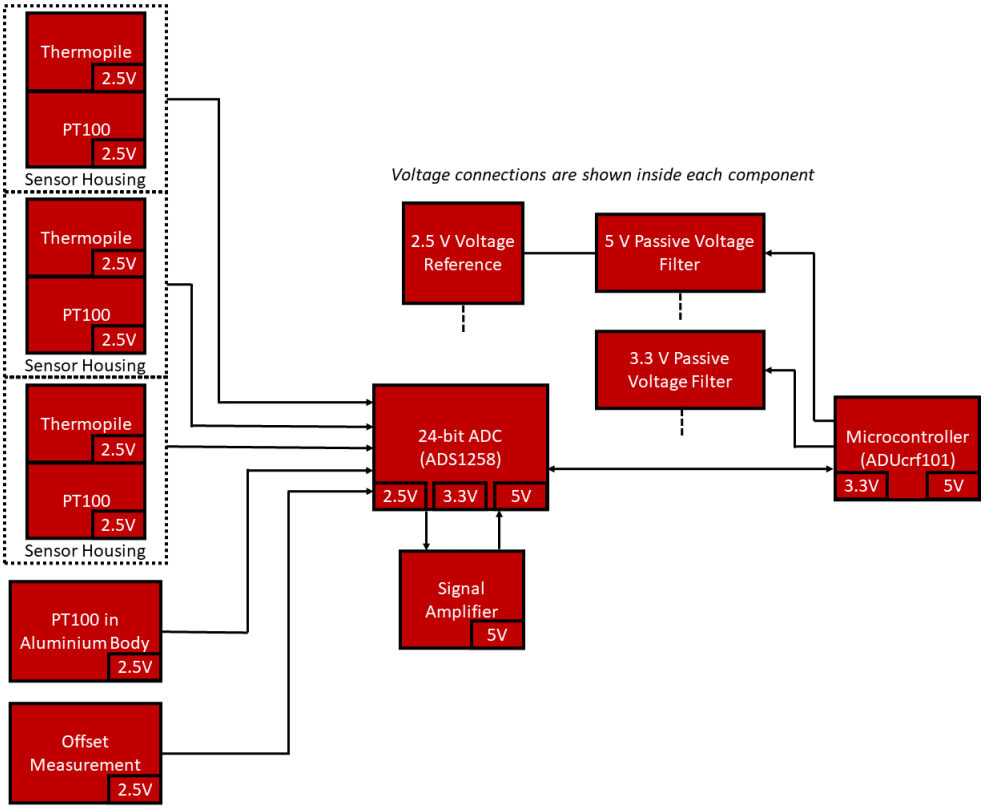


Figure 3.1.: System overview of the radiometer with its component groups and corresponding power rails

The ADC is the main component of the radiometer and is in the center of the diagram. It connects to the measuring probes. This includes three sensors, the Pt100 integrated in the aluminium body and the external offset measurement, visible on the left. Below, the external signal amplifier is represented and connected to the ADC, which handles to amplification of the measured variable. Above, there are the three different voltage levels: a 2.5 V voltage reference generated by the 5 V rail, passive filtered 3.3 V for the digital core and passive filtered 5 V for the analog core. On the right, the ADC communicates directly with the microcontroller, which is the flight computer of the weather balloon.

3.1.1. Electrical Interface

The radiometer is connected with a 12-pin connector to the flight computer. An overview over the connections is given in Table 3.2.

Pin Number	Description	Type, seen from Radiometer
1	3.3 V	Power Input
2	Reset	Digital Input
3	Ground	-
4	SPI Clock	Digital Input
5	Ground	-
6	SPI Data In	Digital Input
7	Ground	-
8	SPI Data Out	Digital Output
9	Data Ready	Digital Output
10	Start	Digital Input
11	SPI Chip Select	Digital Input
12	5 V	Power Input

The radiometer is connected to the flight computer via a 12-pin ribbon cable. Mainly controlled by a SPI interface, it also contains several digital control pins. All connections shown here are seen from the radiometers perspective, which has a male connector.

Table 3.2.: Peripheral connection of radiometer with the flight computer

Because of the mechanical design of the used connector, a Molex Picoblade 12-pin through-hole connector, it cannot be plugged in the wrong way. This increases safety when connecting the radiometer to the flight computer, since the radiometer itself does not have any form of reverse polarity protection. In order to save space in the design, the components are connected via a matching ribbon cable, which is lightweight and flat. For a slightly better connection of the ground planes which are on both sides of the circuit board, a through-hole connector is used instead of a surface-mounted device (SMD) connector. This way, the incoming ground connections are directly connected to both sides of the circuit board, because of it's through-hole mounting. As it can be seen in Table 3.2, multiple ground connections are used. These are placed next to digital SPI pins with a high data rate. This way, the ground pins improve shielding of these digital data pins.

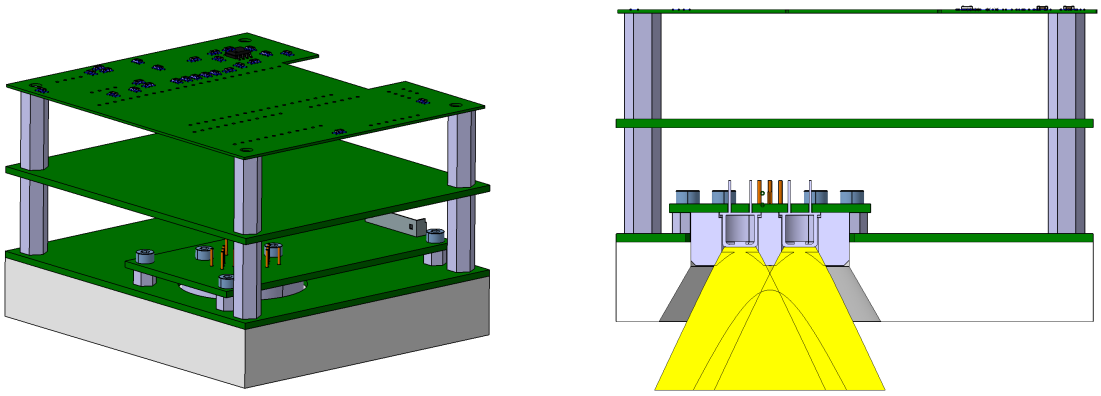
3.2. Mechanical Design

In the beginning of the development phase, the mechanical design was decided in close co-ordination with the engineering team at the Institute of Space Systems in Bremen, since they were in charge of designing the flight avionics system and the Mars weather balloon. The size constraints were predefined by the outline of the already existing mechanical concept. At this point of development, the flight avionics consist of a stack of printed circuit boards at different platforms: There is one battery platform, one power managing platform, one board computer platform and a nadir-looking camera platform. Because the radiometer has to be nadir-looking as well and should point towards the surface of Mars, it needs to be situated on the lowest platform of the balloon. Therefore, it was decided that the radiometer should sit next to the nadir-

looking camera module.

To fit the radiometer next to the camera module, a maximum size of 70 mm x 30 mm was available. The connector with it's ribbon cable is placed on the smaller 30 mm side and points away from the PCB (compare Figure 3.6a).

For the connection between the radiometer and the avionic platform, it was chosen to use four corner-based M2.5 bolts. For optimal weight distribution across the bolts, they are placed symmetrically to the PCB center in each corner. Figure 3.2 gives an overview of the system design and how the radiometer is integrated as a payload of the Mars balloon.



(a) Isometric rendering (b) Side view

Figure 3.2.: System design and integration of the radiometer in the weather balloon
The electronics of the weather balloon consist of stacked printed-circuit boards. The lowest platform contains the nadir-looking radiometer. Below this platform a foam insulation is attached, in which the sensors and the aluminium body are integrated. The other platforms are not shown in detail here, only the radiometer integration is highlighted. The side view in 3.2b highlights the view angle of each sensor in yellow. Each of the three sensors has a similar view angle in a different spot, which results in the seen view angle of the radiometer.

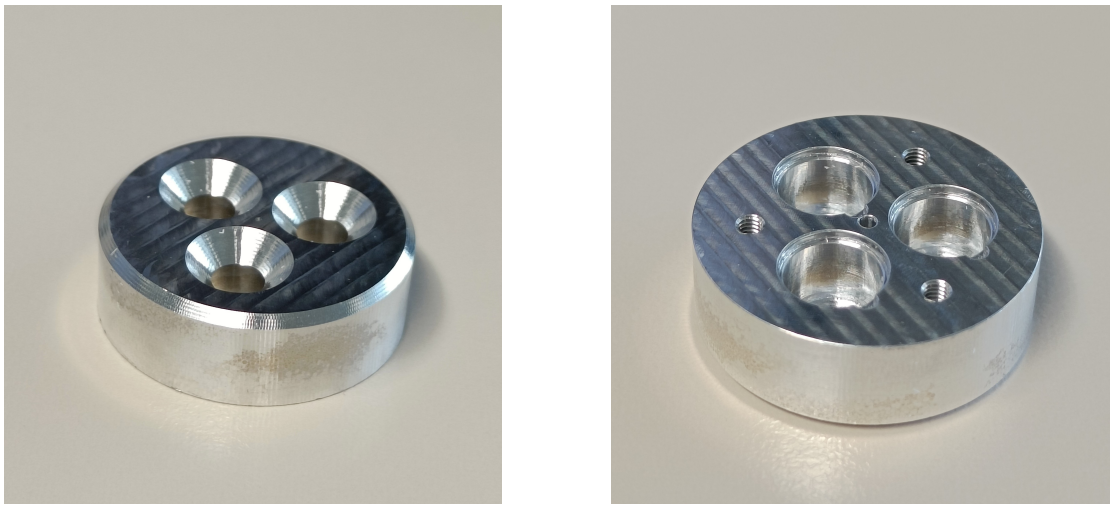
3.2.1. Temperature Stabilization of the Sensor Heads

For best results of radiometric measurements, it is important to have a stable temperature at the cold junctions of the thermopiles. Previous radiometer design like the the HP^3 radiometer in 2018 with NASA's InSight mission have an active temperature stabilization with the help of an integrated electrical powered heating system next to it's sensor heads [16]. This requires the availability of additional electrical energy, as well as more space. For the radiometer design presented in this thesis, these resources were not available and the engineering budgets were limited. This is due to the fact that the Mars weather balloon is only powered by a battery and does not have the possibility to recharge energy, for example with the help of solar panels [1]. Therefore, saving energy is critical for the radiometer. Because of that, it was chosen to rely solely on a passive system for the sensor head temperature stabilization instead.

Due to this design decision, the thermal inertia of the sensors housing becomes important. If the cold junction in a sensor responds too quickly to a change in environmental temperature, the measurement may become unstable and imprecise. To counter these effects, it is possible to stabilize the cold junctions by adding more mass to it, since objects with a higher mass re-

spond slower to changes in temperature. In this presented radiometer design, this is done with the help of an aluminium body, which is thermally connected to each sensor housing. Additionally, each sensor is placed symmetrically with the same distance to each other, in order to evenly distribute the heat between each them. This increases thermal stability.

The aluminium body itself is designed symmetrical to it's center, as it can be seen in Figure 3.3. It is connected via three M2.5 bolts directly to the circuit board of the radiometer, and houses the sensors. The sensors are then connected to the aluminium body with a heat-conducting paste. The paste further increases thermal conductivity between the sensor heads and the aluminium body. Inside this aluminium body, a cutout is integrated which contains the Pt100 responsible for measuring the temperature of the aluminum body itself. In order to attach it, the Pt100 is glued into the aluminium body with the help of a thermal paste. It should be noted that the design of the aluminium body was made by Matthias Grott from DLR and is not my own contribution to this project.



(a) Top Side (b) Bottom Side

Figure 3.3.: View of the aluminium body
The aluminium body sits on top of the thermopile sensors and is thermally coupled to their housings. This helps to stabilize the thermopiles and reduces measurement uncertainty. Connected directly to the radiometer circuit board, a Pt100 is glued inside the aluminium body to measure it's temperature. The surface is flat, the structure seen in the pictures comes from the manufacturing process.

3.3. Schematics Design

The schematics design for this radiometer heavily relies on previous developed radiometers, in particular the radiometer for the upcoming MMX Mission by JAXA [9], and the HP^3 radiometer for the InSight mission [7]. In this chapter, the most important aspects of the schematic design for this development are presented in detail, as well as the implemented testing strategy, which is integrated in the schematic design. All schematics are appended and can be found in the appendix in Figure A.3.1.

3.3.1. Instrumentation Amplifier

The external signal conditioning of the measured signal is done by an instrumentation amplifier. Instrumentation amplifiers are characterized by their high input impedance, their differential amplification and low output impedance [19, pp. 1–2]. Their key components are operational amplifiers. Several different designs of instrumentation amplifiers exist, mainly with three operational amplifiers, but designs with only two operational amplifiers are possible as well. For this application case, an instrumentation amplifier configuration with two operational amplifiers was chosen in order to save space. For the operational amplifiers, the ISL28227 [20] were chosen due to their low input bias currents (typical 1 nA), low offset voltage (typical 10 µV) and low offset voltage drift (typical 0.1 µV / °C). Additionally, this operational amplifier exists as a space graded footprint equivalent and can be directly replaced in the future.

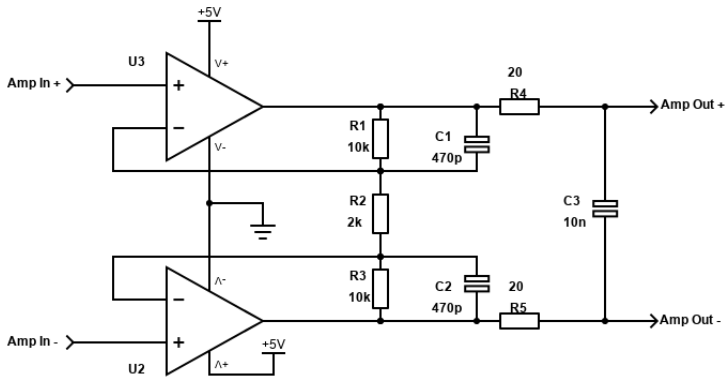


Figure 3.4.: Schematic of the external signal conditioning circuit
The signal amplifier is situated between the ADC's multiplexer which selects the sensors and the analog-to-digital conversion inside the ADC. Before the signal amplifier, the signal is multiplexed through the internal multiplexer of the ADC. The amplifier has a fixed gain of 11 and its common mode input and output voltage is 2.5 V. For the operational amplifier the ISL28227 was chosen [20].

Input signals shown in Figure 3.4 are routed from the multiplexer of the ADC. This routing can be seen in more in detail in the previous chapter, Figure 2.2.

C1 and C2 are stabilization capacitors for the corresponding operational amplifiers. They optimise it's feedback response for high frequencies. Capacitor C3 filters the differential output signal and contributes to it's stabilization. Resistors R4 and R5 further stabilize the operational amplifiers with their current-limiting properties. They also help decoupling the amplification circuit from the traces which have to be driven by it.
R1, R2 and R3 determine the gain of the amplifier. R1 and R3 must have the same resistance. The gain of the amplifier is then given by:

$$G = 1 + 2 \cdot \frac{R1}{R2} \tag{3.1}$$

In this configuration and for this radiometer design the gain of the amplification is 11.

When choosing the electrical components, it is important to chose high precision resistors for the gain determining resistors R1, R2 and R3, which are used in this radiometer design.

3.3.2. Pt100

The developed radiometer measures a total of four individual Pt100. A Pt100, like any other RTD, requires additional circuity to measure it's resistance, which is presented and explained here. Since each Pt100 has similar functionality, the presented circuit is used for all Pt100s across the radiometer. The aim of this circuit is to get a precise measurement of the Pt100's resistance. Therefore, on one end of this resistor a buffered 2.5 V is applied, which is coming from the voltage reference. The Pt100 is then connected via a 47 kΩ resistor to ground. This 2.5 V voltage source is forming a current source together with the 47 kΩ resistor and providing a nearly constant current, since $RT1 << R3$.

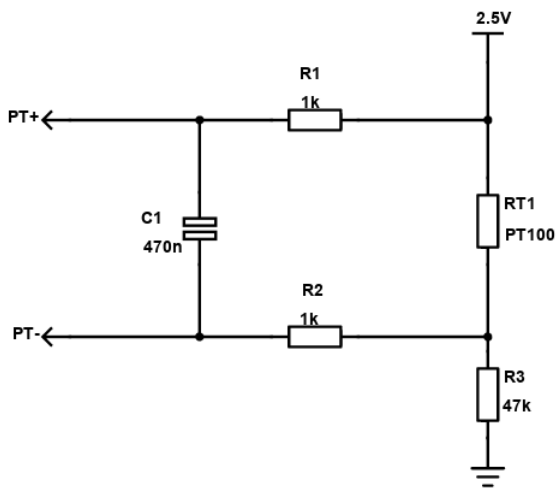


Figure 3.5.: Schematic of Pt100 measuring circuit

The resistors R1 and R2 form together with C1 a RC low-pass filter, which has a time constant of:

$$\tau = RC = R1 \cdot C1 = 1\ \text{k}\Omega \cdot 470\ \text{nF} = 0.47\ \text{ms} \tag{3.2}$$

For calculating the resistance of RT1, the voltage drop across it is measured. Ohm's law is used and for that current flow through the Pt100 needs to be known. The resistance of RT1 is changing with the ambient temperature, but since $RT1 << R3$, it can be assumed that RT1 is constant with 100 Ω, and any change in it's resistance will not affect the amount of current flowing through it. A temperature change of 25 °C leads to a change in resistance of $\Delta R \approx$ 10 Ω, therefore the relative error is:

$$\frac{\Delta R}{R_{all}} = \frac{10\ \Omega}{47.1\ \text{k}\Omega} \approx 0.02\% \tag{3.3}$$

This amount of error is negligible.

The flowing current through RT1 is calculated as follows:

$$I_{all} \approx 47.1\ \text{k}\Omega \tag{3.4}$$

$$I_{Pt100} = I_{all} = \frac{U_{all}}{R_{all}} = \frac{2.5\ V}{47.1\ \text{k}\Omega} \approx 53.08\ \mu\text{A} \tag{3.5}$$

The measured resistance can then be calculated with:

$$R_{Pt100}(U_{Pt100}) = \frac{U_{Pt100}}{I_{Pt100}} = \frac{U_{Pt100}}{53.08\ \mu\text{A}} \tag{3.6}$$

With the help of the Callendar–Van Dusen equation, presented in Chapter 2.2, the temperature of this resistance can then be derived.

When choosing the electrical components for this circuit, it is especially important to choose a precision resistor for R3. This resistor determines the current flow throw the Pt100, so it should be precise, otherwise equation 3.5 cannot be assumed. Additionally, it is important that this resistor has a low temperature drift, since it is only wanted to measure the temperature drift of the Pt100 and not of the current-limiting resistor.

3.3.3. Implementation of Testpoints

During the schematics design phase, a testing strategy was created and later implemented in the circuit board as testpoints. This enables easy testing of the circuit. Testpoints, later realized as vias on the printed circuit board, are added across different circuit sections. Before and after the amplification circuit testpoints are integrated. This way, the amplification circuit can be verified and tested easily. Additionally, the voltage filtering circuit between the board's connector and it's power rails are extended with testpoints. This allows the voltage filtering circuit to be verified. Each measured signal by the ADC is also connected to a testpoint, these can be used to control the measured voltage at the ADC with the voltage measured by an external measuring instrument. These measured values should be be in the same order of magnitude of the measured one by the ADC. Other significant points to measure are the signal lines between ADC and microcontroller, but these are available at the connector of the radiometer. To measure these signals more easily, the testpoints are integrated on the development board, which will be presented in Chapter 3.7 detail. It simulates the flight computer in order for easy testing of the radiometer, therefore, it has access to all signal lines of the radiometer, which are accessible on its PCB for testing.

3.4. Circuit Board Design

For the designed circuit board, it was decided to use a conventional rigid four layer PCB. In more detail, a FR4 - TG 150 PCB with a total thickness of 1.55 mm and a copper thickness of 35 µm, manufactured with an ENIG surface finish was chosen. These are comparatively cheap to manufacture and therefore suitable for a prototype development. The electrical components were not assembled by the PCB manufacturer, but rather by technical personnel at the DLR. This way, it was possible to create a custom assembly procedure and to assemble several component groups step by step. Each component group was then tested independently. This made troubleshooting easier. This accelerated the efficiency of the development phase. The assembly procedure is presented in Chapter 3.6 in more detail.

Regarding the grounding concept, it was chosen to use a shared ground plane for both digital and analog signals. The four layer composition consists of these layers:

Layer	Description
Outer Top Layer	Signal Layer
Inner Top Layer	Ground Layer
Inner Bottom Layer	Shared Power and Ground Layer
Outer Bottom Layer	Signal Layer

Table 3.3.: Chosen circuit board layers and their functions

Each designed circuit board layer can be found in the appendix from A.3.2 to A.3.5.

All signal traces for measurements are routed on the outer top layer, as directly as possible towards the ADC. This layer is where the ADC is located as well. Therefore, this layer contains all critical analog signal paths which require a high degree of shielding. Because of that, the inner top layer, which is the layer below, was chosen as a ground layer. This helps reducing electromagnetic interference and helps to shield the signal traces by providing a better current-return path for the signals.

The inner bottom layer distributes the two different voltages, 3.3 V and 5 V, across the circuit board. Since these voltages are only needed in certain areas of the circuit board, this layer is not only filled with power planes, but also with ground planes.

The outer bottom layer is again a signal layer, which contains the external amplification circuit and the passive voltage filtering of the power rails. Surface-mounted devices are placed on both outer top and bottom layers, due to the small size of the radiometer and the lack of space.

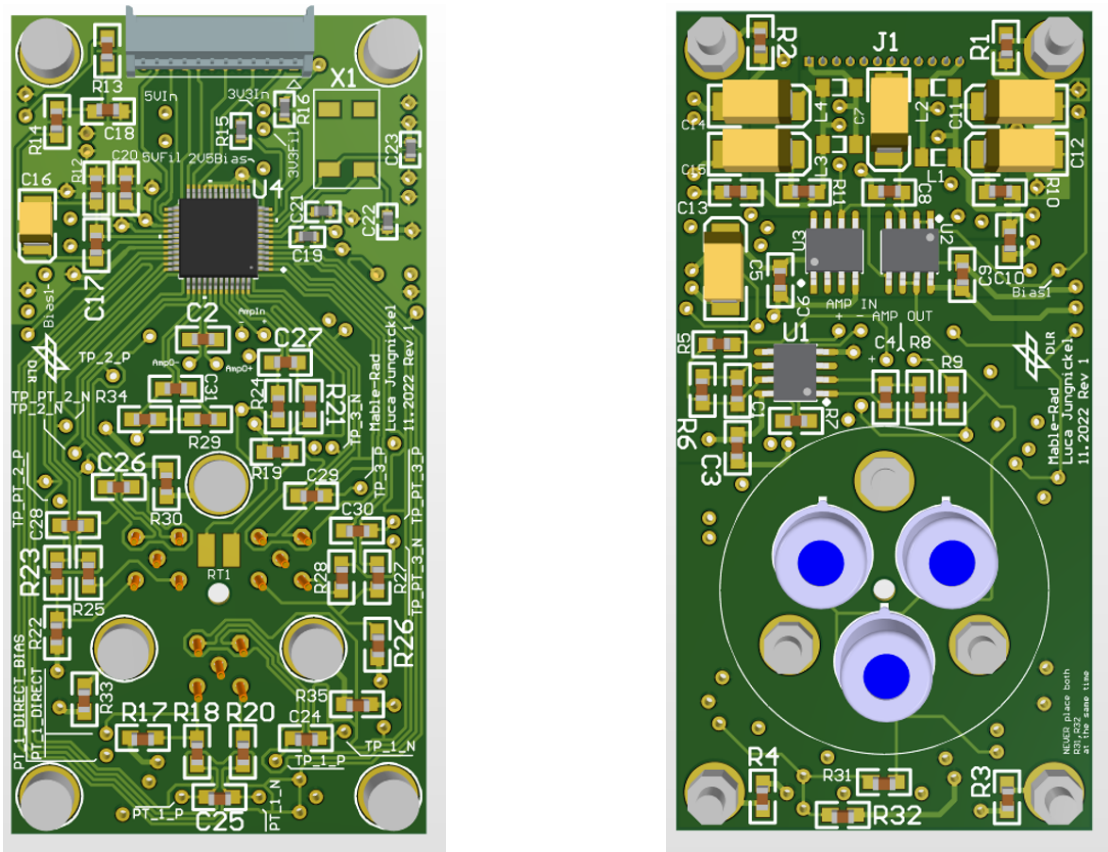
The manufactured prototype of the radiometer is shown as a computer generated preview with component placements in Figure 3.6.

3.4.1. Handling of Sensitive Traces

As already pointed out briefly, some traces need special attention when designing the circuit board. The traces for all thermopiles to the ADC, and for all Pt100 to the ADC were regarded as sensitive and special care was taken while designing this circuit board. For each measurement it was chosen to perform a differential measurement. These traces are laid out as differential pairs. This way, each pair has a constant distance of 0.2 mm while the width of a trace is set to 0.28 mm. The differential pair distance of 0.2 mm is based on the manufacturing capabilities of the PCB manufacturer and chosen to be as small as possible. Furthermore, care was taken to route a differential pair with the same length for each trace. This way, each trace has approximately the same impedance which influences the measured signal similarly to the other trace in this pair.

Another point to increase shielding of these sensitive traces is to shield each differential pair with a shielding track next to the it. The shielding tracks then surround the differential pair where space permits it. As it will be pointed out in Chapter 3.4.2, the shielding tracks are all on one potential and can be tied to ground, to 2.5 V or left floating. This is a placement option for later development and testing. The shielding of this design is shown in more detail in Figure 3.7.

To further minimize disturbances to the measurement signal, the traces from the source to the ADC were routed as directly as possible. Care was taken to not use vias, which would route the signal through the circuit board and add more interference. Finally, the clearance between



(a) Top Side

(b) Bottom Side

Figure 3.6.: Computer generated view of the radiometer
Top side and bottom side computer generated view of the developed radiometer, already assembled with components. The real radiometer looks similar, differences are the assembly of the crystal clock X1, the external Pt100 at the top side at RT1 and the aluminium body, which will surround the sensors on the bottom side. These reference images are used by technical personnel at DLR in order to assemble the PCB.

a differential pair of traces to the next one was maximized. This further decreases inducting capabilities and crosstalk between the different pairs of traces.

3.4.2. Grounding and Shielding Options

Two placement options have been added for this prototype radiometer. This allows for an optional configuration of two aspects with different placements of 0 Ω resistors: the voltage level of the signal-shielding track and the grounding concept of the whole radiometer.

For the signal shielding track there are three different placement options available: Tying it to ground level, to bias level (2.5 V) or leaving them floating. This can be set with the resistors R31 and R32 as follows:

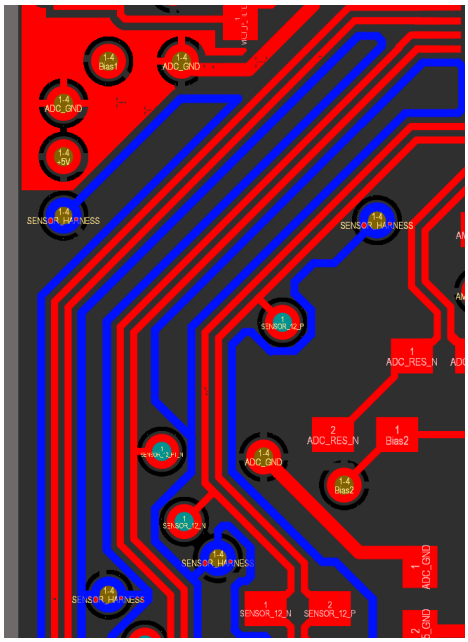


Figure 3.7.: Detail of circuit board trace design with three differential traces, surrounded by shielding traces

Each measurement is taken differentially and routed directly to the ADC, which is located to the upper-right. The differential traces are next to each other, they are surrounded by a shielding trace, which are highlighted in blue for better visibility. Therefore, three to four traces are visible here, which corresponds to one measurement. Each differential pair is laid out in a way that it has a big distance to the next differential pair. All shielding traces are connected to each other and are on the same potential.

Signal Shielding Track Voltage Option	Placement of Resistor
Ground (0 V)	R32
Bias (2.5 V)	R31
Floating	do not place either of the resistors

Table 3.4.: 0 Ω resistor placement options for setting the signal shielding track voltage of the radiometer

At this point of development a system-grounding concept of the weather balloon was yet to be decided. Therefore, the radiometer is designed to support all mounting holes as either floating or grounded. This can be set by assembling 0 Ω resistors for grounding the corresponding mounting hole. If it is wished to leave them floating, the resistors should not placed.

Mounting Hole	Placement of Resistor
MH1 (upper-left corner of PCB)	R1
MH2 (upper-right corner of PCB)	R2
MH3 (lower-left corner of PCB)	R3
MH4 (lower-right corner of PCB)	R4
MH5 (aluminium body, upper hole)	R30
MH6 (aluminium body, lower-left hole)	R33
MH7 (aluminium body, lower-right hole)	R35

Table 3.5.: Resistor placement options for choosing the grounding concept of the radiometer

3.5. Firmware Design

3.5.1. Overview

An integral part of the development of this radiometer is not only to develop and layout the electronics, but to develop the firmware for it as well. The purpose of the firmware is to control the radiometer, its startup, its measuring routine and to provide an interface for the flight computer to interact with. The control between the flight computer and the radiometer is primarily handled via the SPI interface, as well as with additional digital control pins. An overview about the hardware interface is given in Table 3.2. The software interface of the radiometer should be as small, easy and safe as possible to use for other engineers and software developers, on which focus was laid in this development.

For the Mars weather balloon, it was chosen to use the microcontroller ADuCRF101 by Analog Devices as a flight computer. This features an ARM Cortex-M3 processor at the core, as well as an integrated RF transceiver, which supports transceiving data in the ISM radio band of 431 MHz to 464 MHz [17]. The flight computer controls multiple different scientific instruments, like pressure, temperature, humidity sensors and accelerometers [1]. Currently the flight computer is not a real-time operating system but a bare-metal system. Therefore, scheduling or multi-threading is not supported at the moment. Instead, data from each scientific instrument is polled sequentially and each poll should only last a predefined time. At the moment every driver of a instrument needs to ensure that by itself, since the microcontroller has no possibility to intervene due to the lack of a real-time operating system. Because of that, special care was taken to prevent the developed firmware from being blocked. This is explained in more detail in Chapter 3.5.6 about the handling of errors and timeouts.

The development of the firmware was done in the programming language C++. The firmware has to be easily reconfigurable via static code changes, as well as maintainable and follows DLR's internal C++ coding standard for avionics systems. It cannot be reconfigured while the flight computer is running, and therefore provides safety against unwanted reconfiguration. For the design of the software, the principle of object orientated encapsulation was followed. Therefore, two classes were created: *Ads1258*, the driver for the ADC and *Rad*, an extension of the ADS1258 driver with custom configuration and measurement procedure.

Additionally, two integration tests were created in order to test the radiometer. Both integration tests are directly flashed to the flight computer, either to the one used during flight operations, or to the development board of the radiometer. The first and smaller test, called *mable-rad-one-channel-test*, sets up the radiometer and continuously measures one channel. It formats the output as a CSV (comma-separated values) table, which can be directly exported and viewed by a spreadsheet program. The sample length, the number of measurements and the measured channel can be configured before the test case is flashed to the microcontroller.

The second test case, named *mable-rad-complete-test*, resembles the complete measurement procedure taken by the Mars weather balloon during the mission. Here, each channel is measured with a duration of one second, then the driver switches to the next channel. If all 8 channels are measured once, the measured data is returned to the flight computer together with corresponding timestamps and other instruments can be polled. In this test case, it is only possible to set an amount of repetitions of this measurement routine. Configuration of the measurement routine itself is not desired since any change here could alter the measurement uncertainty of the radiometer.

3.5.2. Internal Oversampling

In order to explain the upcoming algorithms, the procedure how sampling in the ADC works must be explained first.

The radiometer performs oversampling at different stages - internally and externally. Internal oversampling is done by the ADC itself, external oversampling is performed by the flight computer.

Internal oversampling of the ADC is shown in Figure 3.8. The ADC is set to oversample 64 samples internally, which is the maximum amount it can oversample. Additionally, a 384 µs delay is set to be performed after each channel switch. Since the ADC is set to chop the measurement, an inverse measurement has to be performed. The ADC starts with switching to the channel to be measured, then it pauses for 384 µs. After this 64 samples are taken and averaged. When this is completed, the same procedure will be repeated, but with inverting the channel to be measured for chopping. These measurements will be averaged again with the previously taken measurements. The whole measurement takes around 2 ms, the result of this measurement is returned to the flight computer. From the view of the firmware, this is regarded is one sample. The firmware itself performs oversampling with these samples, which will be presented in the upcoming algorithm part.

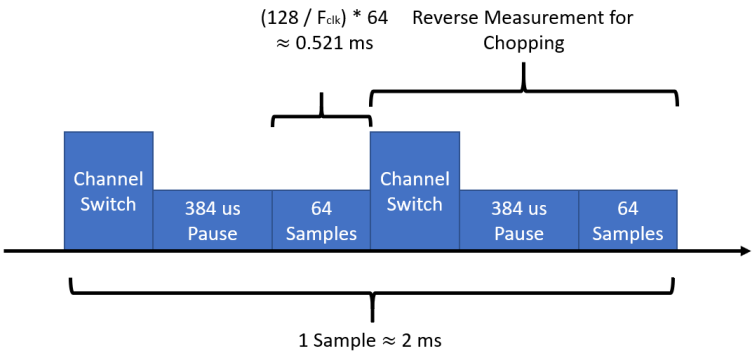


Figure 3.8.: Composition of internal oversampling of the ADC

3.5.3. Measurement Procedure

The following flowcharts show the measuring sequence of the radiometer and the algorithms used in this firmware. Their procedures and the occurrence of possible errors are shown here.

At the beginning the radiometer is initiated with an initialization routine, called *Rad::init*. This writes the register settings shown in Table 3.6 to the ADC and confirms them. When the radiometer driver is initiated, it initializes the ADC. First, it is being reset via a digital pin, then all registers are read and checked if they contain the default values. If successful, the driver writes the custom register settings to the ADC, which is then read back and checked. This procedure returns whether it was successful. This routine is shown in Figure 3.9.

After this initialization phase, data from the radiometer can be polled by calling the *Rad::measure* function, whose routine is shown in Figure 3.10. This procedure oversamples each channel of the radiometer sequentially. The function *Rad::measure* is the main function of the radiometer

and handles its measurement procedure. First, it switches to the channel which should be read and checks if the switch was successful. If not, the procedure immediately aborts, otherwise a predefined number of samples x are fetched and discarded (called sample discard). This number x is configured such that signals are stable after switching of a channel. x depends on the time constant of the low pass filter and should not be reconfigured. It is checked again if an error occurred and eventually the function aborts. After this sample discard, the set channel is oversampled a predefined (and in the flowchart unspecified) number of times. If no error occurred, this procedure is repeated for the next channels until all eight channels are measured. Then, all samples are returned with corresponding timestamps to the calling routine at once.

The function *Rad::measure* relies on two other subfunctions, *Ads1258::getSample* (Figure 3.11) and *Ads1258::getDataAndOversample* (Figure 3.12).

The sample function *Ads1258::getSample* retrieves one sample from the ADC, here a timeout is implemented so it cannot block the central processing unit (CPU) for too long. First, it activates the "Chip Select" pin of the SPI interface, then it polls the "Data Ready" pin until it is active or an timeout occurred. If a timeout occurred, the "Chip Select" pin is deactivated and the procedure aborts with an error. If the "Data Ready" displays a valid sample, this sample is read from the ADC and the "Chip Select" pin is then deactivated. The sample is returned to the function caller and marked as successful.

The oversample algorithm *Ads1258::getDataAndOversample* performs oversampling of a channel. Additionally, this function takes a timestamp before the measurement is taken, which is returned besides the sample. First, it saves a timestamp and starts sampling. The function takes a number of *numTotalSamples* samples, given as a function parameter. It holds a counter variable X, which contains the current number of samples which were already taken. On every execution of the loop, the counter variable X is incremented. The function takes a sample with the *Ads1258::getSample* function and adds it to the *allSamples* variable. This loop is repeated until all samples are collected successfully. Otherwise, if an error occurs, the function immediately aborts. If no error occurred, the average is calculated by dividing the two variables, and the sample and timestamp are returned.

It should be noted that for a clearer representation of the flow charts potential occurring SPI read or write errors are not shown in the flowcharts. If one of these errors occur, the algorithms perform necessary recover steps, like deactivating the "Chip Select" pin in Figure 3.11, and abort with an error.

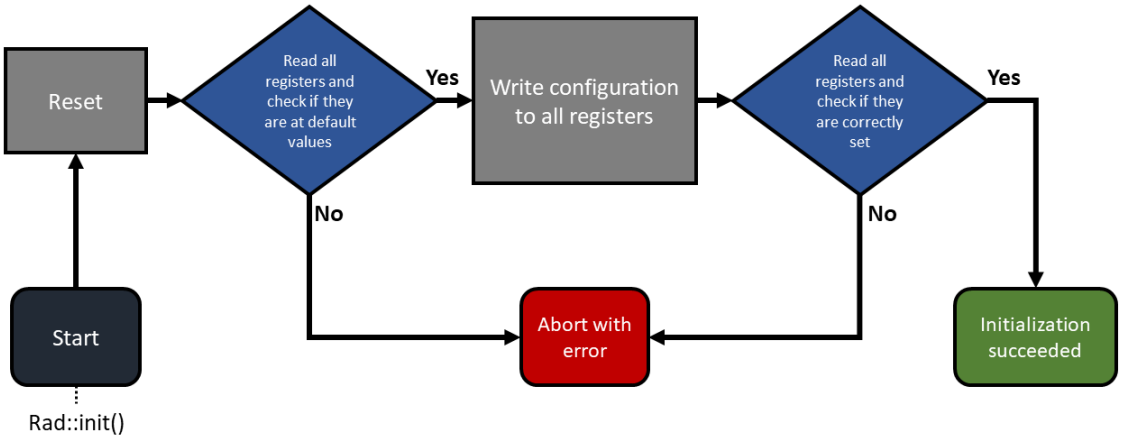


Figure 3.9.: Flowchart of function Rad::init in radiometer driver

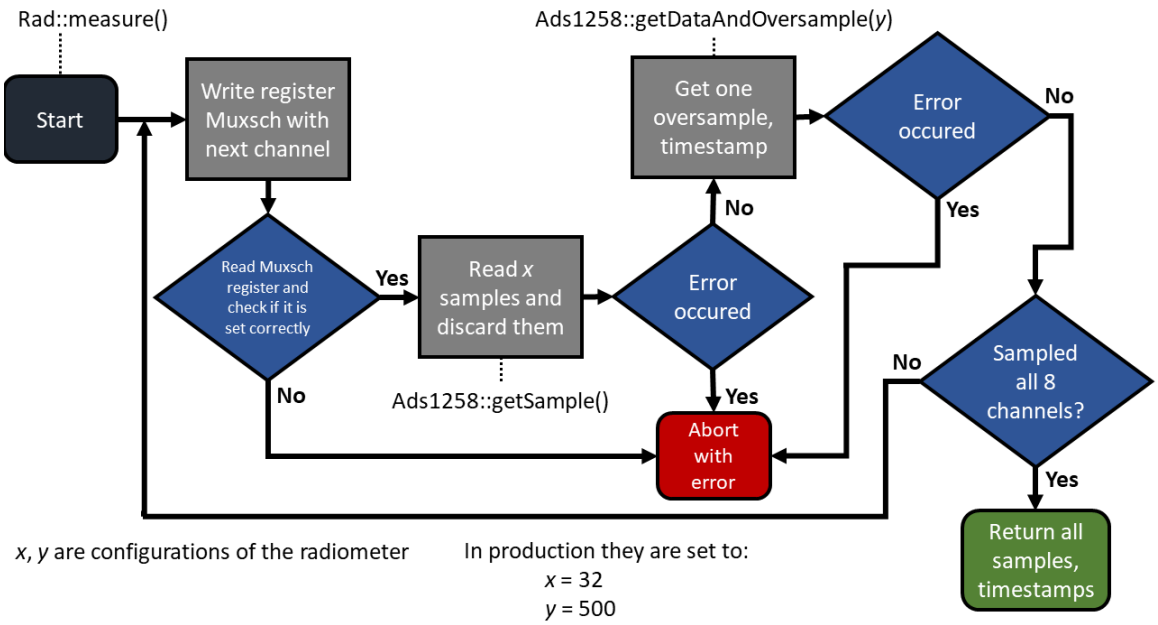


Figure 3.10.: Flowchart of function Rad::measure in radiometer driver

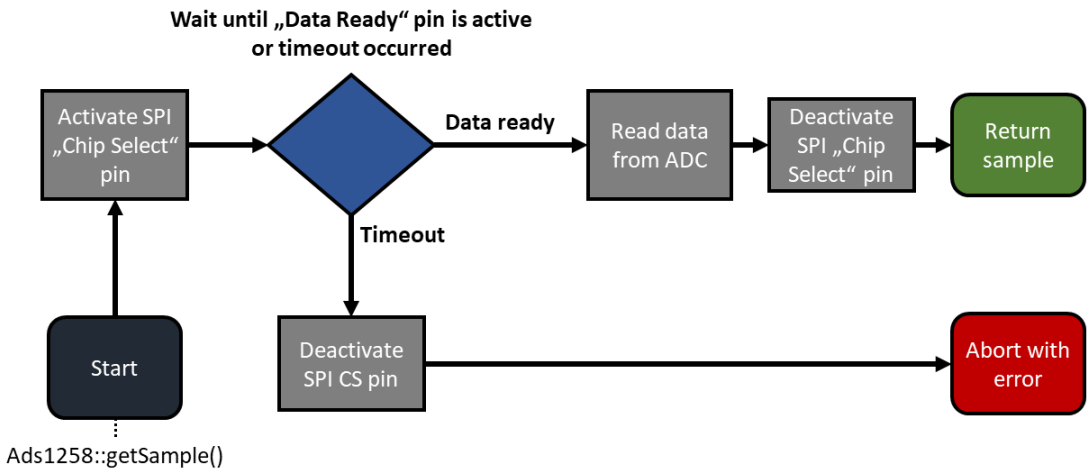


Figure 3.11.: Flowchart of function Ads1258::getSample in ADS1258 driver

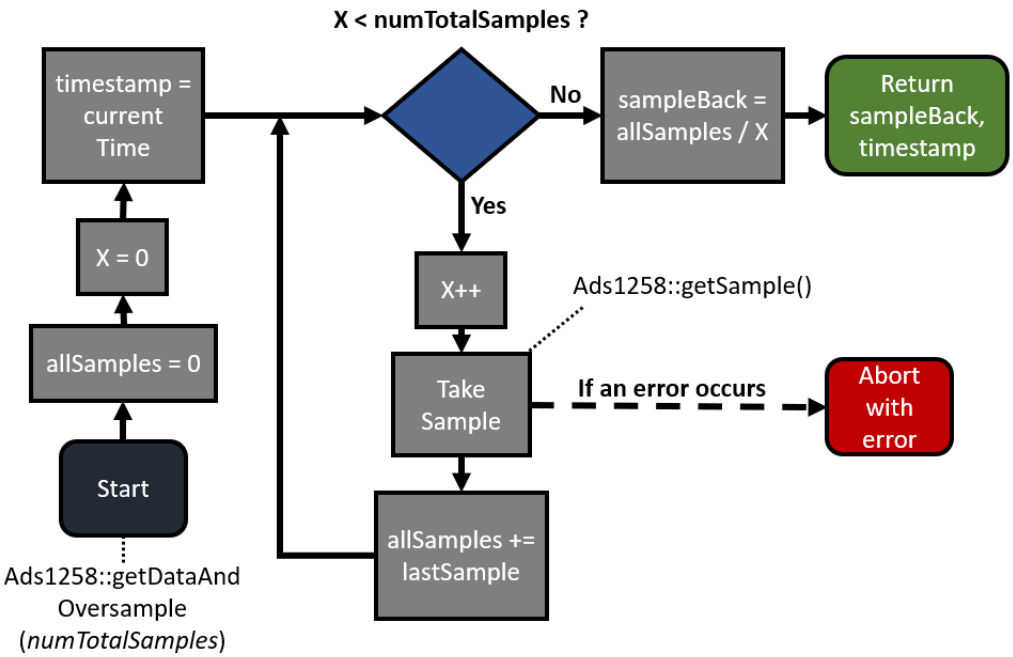


Figure 3.12.: Flowchart of function Ads1258::getDataAndOversample in ADS1258 driver

3.5.4. ADC Configuration

The configuration of the ADC is shown in Table 3.6. Here all registers of the ADS1258-ep are described with an explanation of why these settings are chosen. The prefix *0b* in the con-

figuration column stands for a number in binary representation, *0x* represents a hexadecimal number. The first rows separated by dashed lines are accessed by the register *CONFIG0*. The rows from *IDLMOD* to *DRATE* are set with the register *CONFIG1*.

Since the ADC is set to Fixed-Channel mode, some register settings do not have any effect, this is marked with "does not care". In these cases the default register configurations are set. More information about the functionality of the registers can be found in the ADS1258-ep datasheet [18, pp. 37–41].

Register Name	Register Setting Bit(s)	Configuration	Explanation
CONFIG0	SPIRST	0b0	Timeout option for SPI interface reset, does not matter for this driver
	MUXMOD	0b1	Fixed-Channel mode, driver switches the channel manually
	BYPAS	0b1	Activates external signal conditioning
	CLKENB	0b0	Disable generation of an external clock signal (not used in this design)
	CHOP	0b1	Activates signal chopping, one sample is measured twice with reverse polarity and averaged
	STAT	0b0	Default settings, does not care because of Fixed-Channel mode
CONFIG1	IDLMOD	0b0	Does not care, ADC does not idle in this design
	DLY	0b111	Sets highest internal delay of 384 µs after a sample was taken
	SBCS	0b00	Deactivates sensor bias current source (feature is not used in this design)
	DRATE	0b00	Highest possible internal oversampling with 64 samples for one oversample
MUXSCH	all	changing	Sets the channel combination which should be measured
MUXDIF	all	0x00	Default settings, does not care because of Fixed-Channel mode
MUXSG0	all	0xFF	Default settings, does not care because of Fixed-Channel mode
MUXSG1	all	0xFF	Default settings, does not care because of Fixed-Channel mode
SYSRED	all	0x00	Default settings, does not care because of Fixed-Channel mode
GPIOC	all	0x00	All GPIO (general-purpose input/output) pins are configured as outputs
GPIOD	all	0x00	All GPIO pins are tied to ground

Table 3.6.: Register configuration for the used ADC (ADS1258-ep)

3.5.5. External Oversampling

The radiometer should sample a channel longer than the maximum internal oversampling time. For that the previously presented oversampling algorithm in the firmware is implemented. Oversampling performed by the firmware is called external oversampling. In this chapter the number of samples for 1 s of external oversampling is calculated, as well as the integration time of this sample. The integration time is the time the measurement is actively being taken. This time is an important quantity when calculating the noise performance and is important in future work.

For the calculation of number of samples for external oversampling, the equation given in the datasheet is used to calculate how many samples per second can be taken by the ADC [18, p. 19]:

$$\text{Data Rate (Fixed-Channel Mode)} = \frac{f_{clk}}{128(4^{0b11-DR} + CHOP(4.265625 + TD)) \cdot 2^{CHOP}} \quad (3.7)$$

with the settings: $CHOP = 1$, $DR = 0b00$, $TD = 48$, $f_{clk} = 15.729$ MHz.

These settings result from the configuration given in Table 3.6 in combination with corresponding information from the datasheet. The internal clock f_{clk} is given by the "clock output frequency for internal clock" in the datasheet [18, p. 4]. TD is the "Time Delay" which is given by a table in the datasheet [18, p. 22, Table. 5].

This leads to:

$$\text{Data Rate (Fixed-Channel Mode)} \approx 528 \text{ Sps} \quad (3.8)$$

which is slightly larger than the 1075 / 2 = 537.5 Sps given in a reference table in the datasheet [18, p. 22], which was calculated with a higher clock speed of 16 MHz. In this design a crystal oscillator is directly tied to the ADC for clock generation, which results in a lower clock speed of 15.729 MHz.

To reduce noise it was decided that the radiometer measures each channel for 1s, and discard the first 32 samples of this measurement. The decision was made because of measurements results in Chapter 4.1.1. Here, different measurement procedures and settings were tested to find the setting which leads to acceptable noise and low offset voltages. Since the first 32 samples are discarded and the ADC can take 528 samples per second, this leads to:

$$N_{1s} = 528 \text{ samples} - 32 \text{ samples} \approx 500 \text{ samples} \quad (3.9)$$

for measuring a channel around 1 s. Therefore the firmware is set to oversample 500 samples into one sample.

Note that 1 s is not the active time the channel is integrating the measurement because after each measurement a pause of 384 µs is done. This delay is configured in the register *DLY* (see Table 3.6) and it was chosen to have the highest possible delay value. This gives the channel time to settle after switching of a channel.

$$t_{delay} = 384 \text{ µs} \quad (3.10)$$

To calculate the time a channel is integrated if the measurement is performed one second, one has to subtract the waiting time for this measurement from one second. The ADC waits twice for one sample, once before the first measurement is taken and once after the channel is inver-

ted in order to perform chopping. A diagram of this can be seen in Figure 3.8. This then leads to:

$$t_{500\ active\ sample} = 1\ \text{s} - (2 \cdot 500 \cdot t_{delay}) = 1\ \text{s} - 384\ \text{ms} = 616\ \text{ms} \tag{3.11}$$

This time is an important quantity when calculating the noise performance and is important in future work.

3.5.6. Error Handling

While the radiometer is operated by its driver, different errors could occur and should be handled. This chapter gives an overview of potentially occurring errors and describes strategies for handling them.

One potential error source is the SPI communication. Here, reading or writing might not work or timing errors could occur. Therefore, timeout functions have been implemented, so the SPI communication cannot block the flight computer with waiting forever for a signal to occur.

Another possible error could be that the "Data Ready" pin is never toggled by the radiometer. This could happen if there is a fault in the connection between the radiometer and the flight computer, for example if the radiometer is plugged out while a measurement is performed. Here the driver should have a timeout function implemented in order to detect this error.

Further, writing a configuration to the ADC could not be successful and the ADC may not save it. Therefore, a write to a register is always followed by a read back of the same register. If the read data does not match the written one, this is regarded as an error, since the ADC should directly change it's register value to the intended one.

In the current driver design, each occurring error is handled the same way and aborts the whole procedure. If an error occurs, the function which detected it returns an error to the caller of this function. If this caller is again part of the driver, it throws the error again to the calling function. This is repeated until the caller is the flight computer itself. The flight computer should repeat calling the main measuring function until it is successful. If the function call fails too often, most likely there is an electrical failure, and then the usage of the radiometer should be completely aborted. Power cycling could also be implemented in the flight computer in order to restart the radiometer properly.

Currently the caller does not know why the radiometer driver throws an exception and where this exception originates. This can be improved in the future and is included in future work, but for this prototype it is sufficient to know an error occurred.

In order to help debugging, each class can be set to a debug mode. This way, debug messages will be printed via the serial interface whenever an event of interest happens. This is not suitable for use during flight operations and does not replace a logging system, but this debug system could be modified into a logging system in the future.

At the current phase of development, the used microcontroller does not support multithreading or has a process management system since it is a bare-metal system and not a real-time operating system. Therefore, it is crucial to ensure that the driver of the radiometer does not stay in a blocking state for too long. To ensure this, blocking functions and code snippets are identified and for each a timeout function was implemented.

3.5.7. Data Processing

To evaluate the raw measurement data, two different firmware integration test cases, a python preprocessing script and a MATLAB plot script, were developed for easier and faster data processing. Each of the two integration test formats its output as a CSV table. Both test cases work the same way, the CSV table stores the measured data, their corresponding timestamps and their measured channels. The serial output of the microcontroller then is directly saved as a CSV table.

After the complete integration test has been performed, the generated table must be additionally preprocessed. For this, the developed python preprocessor script is used, which creates eight tables, one for each measured channel, out of one input table. Besides that, the script renames the channels for easier interpretation of the data. After the file is preprocessed, it can be plotted. Here the MATLAB script is used, which plots a $U(t)$ diagram as well as a histogram of the measured voltages for each channel. The histogram is supplemented with the measured average and the standard deviation of 1σ. Additionally, the script calculates the measured average and the standard deviation of the given channel and writes it separately to a file.

3.6. Startup Procedure

For better testing of the radiometer prototype, it was decided to not assemble all components at once. Instead, each component group was assembled and tested with a previously defined test strategy. The next assembly step was only done if all previous tests were successful. The assembly was done at the Institute of Planetary Research by technical personnel and each component group was hand soldered to the circuit board.

Due to limited time for the thesis, the radiometer was assembled and tested until the point of when the sensor simulators are attached. These sensor simulators consist of resistors with a similar resistance than the real sensors, but have no functionality. They are high precision resistors with a low temperature coefficient, in order to measure the temperature drift behaviour of the radiometer channels. This test and its results are presented in the next chapter. The last step to completely finish the radiometer is calibrating it and gathering final engineering budget data about the project, like noise and NETD performance.

The first assembly step regards all passive components like resistors, capacitors and inductors. These are assembled to the PCB. As a result, the passively filtered voltages 3.3 V and 5 V are applied to all corresponding traces and pins.

In a second step all operational amplifiers and the voltage reference are installed. After that, the external signal amplification can be tested by applying an external signal to the signal amplifier with the help of corresponding testpoints. Here, a *gain* of ≈ 11 is expected. Additionally, the 2.5 V bias voltage generated by the voltage reference is applied and measurable at the correct pins and traces.

The third assembly step regards the component group of the ADC. In this step, the ADC and the crystal oscillator are assembled. Further, the external Pt100 channel inside the aluminium body is substituted with a 100 Ω resistor to simulate the Pt100.

Now, the radiometer can be tested with the development board. The driver is tested as follows: It is flashed to the microcontroller and should produce first valid measurements of the external Pt100 channel and the external offset channel. Since the other channels are not completely

assembled yet, their measured data should not be overinterpreted. Still it is expected that the measurements of the not assembled Pt100 channels should result in 2.5 V, and the thermopiles should float around 0 V. This is due to their schematic designs and can be seen in Chapter 3.3.2 in more detail.

The calculation of the current flow through the Pt100 channel was done in Eq. 3.5 and resulted in $I_{Pt100} = 53.08\ \mu A$. Then, the voltage measured at the assembled Pt100 channel (substituted with a 100Ω resistor) should be around:

$$U_{Pt100} = R_{Pt100} \cdot I_{Pt100} = 100\ \Omega \cdot 53.08\ \mu\text{A} = 5.308\ \text{mV} \tag{3.12}$$

$$U_{meas} = U_{Pt100} \cdot gain = 5.308\ \text{mV} \cdot 11 \approx 58.39\ \text{mV} \tag{3.13}$$

The voltage measured at the external offset channel should be around 0 V and should only be of the order of µV, which represents the offset voltage introduced from the circuit board and their components.

The next assembly step includes assembling sensor simulators instead of the real ones. These sensor replacements consist of resistors with a similar resistance and a low temperature coefficient of ± 5 ppm / °C. The behaviour of each channel at different temperatures is then measured. With this data it is possible to determine the measurement uncertainty of each channel and how this uncertainty drifts with temperature.

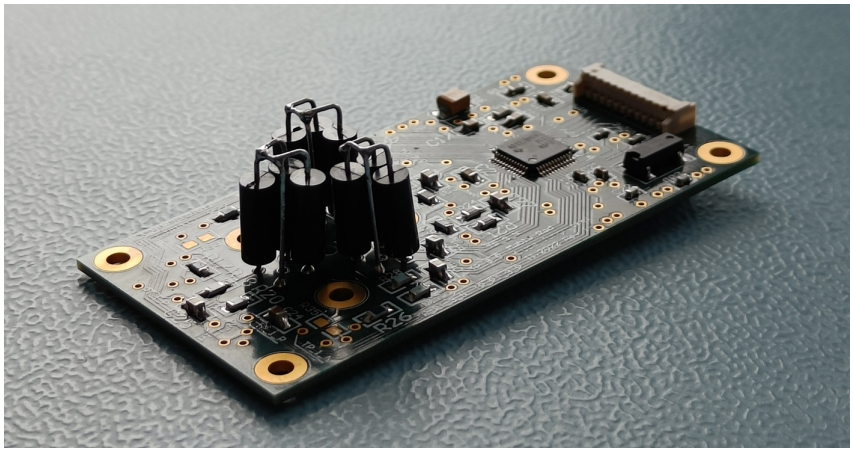


Figure 3.13.: Radiometer with assembled sensor simulators
The sensor simulators were installed in order to measure temperature drifts between -20 °C and +60 °C. The sensors are substituted with high precision resistors with a low temperature drift of ± 5 ppm / °C. More details about this configuration and its results can be found in Chapter 4.2.

The last step for finishing the radiometer would be removing the sensor simulators and the 100 Ω resistor from the external Pt100 channel. Now, all thermopile sensors and the real Pt100 would be installed. After that, the aluminium body would be assembled and the Pt100 would be glued inside it. Now the radiometer should be calibrated with a laboratory black body and finally integrated into the prototype Mars weather balloon. This step was not done within the scope of this thesis, but for completeness it is shortly described here.

3.7. Development Board for Radiometer

In order to develop and test the radiometer, a development board (called "Mable-Rad Development Board") was designed and built. The goal was to resemble the flight computer which is used during flight operations and to allow for an easy and safe testing of the radiometer. The development board has the same electrical connections, the same pinout as the production flight computer, it uses an identical connector to the radiometer and provides the same power rails. Identical code flashed on the flight computer should lead to the same behaviour of the radiometer as when flashed to the development board. The development board is connectable to a computer with a debugger and can communicate to it via a serial interface to provide measured data or status information of the radiometer. The used pinout for the connection between radiometer and development board is given in Table 3.2.

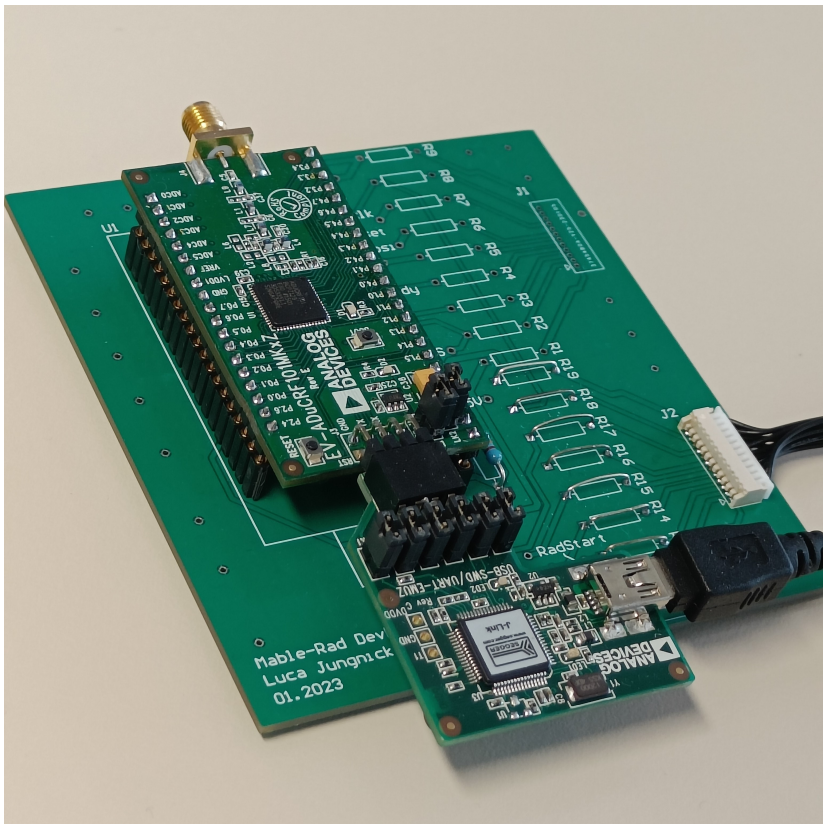


Figure 3.14.: Development Board for radiometer "Mable-Rad DevBoard"
The development board provides the same microcontroller as used for the flight computer, which is connected with a debugger to a computer. The circuit board features a connector to connect the development board directly to the radiometer. All pins have a 0 Ω resistor connected in series. If needed, another resistance can be easily assembled here, for example a 50 Ω for suppressing ringing on a data line could be integrated. The connector is doubled and added again in the upper half of the circuit board, which is currently not used and not assembled. This allows for additional test points and can help with debugging the radiometer. All pins from and to the radiometer are also available on the circuit board as well via labelled testpoints.

4. Experiments and Results

This chapter evaluates the developed radiometer and describe the performed validation tests in order to verify that the radiometer is working as expected are presented here. In total two experiments were performed, which are presented here. The first one, in Chapter 4.1, tested different measurement configurations to determine a suitable configuration of how the measurements should be performed by the radiometer. In the second test, presented in Chapter 4.2, the radiometer was tested inside a temperature test chamber, and the temperature drift of all channels were analyzed. In order to verify the functionality of each channel for both tests, all taken measurements were compared with precalculated expected values and the results were evaluated.

4.1. Measurement Configuration Test

In this experiment, the goal was to test different measurement configurations and determine their effects on offset and noise of the measurements. Additionally, this experiment includes a function test of the measured channels, where expected and precalculated values are compared with the measured voltages in order to determine if they are plausible. All measurements took place at an unstabilized room temperature of around 21 °C. The hardware setup is shown in Figure 4.1. Here the radiometer is directly connected to the development board which controls the radiometer. The development board is connected to a debugger which provides a serial interface for a computer. The development board then outputs the measurement data via this interface.

At this point of development, only the offset channel and external Pt100 channel are populated and this data can be interpreted. While the offset channel is completely assembled, the external Pt100 inside the aluminium body is not assembled with a Pt100 but with a 100 Ω resistor instead. The chosen resistor is a commonly available 0805 SMD resistor with an accuracy of 1% and an unknown temperature coefficient. It is expected that the temperature coefficient is likely large. For easier readability, this Pt100 channel, integrated inside the aluminium body, is called external Pt100 channel going further, since it is the only Pt100 of the radiometer which is not housed in the thermopiles. The other three Pt100s are all directly integrated inside the thermopile's housing, and therefore called internal Pt100s.

In order to understand the chosen configurations, first the term sample discard has to be defined in this context. Sample discard describes the number of samples which are discarded after a channel is switched by the ADC. A higher sample discard could lower the noise of a measurement. Since after a channel switch a different voltage level from the next channel is applied to the ADC, the measurement chain needs time to settle. If the next sample after a channel switch is taken too fast, the measuring chain could not be completely settled and the ADC would measure a non-settled voltage. This measurement will be averaged with the later measured settled values, but it is still influencing the measured value and leads to an increase in noise. To avoid this, it is possible to discard samples after a channel switch. These samples are

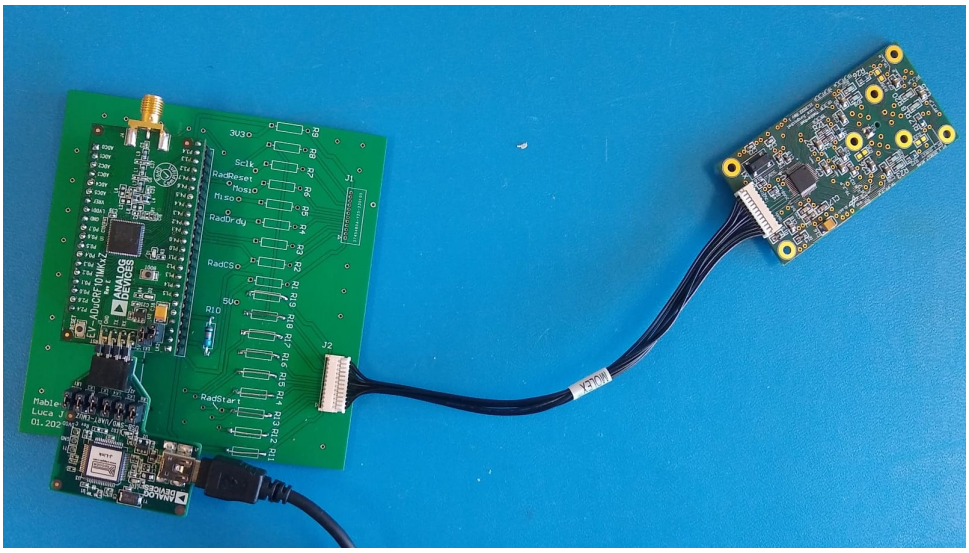


Figure 4.1.: Measurement setup for the test of different measurement configurations, for the offset and external Pt100 channel

On the left there is the development board which is connected via a debugger to a computer. The radiometer on the right is directly connected to the development board. As can be seen, at this stage no sensors and no aluminium body were assembled, therefore only the offset and (replaced with a 100 Ω resistor) the external Pt100 channel give valid data.

still measured, but not included in the averaged sample. In this experiment, the first tested configuration does not drop samples and is called 0 sample discard. The other tested configuration discards 32 samples which introduces an additional time delay of

$$(1 \,/\, 528 \text{ samples per second}) \cdot 32 \text{ samples} \approx 6 \text{ ms} \tag{4.1}$$

It should be noted that due to the set internal delay feature a time delay of 384 us is still present after each measurement. This gives the ADC time to settle which is necessary, since the activated external chopping feature switches the polarity of the two input connections after a measurement is taken. The polarity switch also needs time to settle. This is still referred as 0 sample discard, since no samples taken are dropped.

The two assembled channels are measured with different measurement configurations. In total, four different configurations are tested. Two of the measurements are taken with the channels measured continuously and at different sample lengths of 0.1 s and 1 s respectively. A continuous measurement of a channel means that the ADC always stays on the same channel and no channel switches are done between samples. Since the channel is never switched, sample discarding would not have an effect on the measurement because sample discarding is only effective when a channel is switched to a different one and the voltage level applied to the ADC changes. Therefore, sample discard is set to 0 for these measurement configurations. The test between these two configurations is especially interesting to see what effect the integration time has on the noise of a measurement.

For the two other measurement configurations it was chosen to perform a channel switch after each measurement. When the radiometer is in use on the weather balloon, it also follows the procedure of channel switching. The integration time for all channels are here equal. Then, the channel is switched to the next one until all eight channels are measured. In this test, the

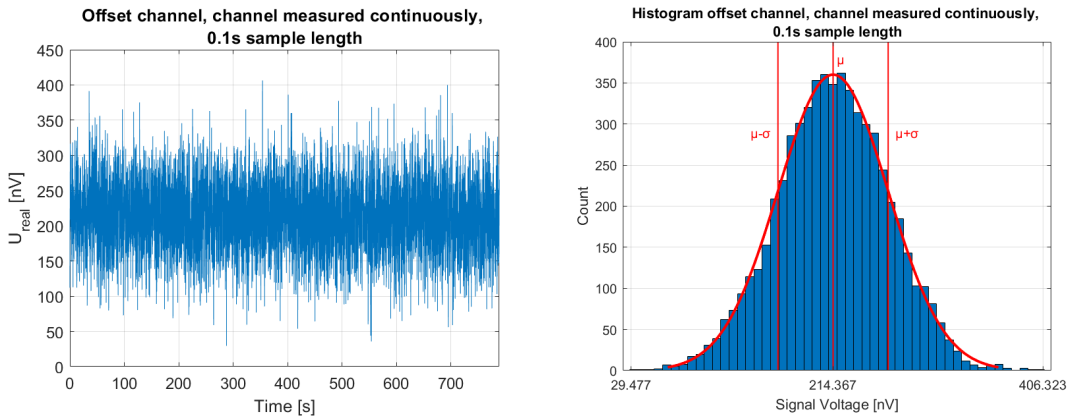
impact of different sample discard options on the standard deviation of the measured signals are tested. This allows a statement to be made as to which measurement configuration leads to the lowest noise and how big the differences in noise are between different configurations.

For each measurement a $U(t)$ plot is then created, as well as a corresponding histogram. The plotted values are the real applied voltages and called U_{real}. Due to the external amplification with a gain of 11, each measured voltage has to be divided by this factor in order to get the voltage of the input of the multiplexer. The measured voltages are not plotted for this thesis but can be derived by multiplying the real voltages by 11. This relationship also applies for the measured average voltage μ_{meas} and standard deviation σ_{meas}, these quantities also have to be divided by the gain. For further data evaluation, it is assumed that the gain is 11. This is not completely true, but a control measurement of the gain lead to a gain factor of 10.96, resulting in a negligible error of 0.36%.

$$\mu_{real} = \frac{\mu_{meas}}{Gain} = \frac{\mu_{meas}}{11} \tag{4.2}$$

$$\sigma_{real} = \frac{\sigma_{meas}}{Gain} = \frac{\sigma_{meas}}{11} \tag{4.3}$$

The measured data is summarized in Table 4.1. Here, the averaged voltages of each measurement configuration of the offset and the external Pt100 channel is shown, as well as the standard deviation for each measurement configuration of the two channels. This standard deviation shows the measurement uncertainty and can be regarded as the noise level. All $U(t)$ and corresponding histograms, which are the basis for the collected data in the tables, are shown in the appendix (see Figures A.1 - A.8) for reference. An excerpt of how this data is plotted is given Figure 4.2, here the measured $U(t)$ and the corresponding histogram of the measurement configuration of a 0.1 s sample length with a continuously measurement is given.



(a) Measured voltage of the offset channel plotted over time

(b) Corresponding histogram

Figure 4.2.: Offset channel, measured continuously with a 0.1 s sample length
Sample length: 0.1 s
Sample discard: 0
Number samples taken: 6000
μ_{real} = 214 nV
σ_{real} = 50 nV

Config.	Length	N	Discard	μ_{real} Offset	μ_{real} Ext. Pt100	σ_{real} Offset	σ_{real} Ext. Pt100
Continuous	0.1 s	6000	-	214 nV	4.998 mV	50 nV	31 nV
Continuous	1 s	600	-	214 nV	4.998 mV	17 nV	11 nV
Switched	1 s	100	0	214 nV	4.998 mV	19 nV	44 nV
Switched	1 s	500	32	213 nV	4.998 mV	16 nV	11 nV

The Pt100 of the external Pt100 channel is substituted with a 100 Ω resistor. A continuous measurement configuration describes the measurement of a channel without channel switches between samples, a switched measurement configuration switches the channels after a sample was taken. Discard stands for how many samples are discarded after a channel switch is performed. N is the number of samples taken, which are then averaged into μ_{real}. σ_{real} is the calculated standard deviation of each measurement series, which corresponds to the noise level.

Table 4.1.: μ_{real} and σ_{real} of offset channel and external Pt100 channel for different measurement configurations

4.1.1. Interpretation of Measured Voltages

Measured Voltage of Offset Channel

As shown in Table 4.1, the applied voltage at the offset channel is around 214 nV. These measured values have a high degree of repeatability since the number of samples taken are high for all configurations and all different measurement configurations lead to a similar result. Ideally, a voltage of exactly 0 V would be measured at the offset channel due to the use of the chopping feature. Since all components in the measurement chain are not ideal, an offset voltage differing from 0 V is measured at this channel. Reasons for that are for example a difference in resistance of the two used 10 kΩ resistors in the offset channel circuit, and asymmetrical currents introduced by the measurement chain. For a comparison of the magnitude of the 214 nV offset voltage, it is important to compare this value with the typical introduced offset voltage by the ADC, which is specified in its datasheet. Typically, the ADC introduces an offset voltage of 1 µV when the external chopping feature is on, in the worst case 10 µV are introduced [18, p. 3]. Therefore, the measured offset voltage is significantly better than expected. It is expected that this offset voltage is the same for each measured channel and it is mostly introduced by the shared measurement chain of all channels. In order to improve measurement results of the other channels, their measurement values should be subtracted from the measured offset voltage. This contributes to cancelling out offsets and decreases measurement uncertainty by removing systematic errors.

Measured Voltage of External Pt100 Channel

The measured real voltage, already divided by the amplifier gain, at the external Pt100 channel is 4.998 mV, as seen in Table 4.1. This measurement also has a high degree of repeatability for the same reasons as for the offset channel before. To check whether the measurement is plausible, the theoretical voltage drop across the Pt100 was calculated and compared with the measured value. As calculated in Eq. 3.5, the current flowing through the Pt100 should be $I_{Resistor}$ = 53.08 µA. Then, the voltage drop across the 100 Ω resistor should be

$$U_{Resistor} = R_{Resistor} \cdot I_{Resistor} = 100\ \Omega \cdot 53.08\ \mu\text{A} = 5.308\ \text{mV} \tag{4.4}$$

The relative error between the measured and the calculated voltage in percent is

$$\delta_x = \frac{U_{resistor} - U_{meas}}{U_{resistor}} \cdot 100\% = \frac{5.308\text{ mV} - 4.998\text{ mV}}{5.308\text{ mV}} \cdot 100\% = 5.84\% \tag{4.5}$$

This high amount of error is not expected. Therefore, a more detailed error analysis was performed in order to find the reason for this magnitude of error. For analyzing the influence of different component groups, the following relationship between the component groups applies:

$$\begin{aligned} U_{meas} = \text{Gain} \cdot U_{real} = \text{Gain} \cdot R_{100\Omega} \cdot I_{resistor} = \text{Gain} \cdot R_{100\Omega} \cdot \frac{U_{reference}}{R_{all}} \\ = \text{Gain} \cdot R_{100\Omega} \cdot \frac{U_{reference}}{R_{100\Omega} + R_{47k\Omega}} \end{aligned} \tag{4.6}$$

Possible reasons for this deviation were determined and their degree of deviation were approximated with the help of Eq. 4.6. This leads to the following worst case errors for each component group:

Component Group	Worst Case Error	Influence on Measured Voltage
100 Ω resistor	1 %	High
47k Ω resistor	0.1 %	Very Low
2.5V voltage reference	0.1 % [21, p. 1, REF43GSZ]	Low
Gain instrumentation amplifier	1 % (estimation)	High
Offset voltage	213 nV	Low

Table 4.2.: Overview of relevant component groups and their worst case error for analysing the high deviation of measured to expected voltage in the external Pt100 channel

The 100 Ω resistor has a worst case error of 1%. For the 47 kΩ, precision resistors were chosen, leading to a worst case error of 0.1%. The worst case error of the voltage reference is taken directly from its datasheet and is 0.1%. The deviation of the external amplifier gain is expected to be 1% in the worst case. This is only an estimation, a more detailed worst case analysis of the external instrumentation amplifier circuit was not performed here. Since the gain determining resistors in the amplifier circuit are precision resistors, it is likely that the gain has a lower uncertainty than the stated 1%. The offset voltage was derived from the previous analysis of the offset channel and has almost no influence due to its very small magnitude in the order of a few hundred nV.

As seen in Table 4.2, the measured relative error of 5.84% cannot be explained by the worst case values of the used component groups. To further narrow down the error, each component group was controlled individually and all values were manually measured: The measured reference voltage is 2.498 V instead of 2.5 V, leading to a relative error of 0.08%. This is in the expected error range of maximum 0.1%. The instrumentation amplifier has a gain of 10.96 instead of 11, leading to a relative error of 0.36%, which is also in the expected error range. The resistance of the two relevant 100 Ω and 47 kΩ resistors cannot be measured directly since they are already assembled in the circuit. Because of that, resistors from the same batch were measured for control. Another 100 Ω resistor from that batch has a resistance of 100.7 Ω. When controlling a resistor from the 47 kΩ batch, the mistake was found, since all resistors from this batch have a resistance of around 50 kΩ. Calculating the expected voltage across

the 100 Ω resistor with a current-limiting resistor of 50 kΩ instead of 47 kΩ , while all other component groups have ideal values, leads to

$$U_{should} = 100\ \Omega \cdot \frac{2.5\ \text{V}}{50.1\ \text{k}\Omega} = 4.990\ \text{mV} \tag{4.7}$$

which has a relative error to the measured value of

$$\delta_x = \frac{4.998\ \text{mV} - 4.990\ \text{mV}}{4.998\ \text{mV}} \cdot 100\% = 0.16\% \tag{4.8}$$

This error is well within the expected range and the error was found. For future work, all 50 kΩ resistors have to be replaced with high precision 47 kΩ resistors.

4.1.2. Interpretation of Standard Deviation

Influence of Sample Length

In the first two measurement configurations, the influence of the chosen integration time on the standard deviation can be analyzed. For the offset channel the noise levels are 50 nV for a sample length of 0.1 s, and 17 nV for a sample length of 1 s. As expected, the higher the sample length gets, the lower the noise level becomes. For the offset channel, the noise level increases by a factor of 50 nV/17 nV ≈ 2.9. Regarding the external Pt100 channel, the same behaviour of a lower noise level at a higher sample length can be observed. Here, the noise level drops from 31 nV for a 0.1 s sample length to 11 nV for a 1 s sample length. This leads to a factor of 31 nV/11 nV ≈ 2.8. The influence of sample length on the standard deviation results from the definition of the standard deviation, as it can be seen in its formula: [22, pp. 1378–1379]:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \mu)^2}{N}} \tag{4.9}$$

where x_i is the measured value from the finite data set which was measured, μ the mean value of the data set, and N is the number of measured values. Since the afctor difference between the two chosen sample lengths of 0.1 s and 1 s is 10, a difference in the standard deviation between the two measurements of $\sqrt{10} \approx 3.16$ is expected. This is around the measured factor of 2.8.

As a result of these measurements, a sample length of 1 s for each channel was chosen. The higher the sample length is, the lower the noise level, but the instrument needs more computation time of the flight computer. Here, a 1 s sample length per channel proves to be a good compromise between a suitable time length and a high precision of the radiometer.

Influence of Sample Discard

Regarding the two tests with different sample discard options of 0 and 32 sample discard, it can be observed that a larger number of discarded samples leads to a lower noise level, which is expected. For the offset channel, the noise level drops from 19 nV to 16 nV when setting the sample discard from 0 to 32. This is only a small improvement in noise reduction. More interesting is the noise level drop of the external Pt100 channel which drops from 44 nV to 11 nV.

This difference in noise can be explained when regarding the channel which was set before the ADC switched to the external Pt100 channel. When the channel switched, the applied voltage drops from 2.5 V to the external Pt100 channel voltage, which is 55 mV ($11 \cdot 4.998$ mV). Since the measurement chain is not completely settled due to that no samples are discarded, the noise level increases significantly compared to when sample discard is set to 32. This explaines the high difference in noise for the external Pt100 channel with 0 sample discard. For the offset channel, this effect could not be seen so clearly since the channel before had a voltage of around 50 mV, which then dropped to the offset voltage of 2.2 mV ($11 \cdot 214$ nV).

This test showed that best results can be achieved when the sample discard option is set to 32 instead of 0. Also, it shows that the improvement in lower noise depends on the channel which was set before. If the difference in voltages between the channels next to each other is high and no sample discard is set, the noise level increases. As a result of both tests, a measurement configuration for the radiometer of a sample length of 1s and a sample discard of 32 is set in the firmware, since this configuration proved to result in lowest noise compared to the other test cases.

Differences in Absolute Noise Levels

In all tests, except the 0 sample discard test case, the external Pt100 channel has lower noise levels than the offset channel. As explained above, the 0 sample discard case is an exception since the measurement chain here is not completely settled and is not regarded here further.

For explaining the observed effect, it is necessary to analyze the corresponding electrical circuits (see Appendix, A.3.1). The Pt100 circuit has 1 kΩ resistors in series between ADC and its measuring probe, while the offset measuring circuit has 10 kΩ resistors instead. Therefore, it is believed that the higher noise level in the offset circuit comes mostly from thermal noise of these resistors with a higher resistance, since the noise introduced by a resistor depends on its resistance [23, p. 107–108]. This relationship is known as Johnson-Nyquist noise.

4.2. Temperature Drift

For this experiment, the goal was to determine the influence of temperature on the behaviour of the radiometer. Especially the drift of the offset voltage is of interest, since all channels are influenced by this drift. When the behaviour of the offset channel drift is well known, it can be cancelled out when data from the radiometer is interpreted. This removes a source of systematic error and reduces measurement uncertainty.

In order to determine the temperature drift of the thermopile channels, it was chosen to assemble a configuration of two precision resistors with a very small temperature drift coefficient instead of the thermopile sensors here. Each resistor combination has a different ratio. There are two asymmetrical combinations of a 1 kΩ -100 kΩ and a 10 kΩ -100 kΩ assembled, as well as a symmetrical resistor combination of 10 kΩ -10 kΩ. Here, it was important to choose resistors with a temperature drift as low as possible since it is wanted to solely measure the temperature drift of the measurement chain. The assembled resistors will be later replaced with the thermopile sensors when the radiometer will be fully integrated. Therefore, the influence of these resistors should be as small as possible. With the assembly of asymmetrical resistors, it is possible to measure the influences of errors sizes better, since they are not cancelled out by the external chopping of the ADC and are emphasized in the collected data.

The Pt100 inside the thermopiles were replaced with 100 Ω high precision and low temperature coefficient resistors each. Here it is also important to choose resistors with a low temperature drift, since the same goal as for the thermopile measurement applies: it is only wished to measure the temperature drift of the measurement chain.

The assembly of the external Pt100 channel remains the same as in the previous experiment, it is assembled with a generic 0805 SMD 100 Ω resistor. Here, it is important to note that this resistor has an unknown high temperature drift coefficient and an unknown precision. It is believed that the resistance has a worst case deviation of ±1%. The offset channel is fully assembled as specified in the schematics.

An overview of how each channel is assembled for this experiment can be seen in Table 4.3.

Channel	R_1	R_2	Type
Thermopile 1	1 kΩ	100 kΩ	precision resistors
Thermopile 2	10 kΩ	100 kΩ	precision resistors
Thermopile 3	10 kΩ	10 kΩ	precision resistors
Pt100 of Thermopile 1	100 Ω	-	precision resistor
Pt100 of Thermopile 2	100 Ω	-	precision resistor
Pt100 of Thermopile 3	100 Ω	-	precision resistor
External Pt100	100 Ω	-	standard resistor
Offset	-	-	completely assembled

A precision resistor stands for ±0.1 % precision ± 5 ppm / °C TC resistor, a standard resistor for a standard 0805 SMD resistor with a high, unknown TC. The offset channel is a special case here, since it is already completely assembled as in the final radiometer. All other channels are substituted by the given resistor configuration.

Table 4.3.: Overview of channel's type of assembly for temperature drift test

The recording of this series took place inside a laboratory climate chamber. To create enough data for a meaningful interpretation, it was chosen to create a measurement series from -20

°C (253.15 K) to +60 °C (333.15 K) in steps of 20 °C. The measurement started at -20 °C and incremented then in steps of 20 °C until it reached 60 °C. In order to find this suitable temperature range, all integrated components have been checked for their maximum allowed temperature range. The component with the lowest temperature range then determines the used range. In this radiometer design, the voltage reference REF43 had the smallest temperature range from -40 °C to 85 °C [21, p. 1]. Therefore, a more conservative range of -20 °C to +60 °C was chosen. This way, the radiometer operated well within the possible range and was safe from failure due to extreme temperatures. A higher temperature range would have been also possible to choose, but a range of 80 °C is sufficient to determine the drifting behaviour of the channels.

For each temperature, 30 min of data at a stable temperature plateau have been collected. This leads to a total number of samples of $(30\text{ min} \cdot 60) \cdot 1\text{ Sps} = 1800$ samples for all channels. Each channel was measured with $1800/8 = 225$ samples. After 30 min of measurement at a stable temperature, the temperature was increased by 20 °C. The settling of the temperature took around 20 min - 25 min. When the temperature was settled, the next 30 min of measurement continued. The radiometer never stopped sampling during this test, therefore a voltage over time plot for each channel exists over all temperatures, which is inserted for each channel in the appendix. The different temperature ranges over time are shown in an example $U(t)$ plot (Fig. 4.3). Here, a complete $U(t)$ plot of the Pt100 of thermopile 1 channel is visible. The highlighted data areas are where the temperature was regarded as settled and 30 min of data was taken. The temperature changes are also clearly visible and are between the highlighted temperature areas. The measured voltage becomes unstable and a clear drift until the measurement is settled again can be seen here.

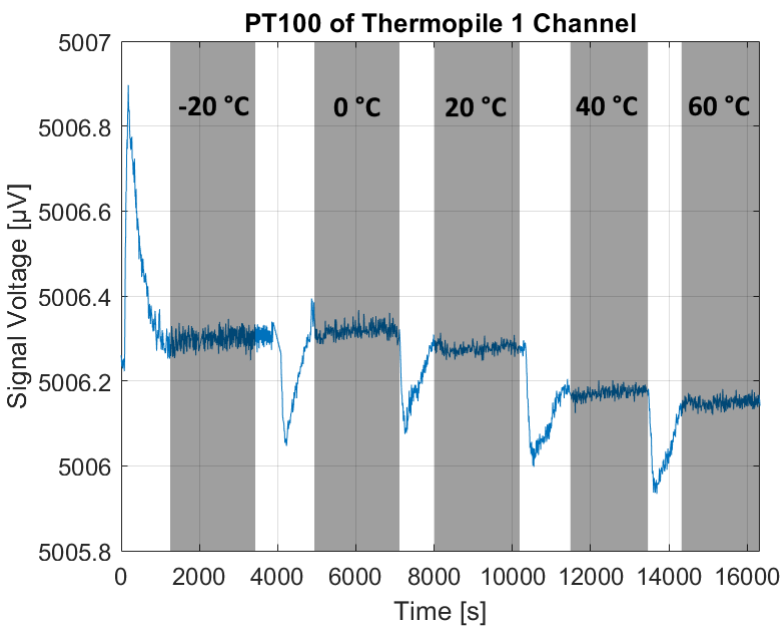


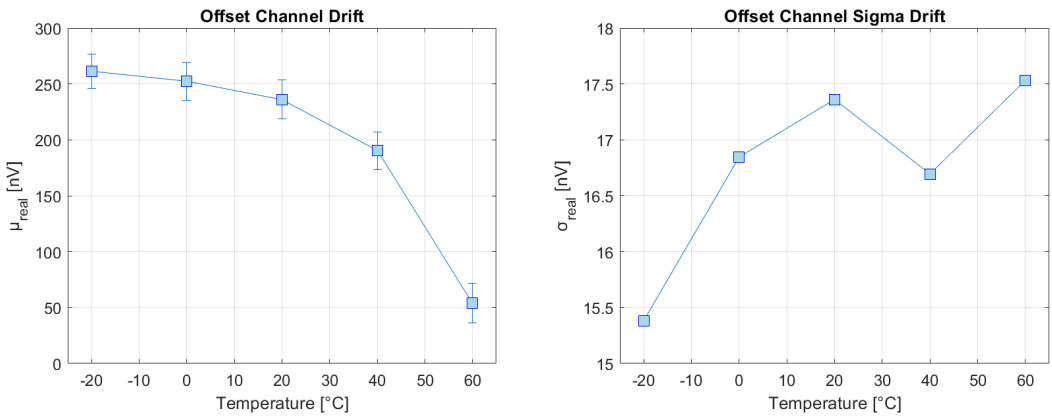
Figure 4.3.: $U(t)$ of Pt100 of thermopile 1 channel with highlighted stable temperature areas

For each measured channel, three different plots were created. The first one is a complete $U(t)$ plot, giving an overview over the measurement and the channel behaviour during temperature changes from -20 °C to 60 °C in steps of 20 °C. Visible plateaus here correspond to a stable

temperature inside the laboratory climate chamber. Resulting from this graph, a second plot was created, showing the drift of average voltage as a function of temperature. For this graph, the average voltage μ_{real} at a settled temperature was calculated and plotted at their corresponding temperature. Additionally, an error bar corresponding to the standard deviation was included. The third graph shows the drift of the standard deviation σ_{real} as a function of temperature. For this data, the standard deviation of each data set of a settled temperature was calculated and plotted at their corresponding temperature.

It should be noted that at the point of this experiment, the channel Pt100 of thermopile 2 was not included in this evaluation. Here, it was found later that the corresponding ADC pin was not correctly soldered to the PCB, resulting in incorrect data. It is believed that this channel behaves similarly to the Pt100 channel of thermopile 1 and 3, since all of these channels are similarly assembled and have the same functionality.

4.2.1. Interpretation of Offset Channel Measurements



(a) Temperature drift of average measured voltages as a function of temperature

(b) Temperature drift of standard deviation as a function of temperature

Figure 4.4.: Offset channel's temperature drift of average voltage and standard deviation

The measured data of the offset channel is summarized in two plots shown in Figure 4.4. In these two plots, the average measured voltage over the given temperature range, as well as the standard deviation of a measurement of each temperature are shown. This data is already summarized from the complete $U(t)$ plot showing the voltage as a function of time for all temperatures. For a simplified view, this plot is not included here. It can be found in the appendix (see Figure A.9).

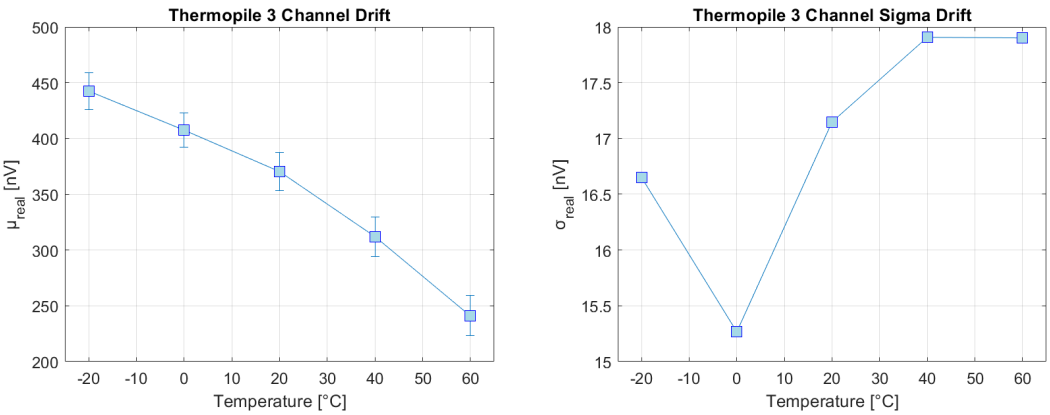
The average voltage of the offset channel, visible in Fig. 4.4a, drifts strictly monotonically decreasing, starting at 260 nV at -20 °C and ending at 50 nV at 60 °C. Since the ideal offset voltage is 0 V, the worst case offset measured is | -260 nV | = 260 nV. This is well within the expected range of 1 µV, which is the typical offset voltage introduced by the ADC when it performs external signal chopping [18, p. 3]. The offset voltage drift peaks between 40 °C and 60 °C where it drifts on average -7.5 nV / °C. As stated in its datasheet, the ADC has a typical offset voltage drift of ± 20 nV / °C [18, p. 3]. Compared to this, this measured offset drift is significantly better than expected.

Generally, the measured offset voltage drift is the result of the offset voltage drift of all used components in the measurement chain. It is believed that the drift mostly results from the operational amplifiers inside the instrumentation amplifier, as well as from the ADC. Comparing the stated offset voltage drift, the used operational amplifier ISL28227 has a typical offset voltage drift of 100 nV / °C [20, p. 7], which is five times more than the ADC. Therefore, it can be assumed that the operational amplifiers have more influence on the offset voltage drift than the ADC, since they typically drift significantly more than the ADC. The majority of the operational amplifier's offset voltage is cancelled out by the external chopping feature of the ADC. However, a small percentage of voltage cannot be removed and remains. This amount of voltage then drifts with temperature and is measured. Further, the voltage that is not removed by chopping, an additional offset voltage from the ADC is added. This voltage also drifts with temperature.

Regarding the noise levels of the offset voltage measurements visible in Figure 4.4b, it can be seen that they range between 15.4 nV for -20 °C and 17.5 nV for 60 °C. These noise levels are very small, showing that the noise performance of this offset measurement is excellent. Additionally, only a very small increase of 0.02 nV / °C in noise is observed. Therefore, the noise level at the offset channel is virtually constant.

4.2.2. Interpretation of Thermopile Channel Measurements

Symmetrical Resistors, Thermopile 3



(a) Temperature drift of average measured voltages as a function of temperature

(b) Temperature drift of standard deviation as a function of temperature

Figure 4.5.: Thermopile 3, assembled with 10 kΩ -10 kΩ high precision low TC resistors, its temperature drift of average voltage and standard deviation

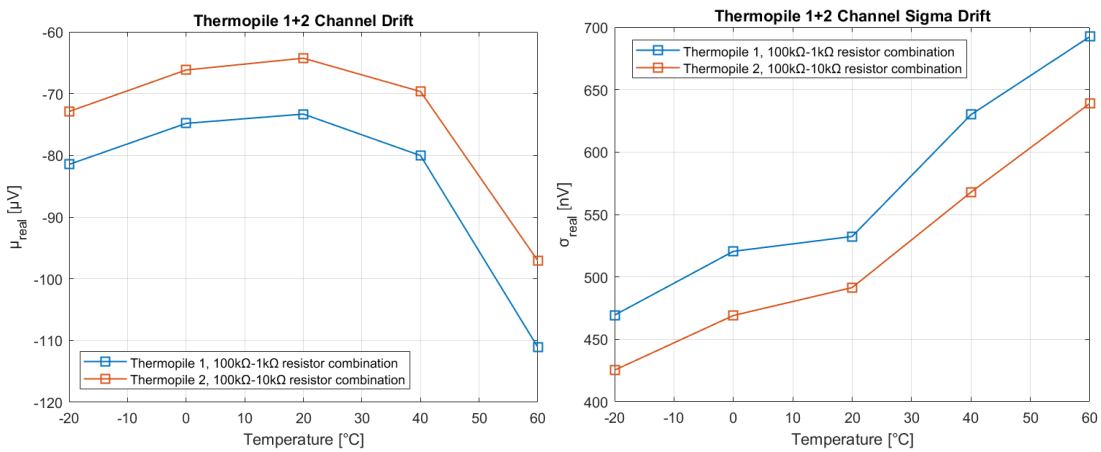
The thermopile 3 channel is symmetrically assembled with two 10 kΩ 0.1% tolerance and 5 ppm / °C resistors. Therefore, the thermopile 3 circuit is similar to the offset measurement circuit and the values are compared with it. Regarding the average measured voltages, a similar trend in relative values can be observed. This is expected since both channels are assembled similarly. The voltages in both channels are falling strictly monotonously. The voltage at the thermopile 3 channel drops around 200 nV over a range of 80 °C, while the offset channel

voltage also drops around 200 nV (see Figure 4.4a). Therefore, it is assumed that the offset voltage drift of -200 nV from -20 °C to 60 °C are channel independent and is generated by the measurement chain itself. From this, it follows that the offset voltage drift can be expected to be similar for all other channels.

Although the drift behaviour between the two channels is similar, a notable difference is the offset in the measured voltages. The offset channel has an average voltage of around 250 nV at 0 °C, while the voltage at the same temperature of the thermopile 3 channel is at 400 nV. This absolute offset of around 150 nV is not expected. It is expected that the absolute offset voltages have the same magnitude for both channels. This additional offset could result from the electrical parts that are not shared by both channels, as well as the channel traces. These are namely leak currents of the multiplexers in the ADC, the traces, as well as external interference on the thermopile 3 channel, which could introduce this additional offset. Nevertheless, the worst case offset of around 450 nV is still below the typical offset of 1 µV [18, p. 3] by the ADC. Therefore, the thermopile 3 offset voltage is clearly in the expected range. In the future, the offset could be removed by subtraction of the measurement values.

The noise level, visible in Figure 4.5b, is similar to the noise level of the offset channel. The noise voltages are between 15.3 nV and 17.9 nV, which are very low noise levels. Most likely they result from thermal noise introduced by the measurement chain. The noise level can be regarded as virtually constant. This is because the noise difference over a temperature range of 80 °C is 2.6 nV, which results in a very small noise drift per degree Celsius of 2.6 nV / 80 °C $\approx$ 0.033 nV / °C.

Asymmetrical Resistors, Thermopile 1 + 2



(a) Temperature drift of average measured voltages as a function of temperature

(b) Temperature drift of standard deviation as a function of temperature

Figure 4.6.: Thermopile 1 and 2, their temperature drift of average voltage and corresponding standard deviation

The thermopile 1 and 2 channels are assembled with an asymmetrical combination of resistors: a 100 kΩ -1 kΩ resistor combination for the thermopile 1, and a 100 kΩ -10 kΩ resistor combination for the thermopile 2 channel. Since the test of both channels produced data in a

similar magnitude, a combined plot is used here, shown in Figure 4.6. The average measured voltage of each channel is visible in Figure 4.6a and the standard deviation for each temperature is given in Figure 4.6b. It is clearly visible how both graphs are related - their only difference is a shift by a fixed offset. This behaviour is expected because the difference in the resistor combination amplifies error mechanisms inside the measurement chain by a different factor. Since the measurement chain is shared by all channels, the error mechanisms apply to all channels of the radiometer in the same way. Therefore, a different resistor combination only introduces the visible fixed offset. Their relationship can be directly calculated from the ratio between the resistors:

$$\Delta_{channels} = \frac{\Delta_{R1}}{\Delta_{R2}} = \frac{100\ \mathrm{k} - 1\ \mathrm{k}}{100\ \mathrm{k} - 10\ \mathrm{k}} = 110\% \tag{4.10}$$

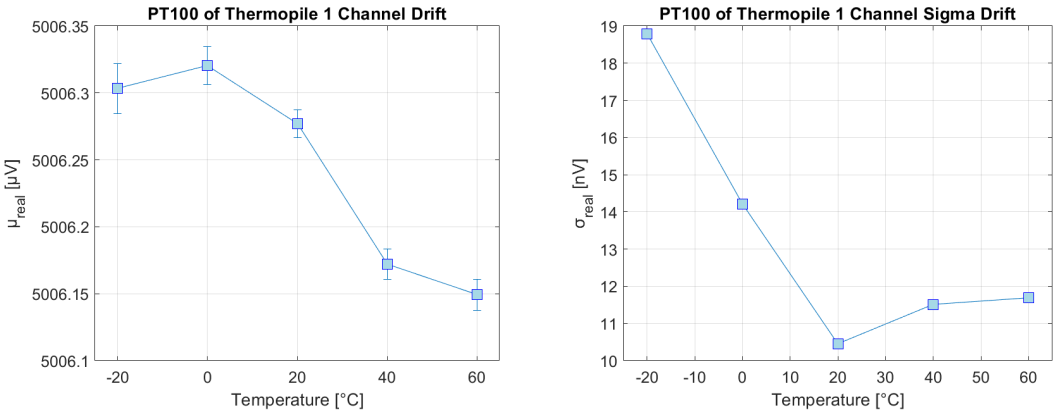
This theoretical relationship applies well to the measured one because the measured voltages on the thermopile 1 channel are roughly 10% more than the voltages on the thermopile 2 channel.

Due to the asymmetrical assembly, asymmetrical currents inside the measurement chain are amplified by the measurement. There are different sources of asymmetrical currents in the measurement chain, most notably the bias currents of the operational amplifiers in the instrumentation amplifier and leakage currents in the multiplexer. At lower temperatures, the offset voltage drifts in a positive direction, while at a higher temperature of 20 °C, this drift is reserved and the voltage drifts in a negative direction. At the highest measured temperature of 60 °C, the positive drift is the strongest. It appears that the positive error mechanism is scaling differently and has a higher magnitude than the negative error mechanism. For example, the positive error mechanism could be scaling like a second degree polynomial, while the negative error mechanism is scaling linearly. Both errors are then added. This would roughly lead to the observed behaviour in the measured voltage. Therefore, it can be concluded that there are at least two opposing error sources. The majority of these error sources lie in the shared measurement chain.

A more detailed analysis of the source of these errors is future work. Since all measured offset voltages and voltage drifts of the symmetrically assembled channels are clearly in the expected range stated by the ADC, a more detailed source analysis is not of importance for the development of this radiometer.

The visible behaviour of noise in Figure 4.6b shows that the measured noise is most likely a result from the measurement chain, since both measured channels have a similar noise characteristic, only shifted by the offset calculated in Eq. 4.10. The noise measurements of the other symmetrical channels, like the thermopile 3 or the offset channel, showed an excellent noise performance as a function of temperature. Therefore, the behaviour of noise on asymmetrical assembled channels is not analyzed further. Later, the asymmetrical resistor configuration is replaced with symmetrical 10 kΩ resistors inside the thermopiles. It is then assumed that the noise will be of a very small magnitude of 10 nV - 30 nV, as shown in previous tests. Since the previous measurements also showed that the noise is also nearly constant as a function of temperature, it can be expected that the noise on these thermopile 1 and 2 channels will also be constant.

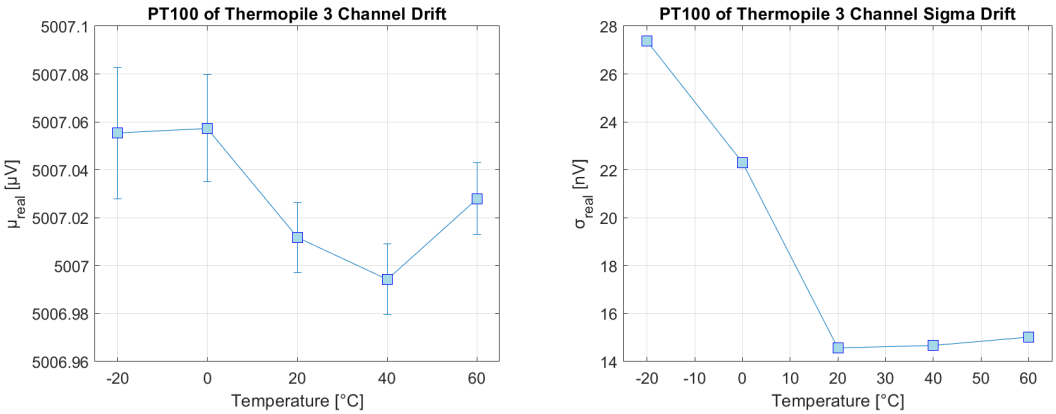
4.2.3. Internal Pt100 Channels



(a) Temperature drift of average measured voltages as a function of temperature

(b) Temperature drift of standard deviation as a function of temperature

Figure 4.7.: Pt100 of thermopile 1, substituted with a 100 Ω resistor, its temperature drift of average voltage and corresponding standard deviation



(a) Temperature drift of average measured voltages as a function of temperature

(b) Temperature drift of standard deviation as a function of temperature

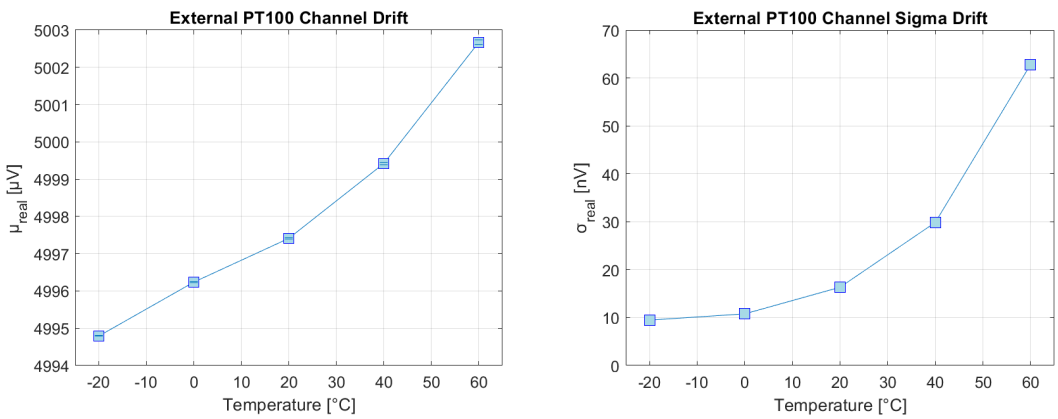
Figure 4.8.: Pt100 of thermopile 3, substituted with a 100 Ω resistor, its temperature drift of average voltage and corresponding standard deviation

For this experiment, the Pt100s of thermopile 1 - 3 were each substituted by a high precision 100 Ω resistor with 0.1% tolerance and low TC of ± 5 ppm / °C. Valid data for the Pt100 of thermopile 2 was not acquired in this test, since at this point, the corresponding pin of the ADC was not soldered correctly to the PCB. This pin was floating and therefore only noise data was measured by it. For this reason, plots of this channel are not included in this thesis. After performing the temperature drift measurement, the pin was resoldered correctly and this channel behaved similarly to the other two Pt100 of thermopile 1 and 3 channels. Therefore, it is expected that the Pt100 of thermopile 2 is working correctly.

The average measured voltage of both channels, as shown in Figure 4.7a and Figure 4.8a, are behaving similarly. Here, the measured voltages are 5006.23 µV ± 0.1 µV for the Pt100 of thermopile 1, and 5007.02 µV ± 0.04 µV for the Pt100 of thermopile 3 channel. This magnitude is expected and indicates correct functionality since the calculated theoretical measured voltage of each channel is 4990 µV (Eq. 4.7). Therefore, the differences in voltage is 0.32% for the Pt100 of thermopile 1, and 0.34% for the Pt100 of thermopile 3 channel. This small difference is negligible. It is believed that the difference of this voltage is caused by the 50 kΩ current-limiting resistor in the Pt100 circuit, since it was found that these resistors were not assembled correctly. Each 50 kΩ resistor has a probable tolerance of 1%, which should be lower in the circuit. In the future, all 50 kΩ resistors have to be replaced with low tolerance 0.1% and low temperature coefficient resistors. For both channels, the average measured voltage is almost constant over the measured temperature range. This is also expected, since the used 100 Ω resistors have a very low TC of 5 ppm / °C. Therefore, their resistance over temperature is nearly constant, resulting in the observed and almost constant behaviour of the measured voltage.

Regarding the noise levels at different temperatures in Figure 4.7b and Figure 4.8b, it can be observed that the noise levels are between 10 nV and 28 nV, which is very small and practically constant. This behaviour is also observed in all other channels which provide valid data, like the offset channel or the thermopile 1 channel. The maximum noise of the Pt100 of thermopile 3 channel is 28 nV, which is larger when compared with the other channels, but is still small.

External Pt100 Channel



(a) Temperature drift of average measured voltages as a function of temperature

(b) Temperature drift of standard deviation as a function of temperature

Figure 4.9.: External Pt100, substituted with a low precision high TC 100 Ω resistor, its temperature drift of average voltage and corresponding standard deviation

In this test, the external Pt100 channel was replaced, like the other internal Pt100 channels, with a 100 Ω resistor instead of the later used Pt100. An important difference to note is that the used 100 Ω resistor is not a low tolerance resistor here. Instead, a generic 0805 SMD resistor with a tolerance of 1% and an unknown, potentially high, temperature coefficient was

used. For this reason the measured results of the temperature drift have a different characteristic than the tests of the internal Pt100 channel, which used high precision low TC 100 Ω resistors instead.

Figure 4.9a shows the average measured voltage as a function of temperature of this channel. The measured voltages are between 4995 µV and 5002.8 µV. This is expected since the calculated voltage across the 100 Ω resistor should be 4990 µV (Eq. 4.7). The average measured voltages are therefore in the correct order of magnitude. In the worst case, the deviation from the theoretical voltage is (5002.8 µV - 4990 µV) / 5002.8 µV ≈ 0.256%. This difference is small and indicates that the channel is working correctly. Additionally, it is visible that the voltage is approximately linear to the temperature. Here, the high temperature coefficient of the used 100 Ω resistor can be observed. Since the resistor increases its resistance at higher temperatures, the voltage drop across it increases, which is then measured. The observed temperature drift corresponds to 25 ppm / °C, which is typical for standard SMD 0805 resistors.

The noise performance of this channel is shown in Figure 4.9b. Here it can be observed that the noise increases from 10 nV to 60 nV at higher temperatures. This high degree of noise at a higher temperature is not expected. The channel is expected to have a similar noise characteristic as the internal Pt100 channels (see in Chapter 4.2.3), which would be around 20 nV and relatively constant. The high noise drift could be caused by the high tolerance 100 Ω resistor itself, since the noise levels at the high precision resistors of Pt100 thermopile 1 + 3 are almost constant, as seen in the previous tested channels (Figure 4.7b and Figure 4.8b). However, as the voltage across the Pt100 is around 5000 µV and the noise level is 60 nV, the noise level will have practically no influence on the measured voltage across the Pt100.

5. Summary and Future Work

5.1. Summary

In this thesis a functional small and light-weight multi-channel radiometer for application on a Mars Weather balloon was successfully developed and tested. This radiometer will be used in the Mars Weather Balloon project of the DLR Institute of Space Systems in Bremen to demonstrate the functionality of the payload.

The radiometer itself hosts three thermopile sensors, measuring radiation in different wavelength bands. These are centered around 15 µm for measuring the average temperature of the air column, 9.35 µm in order to determine the aerosol (dust) maximum, which is a proxy for the opacity of the atmosphere, and 7.94 µm, from which the surface temperature of Mars can be derived [1, pp. 5–6]. The radiometer contains a Pt100 inside each thermopile housing, as well as a Pt100 integrated in an aluminium body surrounding the three sensors, measuring the respective ambient temperature. These temperatures are necessary in order to determine the net received radiation for each channel. In order to remove systematic errors, an additional offset measurement channel is included. With data from this channel it is possible to cancel out offset voltages introduced by the measurement chain. While the schematics were mostly drawn from previous developed radiometers, a new PCB layout was designed and implemented. Further, a new general purpose driver for the ADS1258 was developed, as well as a custom driver for this radiometer. This driver is adapted to the avionics flight system of the Mars Weather balloon, and can be easily integrated into its flight computer. For testing and debugging the radiometer, a custom development board was also created. Several function tests showed that the radiometer is working as expected an all channels produce valid data. The firmware proved to be robust and reliable. Additionally, it was proven that all channels have excellent noise performance, since the noise is < 100 nV for each channel at all measured temperatures. The generated signal levels for all thermopiles are in the magnitude of several hundred µV [8, p. 16, Fig. 7] for temperatures between 123 K to 373 K. Therefore, the signal-to-noise ratio is high and the introduced noise has a negligible influence, resulting in excellent noise performance of the instrument.

5.2. Future Work

Currently, the radiometer produce valid data for all measured channels, but the device is not yet calibrated. This has to be to be able to interpret gathered data from the radiometer in terms of brightness temperature. Calibration of this radiometer will be similar as calibrating previous developed radiometers, like the HP^3 radiometer on InSight [16]. Calibration will be performed using a blackbody as calibration source. After calibration the radiometer, it will be integrated into the Mars Weather Balloon and the whole system can be tested.

Several improvements of the radiometer are possible. The radiometer has to be redesigned

hosting only space-graded components. All used components were already chosen so that there exist a space-graded equivalent, so the needed changes are minimal.

A notable difference between the in the thesis designed radiometer to previous deployed ones is the different approach of temperature stabilization of the sensors. Previous radiometers like the HP^3 are actively temperature stabilized by heater systems [16, p. 3], the in the thesis developed radiometer relies on a passive temperature stabilization instead. It should be tested how this affects performance of the instrument.

On the software side, different improvements are possible. Here, a logging system could be implemented, which would help debugging the radiometer when integrated in the flight system. Additionally, returning of different error types when the driver throws exceptions could be added in the future. This would allow for a more precise handling of errors. Regarding the power consumption of the radiometer, an energy saving mode could be integrated in the firmware. Here, the ADC could be switched to a stand-by mode when the instrument is not currently sampling, which would decrease the power consumption. Another improvement of the general-purpose ADS1258 driver would be to integrate another poll data method of the ADC, called "Direct Mode". Currently, only the "Command Mode" is supported by the ADS1258 driver. The "Direct Mode" can poll a data frame from the ADS1258 with only 3 bytes of transmitted data in the SPI protocol, while the "Command Mode" needs 5 bytes for polling the same data frame. This could be interesting if a high sample rate of the ADS1258 should be used in future projects. However, for lower data rates, like used in this radiometer, the slower "Command Mode" is adequate.

Bibliography

[1] L. Witte, G. Arnold, J. Bertram *et al.*, 'A concept for a mars boundary layer sounding balloon: Science case, technical concept and deployment risk analysis,' *Aerospace*, vol. 9, no. 3, 2022, ISSN: 2226-4310. DOI: 10.3390/aerospace9030136.

[2] Banfield, D. (Ed.), 'Mepag, mars scientific goals, objectives, investigations, and priorities: 2018,' [Online]. Available: https://mepag.jpl.nasa.gov/reports/MEPAG%20Goals_Document_2018.pdf.

[3] A. D. Toigo, C. Lee, C. E. Newman and M. I. Richardson, 'The impact of resolution on the dynamics of the martian global atmosphere: Varying resolution studies with the marswrf gcm,' *Icarus*, vol. 221, no. 1, pp. 276–288, 2012, ISSN: 0019-1035. DOI: 10.1016/j.icarus.2012.07.020.

[4] S. Rafkin and T. Michaels, 'The mars regional atmospheric modeling system (mrams): Current status and future directions,' *Atmosphere*, vol. 10, no. 12, p. 747, Nov. 2019, ISSN: 2073-4433. DOI: 10.3390/atmos10120747.

[5] J. Pla-García, S. C. R. Rafkin, G. M. Martinez *et al.*, 'Meteorological predictions for mars 2020 perseverance rover landing site at jezero crater,' *Space Science Reviews*, vol. 216, no. 8, p. 148, Dec. 2020, ISSN: 1572-9672. DOI: 10.1007/s11214-020-00763-x.

[6] S. C. R. Rafkin and T. I. Michaels, 'Meteorological predictions for 2003 mars exploration rover high-priority landing sites,' *Journal of Geophysical Research: Planets*, vol. 108, no. E12, 2003. DOI: 10.1029/2002JE002027.

[7] T. Spohn, M. Grott, S. E. Smrekar *et al.*, 'The heat flow and physical properties package (hp3) for the insight mission,' eng, *Space science reviews*, vol. 214, no. 5, pp. 1–33, 2018, ISSN: 0038-6308. DOI: 10.1007/s11214-018-0531-4.

[8] M. Grott, J. Knollenberg, B. Borgs *et al.*, 'The mascot radiometer mara for the hayabusa 2 mission,' *Springer*, 2017. DOI: 10.1007/s11214-016-0272-1.

[9] P. Michel, S. Ulamec, U. Böttger *et al.*, 'The mmx rover: Performing in situ surface investigations on phobos,' *Earth, Planets and Space*, vol. 74, no. 1, p. 2, Jan. 2022, ISSN: 1880-5981. DOI: 10.1186/s40623-021-01464-7.

[10] W. R. McCluney, *Introduction to radiometry and photometry* (The Artech House optoelectronics library), eng. Boston u.a.: Artech House, 1994, ISBN: 0890066787.

[11] Eric W. Weisstein, *Radiation - wolfram research encyclopedia*, https://scienceworld.wolfram.com/physics/Radiation.html, Accessed: 2023-02-04.

[12] E. Tiesinga, P. J. Mohr, D. B. Newell and B. N. Taylor, 'Codata recommended values of the fundamental physical constants: 2018,' *Rev. Mod. Phys.*, vol. 93, p. 025 010, 2 Jun. 2021. DOI: 10.1103/RevModPhys.93.025010.

[13] C. Dames, 'Resistance temperature detectors,' in *Encyclopedia of Microfluidics and Nanofluidics*, D. Li, Ed. Boston, MA: Springer US, 2008, pp. 1782–1790, ISBN: 978-0-387-48998-8. DOI: 10.1007/978-0-387-48998-8_1354.

[14] S. Paul and R. Paul, 'Analogien zwischen elektrischen und nichtelektrischen systemen,' in *Grundlagen der Elektrotechnik und Elektronik 2: Elektromagnetische Felder und ihre Anwendungen*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019, pp. 593–664, ISBN: 978-3-662-58221-3. DOI: 10.1007/978-3-662-58221-3_6.

[15] J. Hartmann and F. Bernhard, 'Strahlungstemperaturmessung,' in *Handbuch der Technischen Temperaturmessung*, F. Bernhard, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 1161–1393, ISBN: 978-3-642-24506-0. DOI: 10.1007/978-3-642-24506-0_13.

[16] N. T. Mueller, J. Knollenberg, M. Grott *et al.*, 'Calibration of the hp3 radiometer on insight,' *Earth and Space Science*, vol. 7, no. 5, e2020EA001086, 2020, e2020EA001086 10.1029/2020EA001086. DOI: 10.1029/2020EA001086.

[17] Analog Devices Incorporated, *Aducrf101 (ref. a) datasheet*, https://www.analog.com/media/en/technical-documentation/data-sheets/ADuCRF101.pdf, Accessed: 2023-02-05.

[18] Texas Instruments Incorporated, *16-channel, 24-bit, analog-to-digital converter datasheet (rev. d), ads1258-ep*, https://www.ti.com/lit/ds/symlink/ads1258-ep.pdf, Accessed: 2023-02-05.

[19] L. Safari, G. Ferri, S. Minaei and V. Stornelli, 'Principles of instrumentation amplifiers,' in *Current-Mode Instrumentation Amplifiers*. Cham: Springer International Publishing, 2019, pp. 1–13, ISBN: 978-3-030-01343-1. DOI: 10.1007/978-3-030-01343-1_1.

[20] Renesas Electronics Corporation, *Isl2817 isl28227 isl28227seh datasheet (rev 8.00)*, https://www.renesas.com/us/en/document/dst/isl28127-isl28227-isl28227seh-datasheet?r=494006, Accessed: 2023-02-05.

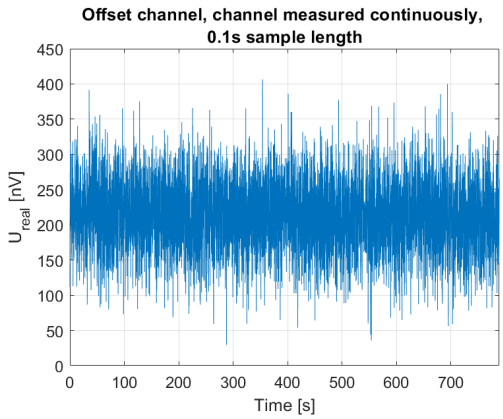
[21] Analog Devices Incorporated, *Ref43 +2.5v low power precision voltage reference data sheet (rev. d)*, https://www.analog.com/media/en/technical-documentation/data-sheets/REF43.pdf, Accessed: 2023-02-05.

[22] S. H. K. M., 'Standard deviation,' in *International Encyclopedia of Statistical Science*, M. Lovric, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1378–1379, ISBN: 978-3-642-04898-2. DOI: 10.1007/978-3-642-04898-2_535.

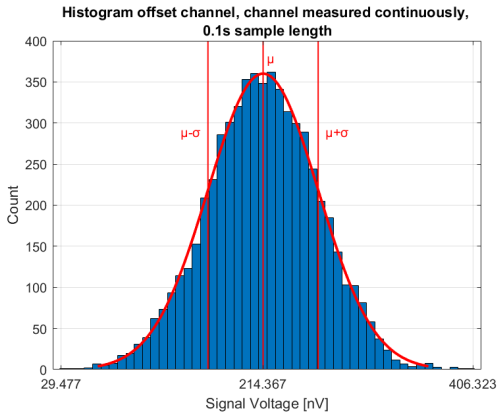
[23] T. J. Rouphael, 'Chapter 3 - noise in wireless receiver systems,' eng, in *Wireless Receiver Architectures and Design*, Elsevier Inc, 2015, pp. 105–178, ISBN: 9780123786401.

A. Appendix

A.1. Data Experiment 1 - Measurement Configuration Test

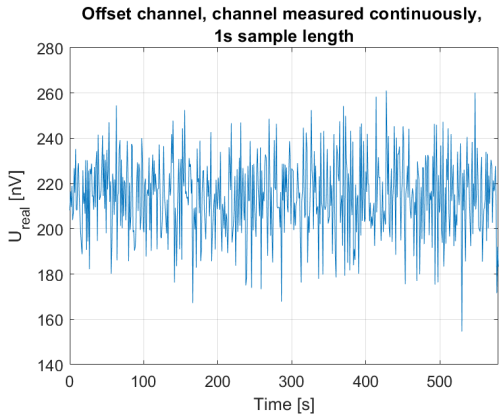


(a) Measured voltage of the offset channel plotted over time

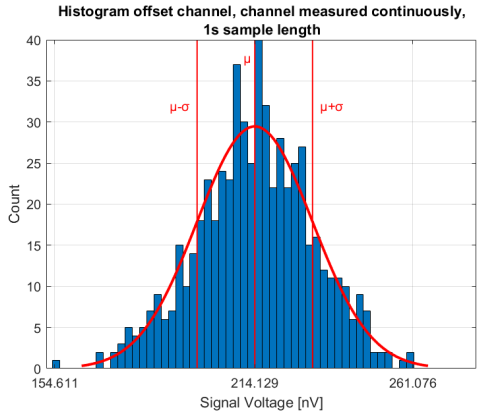


(b) Corresponding histogram

Figure A.1.: Offset channel, measured continuously with a 0.1 s sample length
Sample length: 0.1 s
Sample discard: 0
Number samples taken: 6000
μ_{real} = 214 nV
σ_{real} = 50 nV

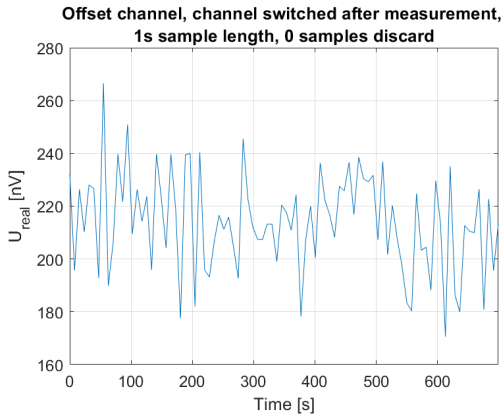


(a) Measured voltage of the offset channel plotted over time

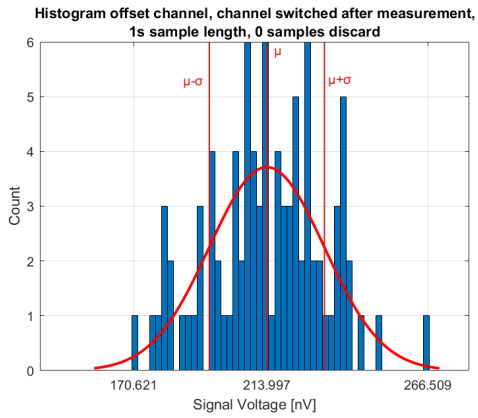


(b) Corresponding histogram

Figure A.2.: Offset channel, measured continuously with a 1 s sample length
Sample length: 1 s
Sample discard: 0
Number samples taken: 600
μ_{real} = 214 nV
σ_{real} = 17 nV

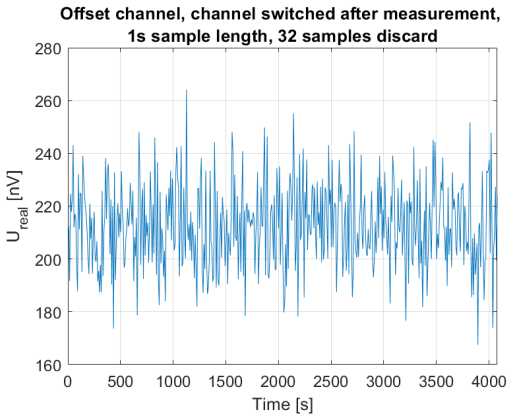


(a) Measured voltage of the offset channel plotted over time

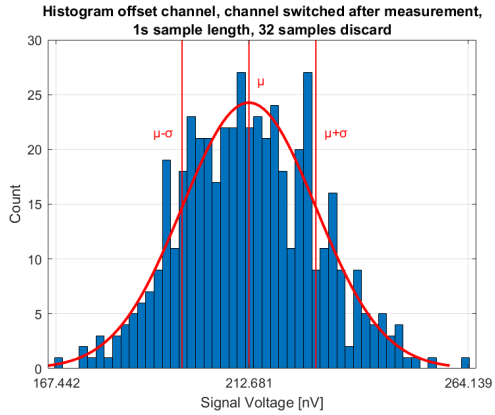


(b) Corresponding histogram

Figure A.3.: Offset channel, channel is switched after measurement, 1 s sample length, 0 sample discard
Sample length: 1 s
Sample discard: 0
Number samples taken: 100
μ_{real} = 214 nV
σ_{real} = 19 nV



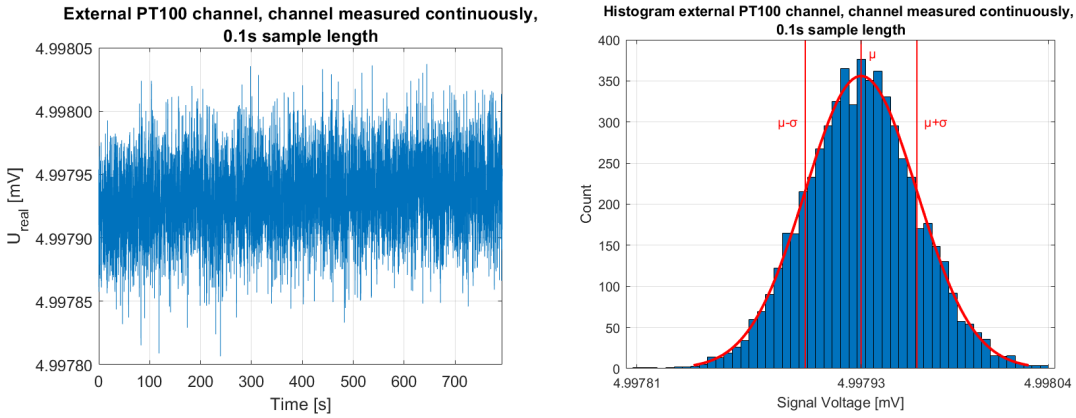
(a) Measured voltage of the offset channel plotted over time



(b) Corresponding histogram

Figure A.4.: Offset channel, channel is switched after measurement, 1 s sample length, 32 sample discard
Sample length: 1 s
Sample discard: 32
Number samples taken: 500
μ_{real} = 213 nV
σ_{real} = 16 nV

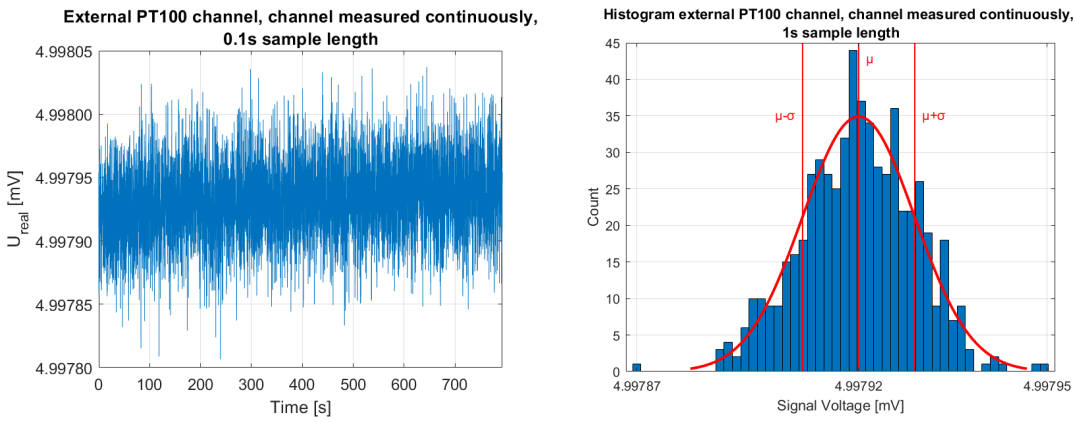
A.1.1. External Pt100 channel, replaced with a high tolerance, high TC 100 Ω resistor



(a) Measured voltage of the external Pt100 channel plotted over time

(b) Corresponding histogram

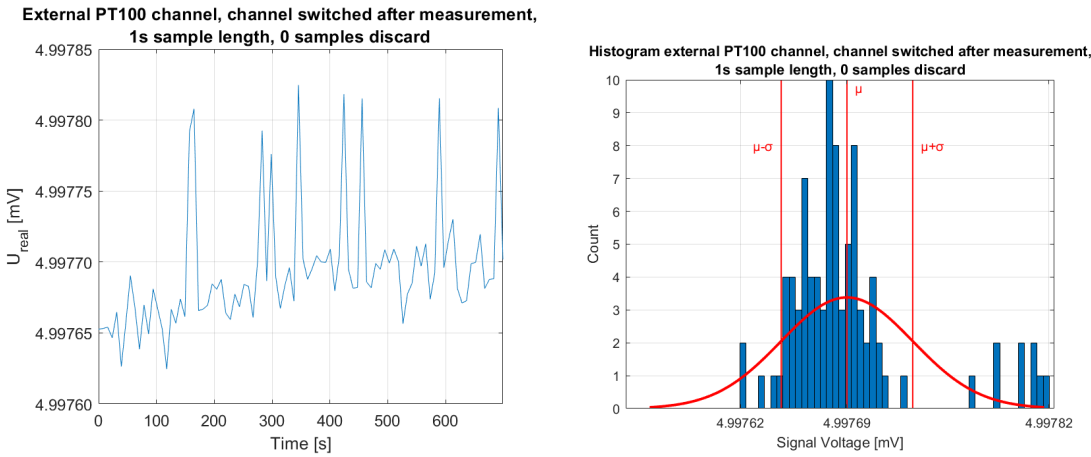
Figure A.5.: External Pt100 channel, replaced with a 100 Ω resistor, measured continuously with a 0.1 s sample length
Sample length: 0.1 s
Sample discard: 0
Number samples taken: 6000
μ_{real} = 4.998 mV
σ_{real} = 31 nV



(a) Measured voltage of the external Pt100 channel plotted over time

(b) Corresponding histogram

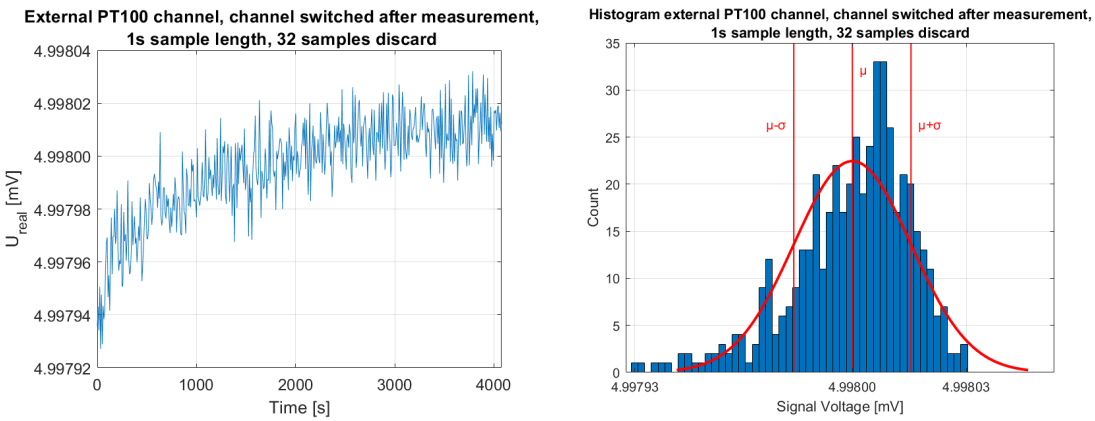
Figure A.6.: External Pt100 channel, replaced with a 100 Ω resistor, measured continuously with a 1 s sample length
Sample length: 1 s
Sample discard: 0
Number samples taken: 600
μ_{real} = 4.998 mV
σ_{real} = 11 nV



(a) Measured voltage of the external Pt100 channel plotted over time

(b) Corresponding histogram

Figure A.7.: External Pt100 channel, replaced with a 100 Ω resistor, channel is switched after measurement, 1 s sample length, 0 sample discard
Sample length: 1 s
Sample discard: 0
Number samples taken: 100
μ_{real} = 4.998 mV
σ_{real} = 44 nV



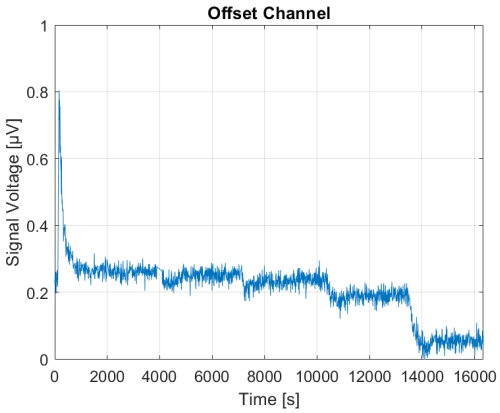
(a) Measured voltage of the external Pt100 channel plotted over time

(b) Corresponding histogram

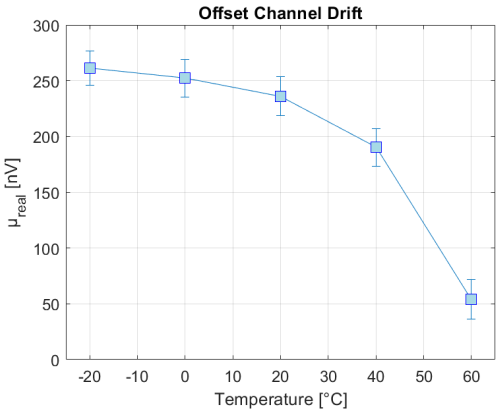
Figure A.8.: External Pt100 channel, replaced with a 100 Ω resistor, channel is switched after measurement, 1 s sample length, 32 sample discard
Sample length: 1 s
Sample discard: 32
Number samples taken: 500
μ_{real} = 4.998 mV
σ_{real} = 11 nV

A.2. Data Experiment 2 - Temperature Drifts

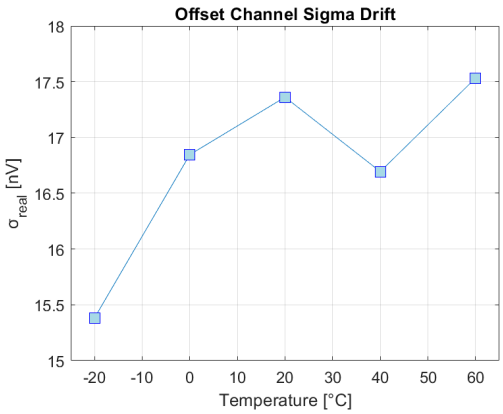
A.2.1. Offset Channel



(a) Measured voltage of the offset channel plotted over time, temperature changes every 30 min by 20 °C, starting at -20 °C



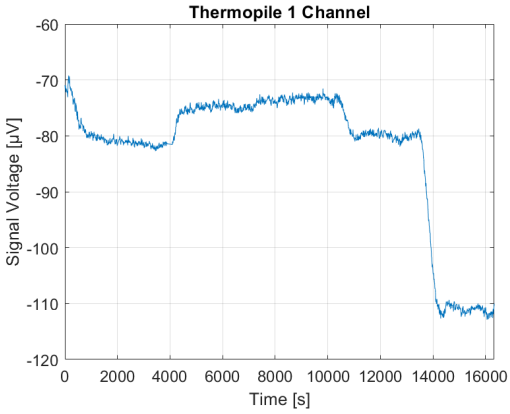
(b) Corresponding drift of average voltages as a function of temperature



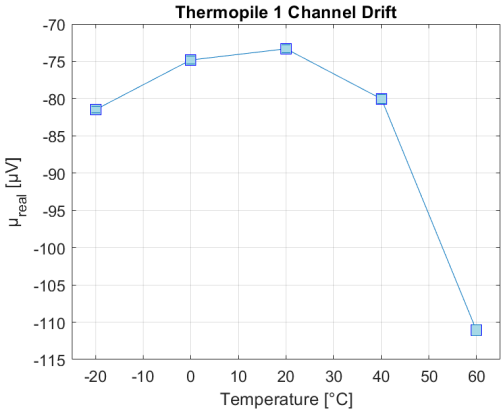
(c) Corresponding drift of the standard deviation as a function of temperature

Figure A.9.: Measured temperature drift, corresponding average voltage per temperature and corresponding standard deviation of offset channel, from -20 °C to 60 °C in steps of 20 °C

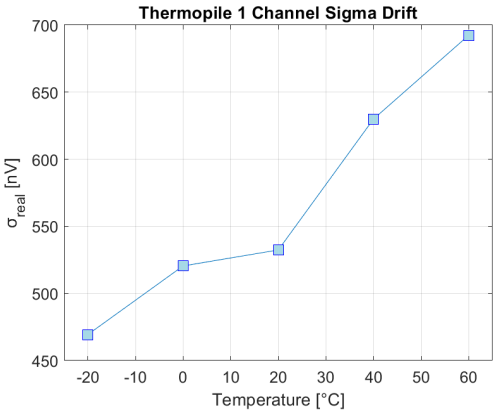
A.2.2. Thermopile 1 Channel (100 kΩ - 1 kΩ resistor combination)



(a) Measured voltage of the thermopile 1 channel plotted over time, temperature changes every 30 min by 20 °C, starting at -20 °C



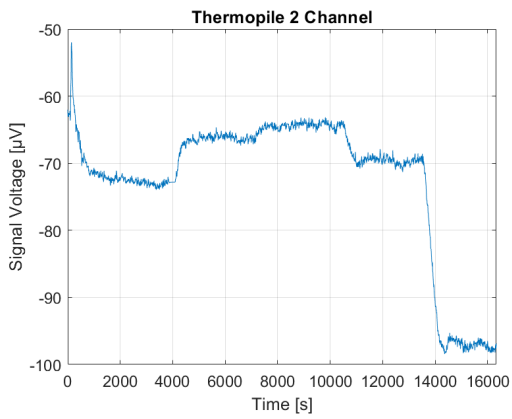
(b) Corresponding drift of average voltages as a function of temperature



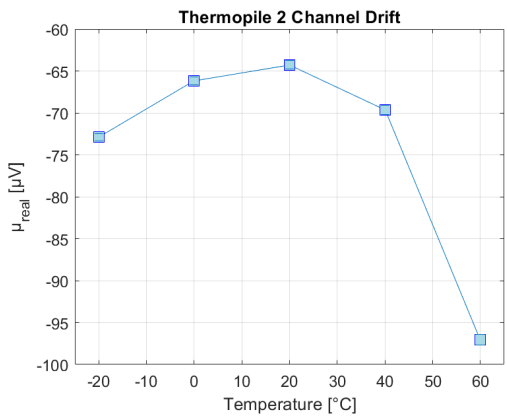
(c) Corresponding drift of the standard deviation as a function of temperature

Figure A.10.: Measured temperature drift, corresponding average voltage per temperature and corresponding standard deviation of the thermopile 1 channel, from -20 °C to 60 °C in steps of 20 °C

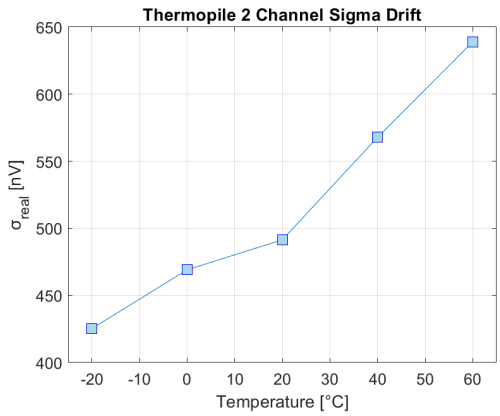
A.2.3. Thermopile 2 Channel (100 kΩ - 10 kΩ resistor combination)



(a) Measured voltage of the thermopile 2 channel plotted over time, temperature changes every 30 min by 20 °C, starting at -20 °C



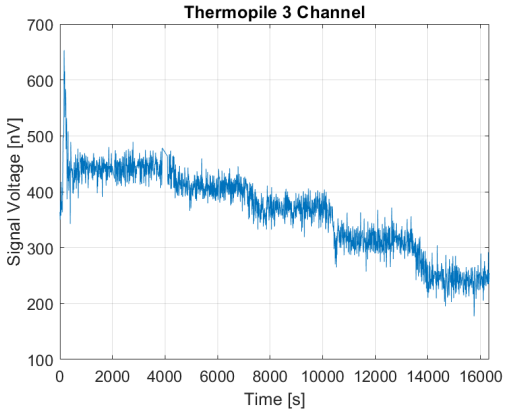
(b) Corresponding drift of average voltages as a function of temperature



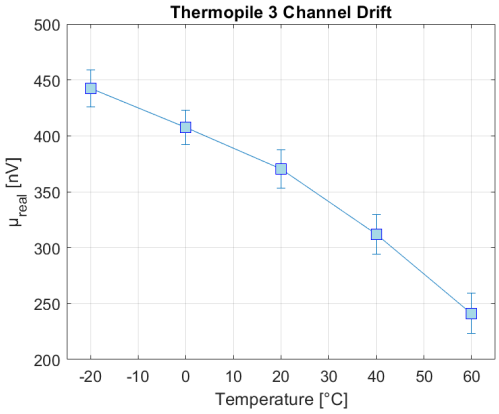
(c) Corresponding drift of the standard deviation as a function of temperature

Figure A.11.: Measured temperature drift, corresponding average voltage per temperature and corresponding standard deviation of the thermopile 2 channel, from -20 °C to 60 °C in steps of 20 °C

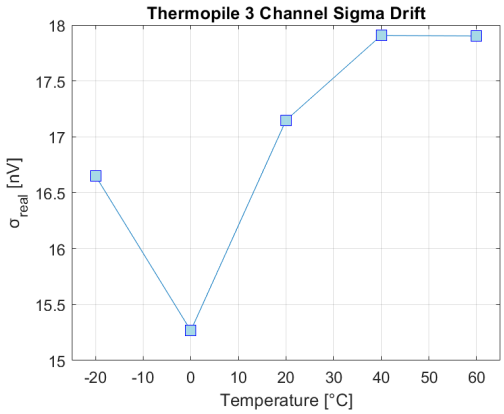
A.2.4. Thermopile 3 Channel (10 kΩ - 10 kΩ resistor combination)



(a) Measured voltage of the thermopile 3 channel plotted over time, temperature changes every 30 min by 20 °C, starting at -20 °C



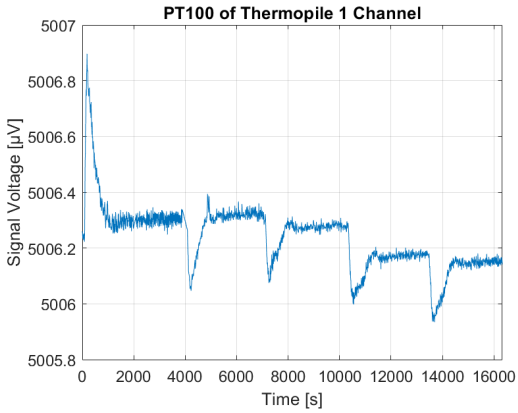
(b) Corresponding drift of average voltages as a function of temperature



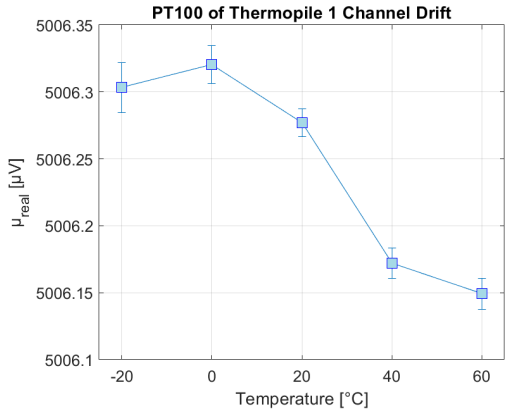
(c) Corresponding drift of the standard deviation as a function of temperature

Figure A.12.: Measured temperature drift, corresponding average voltage per temperature and corresponding standard deviation of the thermopile 3 channel, from -20 °C to 60 °C in steps of 20 °C

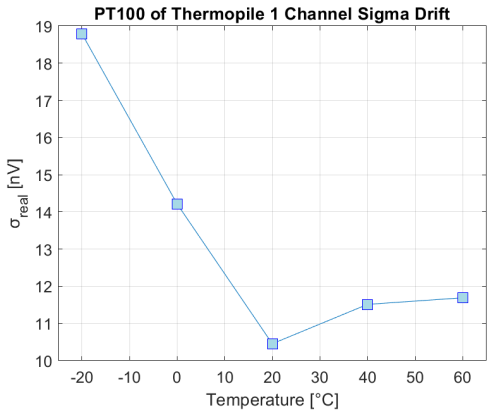
A.2.5. Pt100 of Thermopile 1 Channel (substituted with a ±0.1% precision 5 ppm / °C TC 100 Ω resistor)



(a) Measured voltage of the Pt100 of thermopile 1 channel plotted over time, temperature changes every 30 min by 20 °C, starting at -20 °C



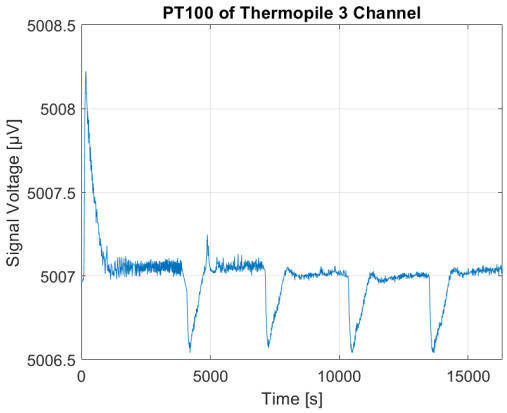
(b) Corresponding drift of average voltages as a function of temperature



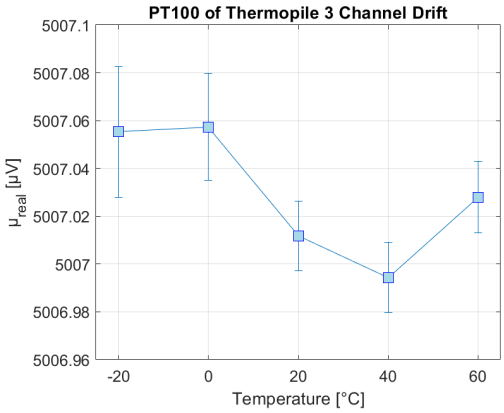
(c) Corresponding drift of the standard deviation as a function of temperature

Figure A.13.: Measured temperature drift, corresponding average voltage per temperature and corresponding standard deviation of the Pt100 of thermopile 1 channel, from -20 °C to 60 °C in steps of 20 °C

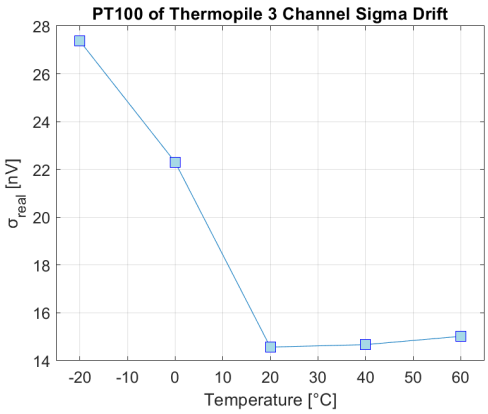
A.2.6. Pt100 of Thermopile 3 Channel (substituted with a ±0.1% precision 5 ppm / °C TC 100 Ω resistor)



(a) Measured voltage of the Pt100 of thermopile 3 channel plotted over time, temperature changes every 30 min by 20 °C, starting at -20 °C



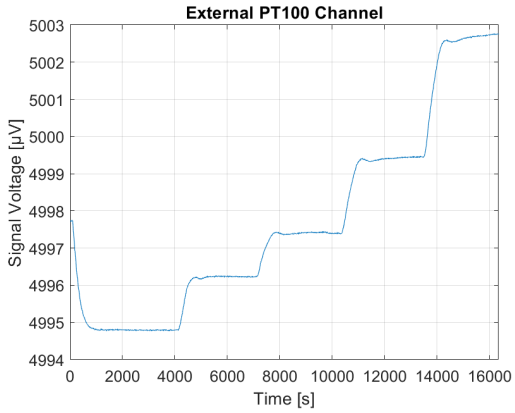
(b) Corresponding drift of average voltages as a function of temperature



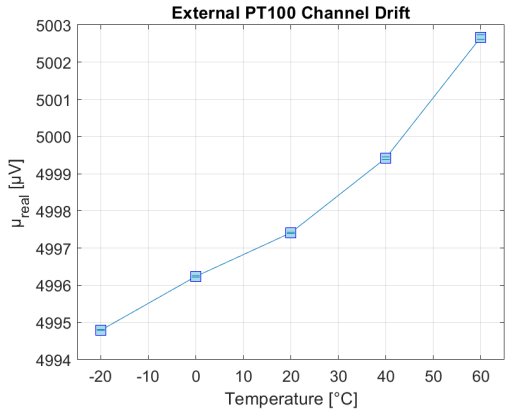
(c) Corresponding drift of the standard deviation as a function of temperature

Figure A.14.: Measured temperature drift, corresponding average voltage per temperature and corresponding standard deviation of the Pt100 of thermopile 3 channel, from -20 °C to 60 °C in steps of 20 °C

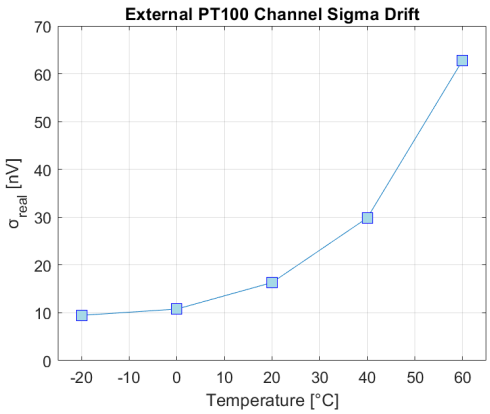
A.2.7. External Pt100 channel (substituted with a generic 100 Ω, high TC and high tolerance resistor)



(a) Measured voltage of the external Pt100 channel plotted over time, temperature changes every 30 min by 20 °C, starting at -20 °C



(b) Corresponding drift of average voltages as a function of temperature

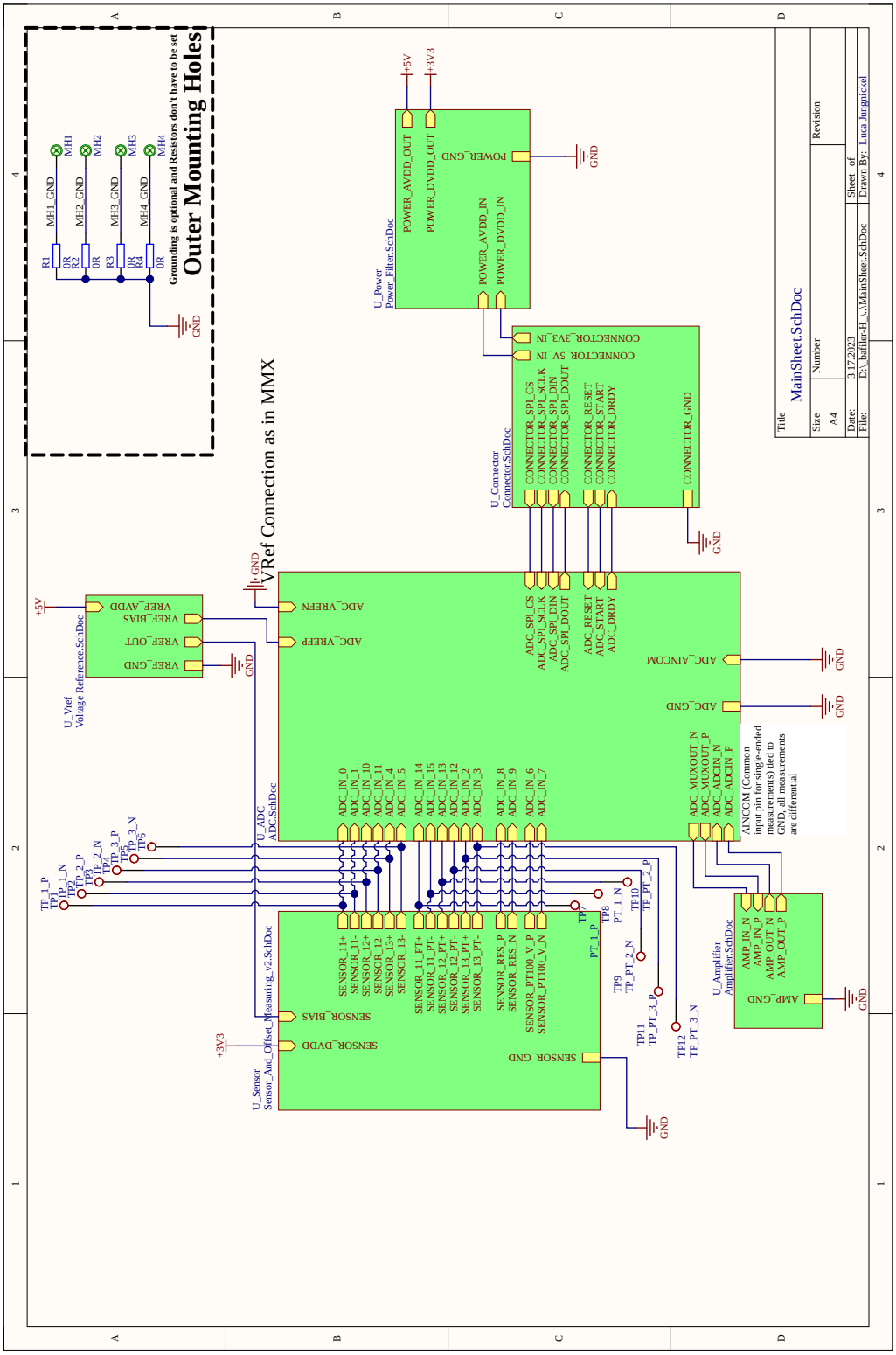


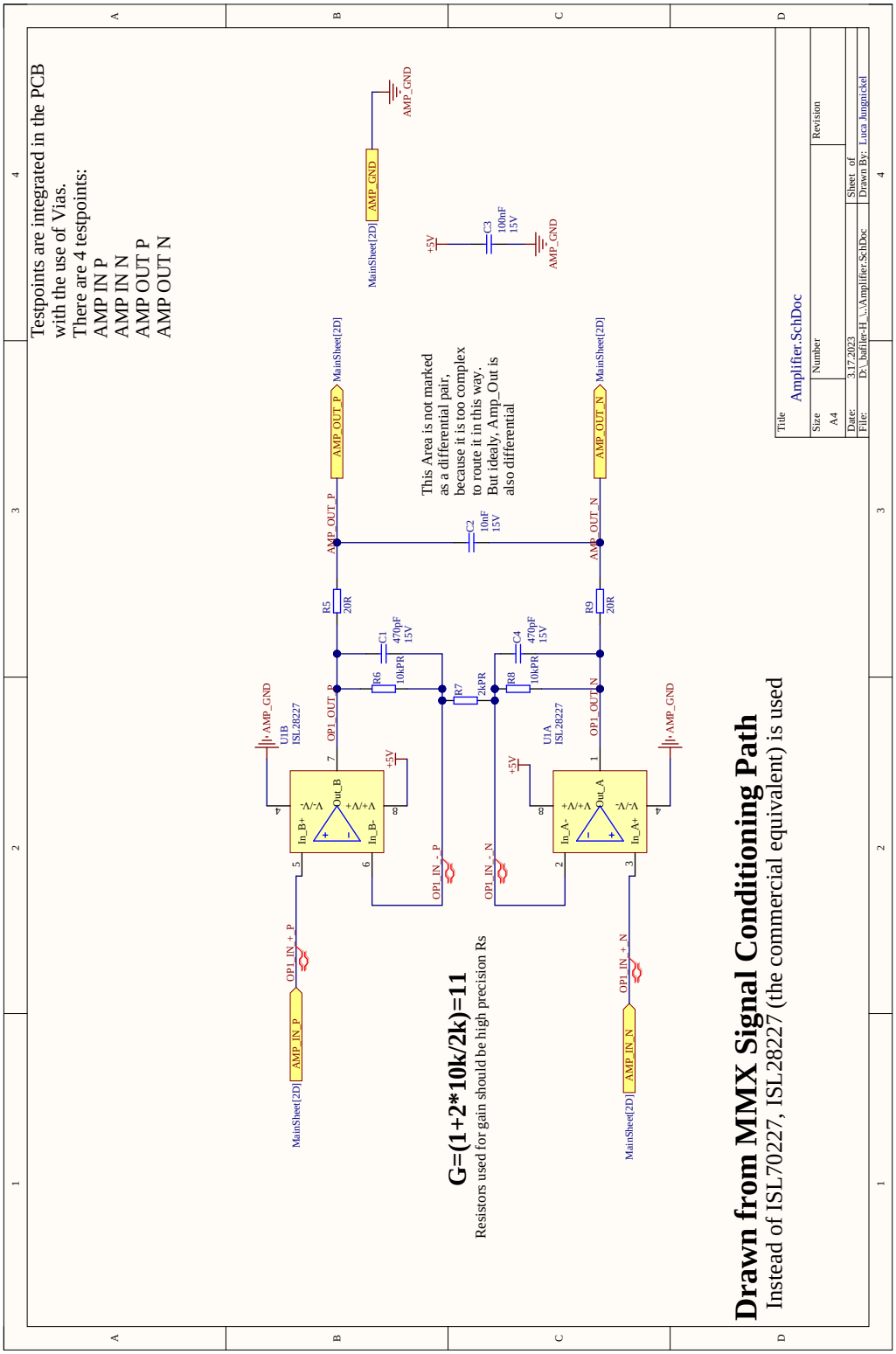
(c) Corresponding drift of the standard deviation as a function of temperature

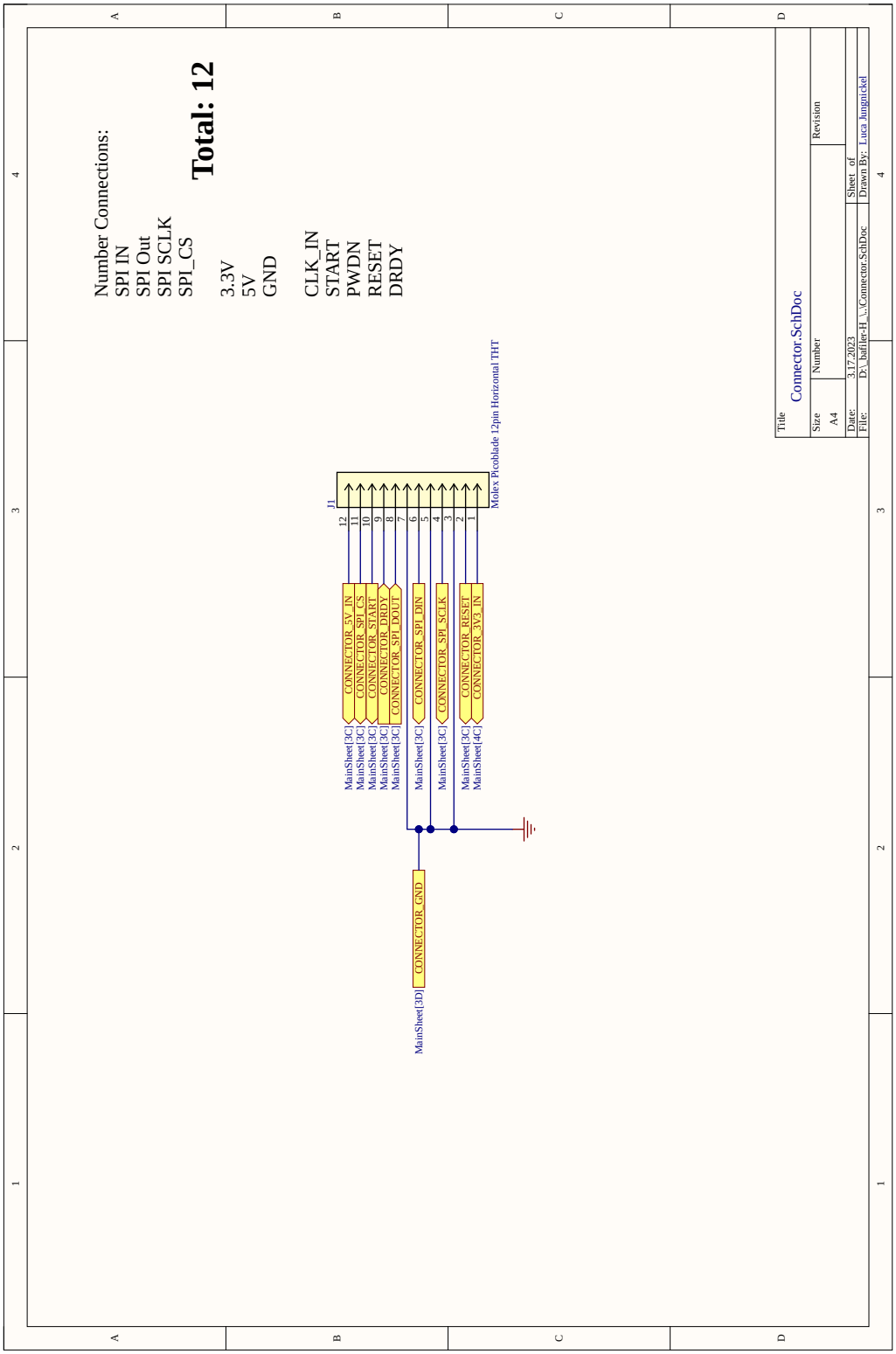
Figure A.15.: Measured temperature drift, corresponding average voltage per temperature and corresponding standard deviation of the external Pt100 channel, from -20 °C to 60 °C in steps of 20 °C

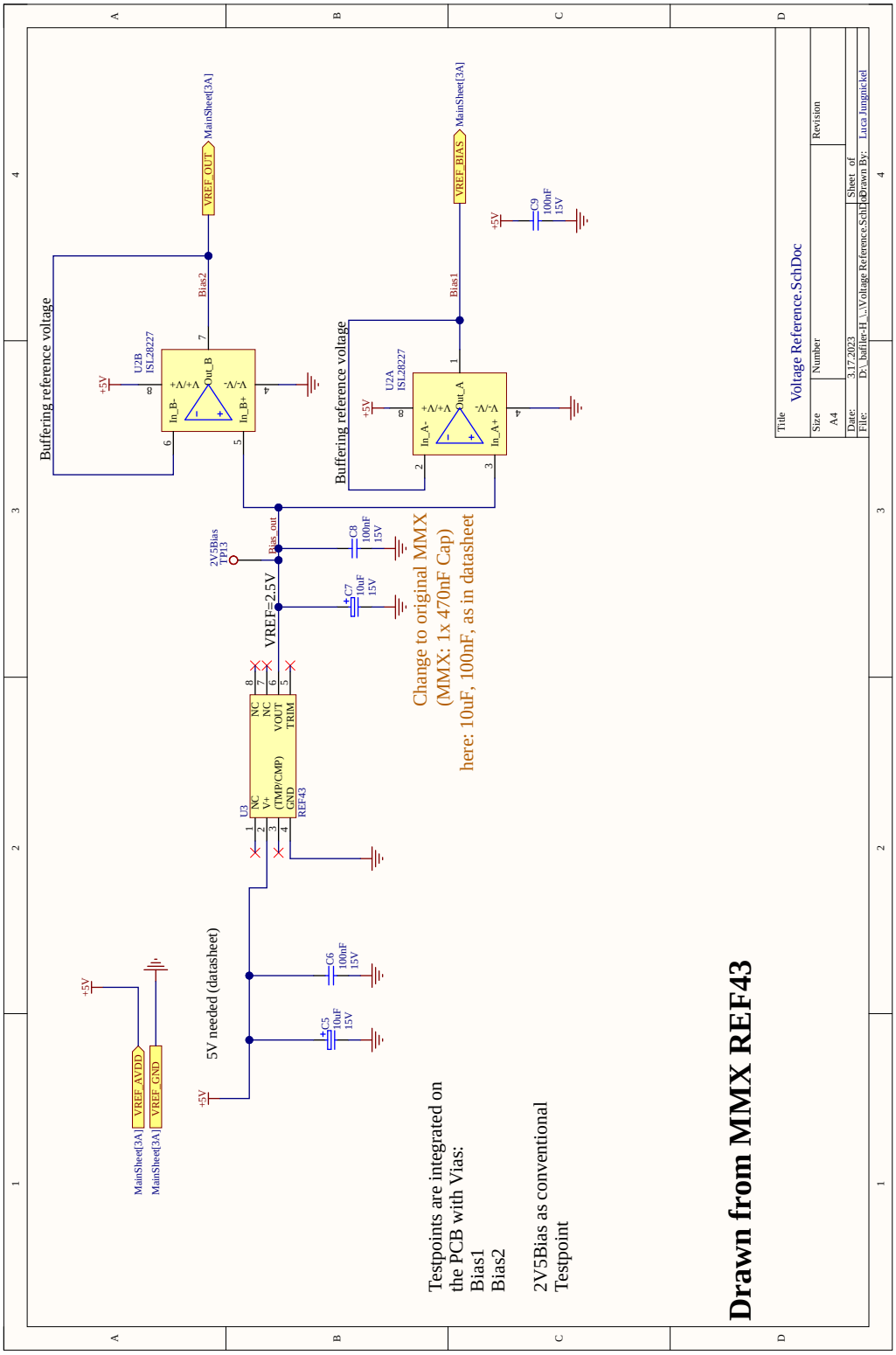
A.3. MableRad - Schematics and PCB Design

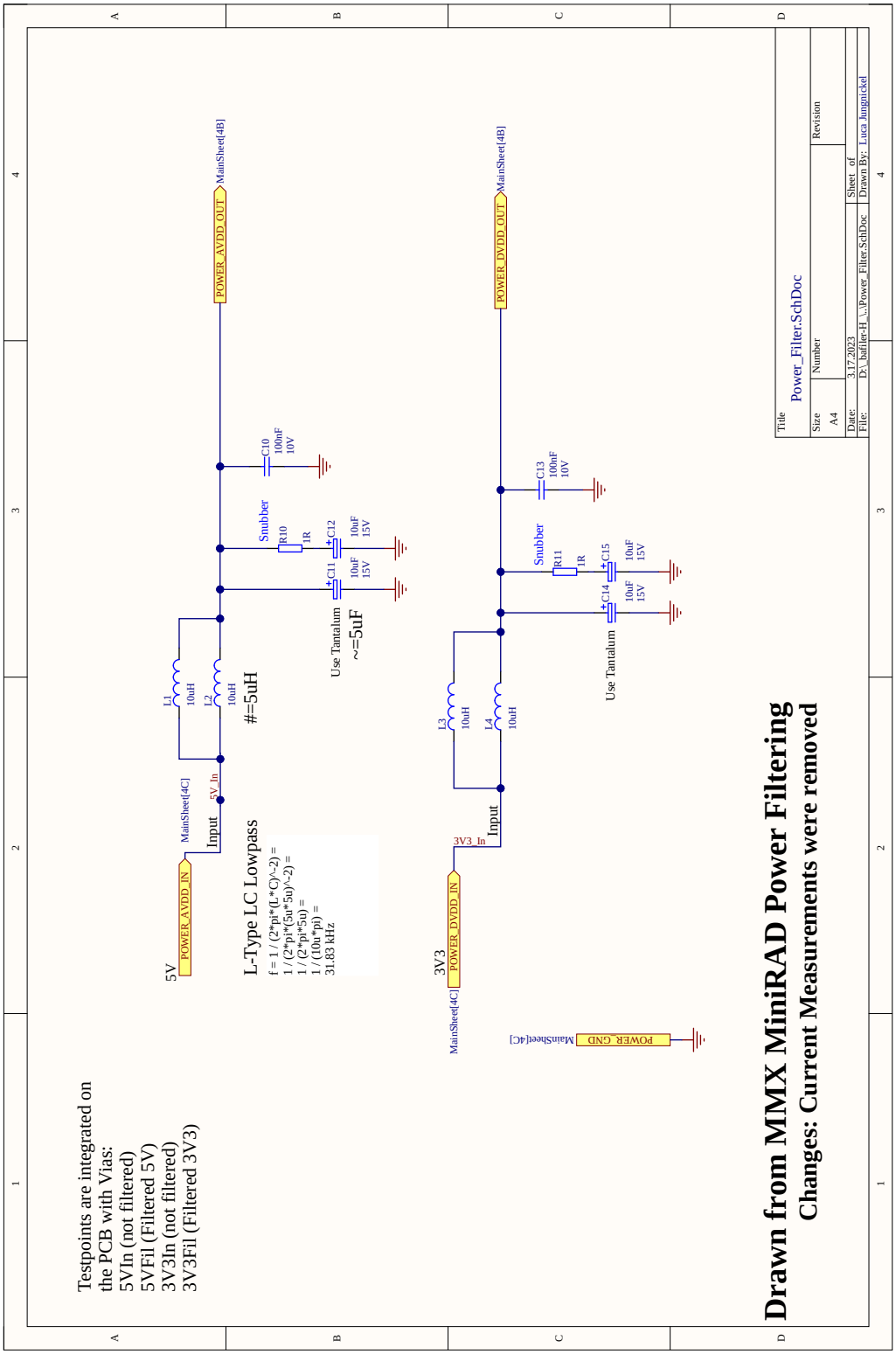
A.3.1. Schematics

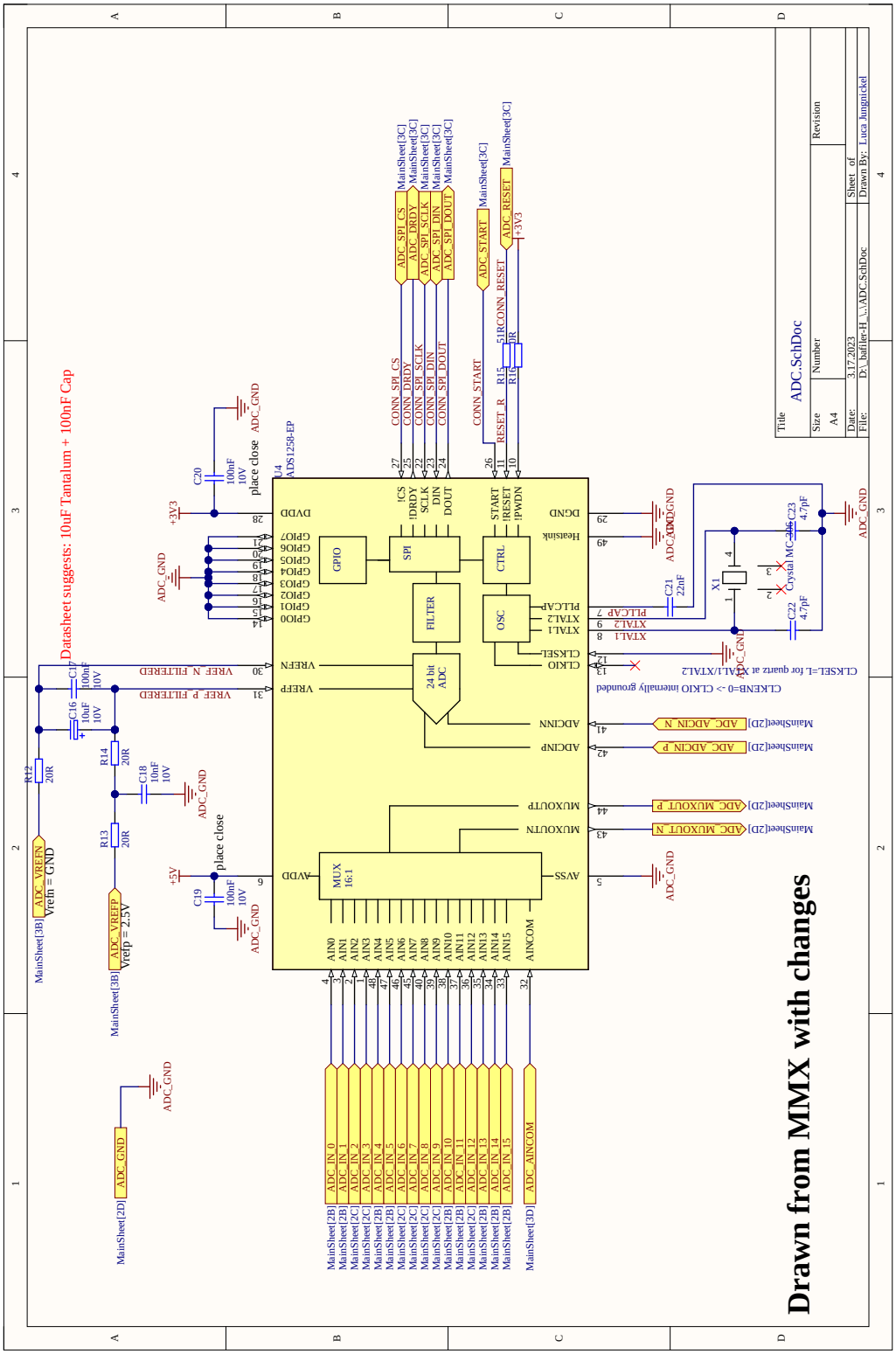


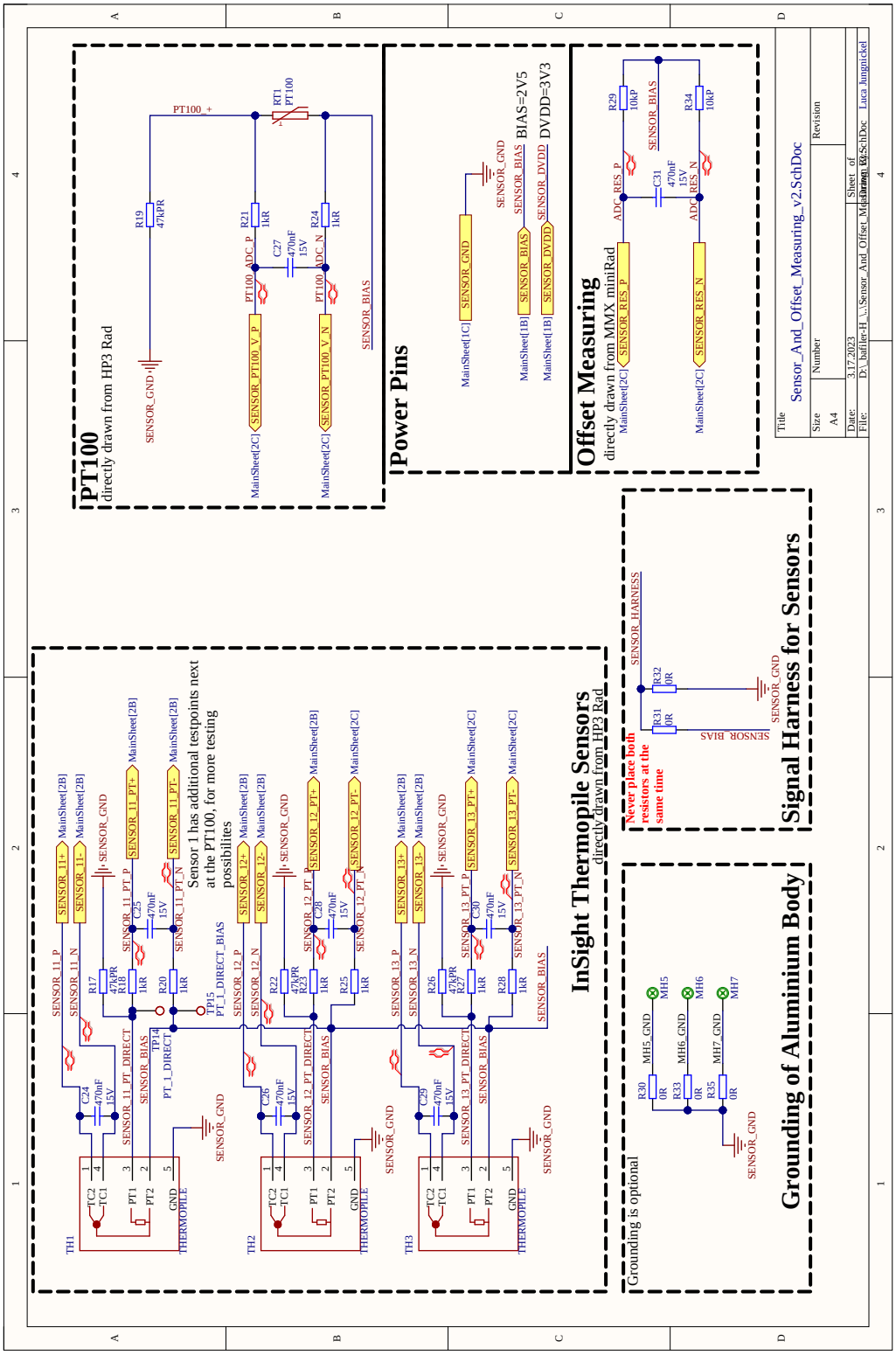

Testpoints are integrated in the PCB
with the use of Vias.
There are 4 testpoints:
AMP IN P
AMP IN N
AMP OUT P
AMP OUT N
This Area is not marked
as a differential pair,
because it is too complex
to route it in this way.
But idealy, Amp_Out is
also differential
G=(1+2*10k/2k)=11
Resistors used for gain should be high precision Rs
Drawn from MMX Signal Conditioning Path
Instead of ISL70227, ISL28227 (the commercial equivalent) is used
Amplifier.SchDoc
Size
A4
Number
Revision
Date: 3.17.2023
Sheet of
File: D:_baffler-H_\\Amplifier.SchDoc
Drawn By: Luca Jungnickel


Number Connections:
SPI IN
SPI Out
SPI SCLK
SPI_CS
Total: 12
3.3V
5V
GND
CLK_IN
START
PWDN
RESET
DRDY
J1
Molex Picoblade 12pin Horizontal THT
CONNECTOR_5V_IN
CONNECTOR_SPI_CS
CONNECTOR_START
CONNECTOR_DRDY
CONNECTOR_SPI_DOUT
CONNECTOR_SPI_DIN
CONNECTOR_SPI_SCLK
CONNECTOR_RESET
CONNECTOR_3V3_IN
CONNECTOR_GND
MainSheet[3C]
MainSheet[4C]
MainSheet[3D]
Title
Connector.SchDoc
Size
A4
Number
Revision
Date: 3.17.2023
Sheet of
File: D:_Baffler-H_\\Connector.SchDoc
Drawn By: Luca Jungnickel


Buffering reference voltage
U2B
ISL28227
+5V
In_B-
In_B+
Out_B
+Λ/+Λ
-Λ/-Λ
Bias2
VREF_OUT
MainSheet[3A]
Buffering reference voltage
U2A
ISL28227
In_A-
In_A+
Out_A
Bias1
VREF_BIAS
C9
100nF
15V
2V5Bias
TP13
Bias_out
C8
100nF
15V
C7
10uF
15V
VREF=2.5V
U3
REF43
NC
V+
(TMP/CMP)
GND
VOUT
TRIM
Change to original MMX
(MMX: 1x 470nF Cap)
here: 10uF, 100nF, as in datasheet
5V needed (datasheet)
C6
100nF
15V
C5
10uF
15V
VREF_AVDD
VREF_GND
Testpoints are integrated on
the PCB with Vias:
Bias1
Bias2
2V5Bias as conventional
Testpoint
Drawn from MMX REF43
Title
Voltage Reference.SchDoc
Size
A4
Number
Revision
Date: 3.17.2023
Sheet of
File: D:_baffler-H__\Voltage Reference.SchDoc
Drawn By: Luca Jungnickel

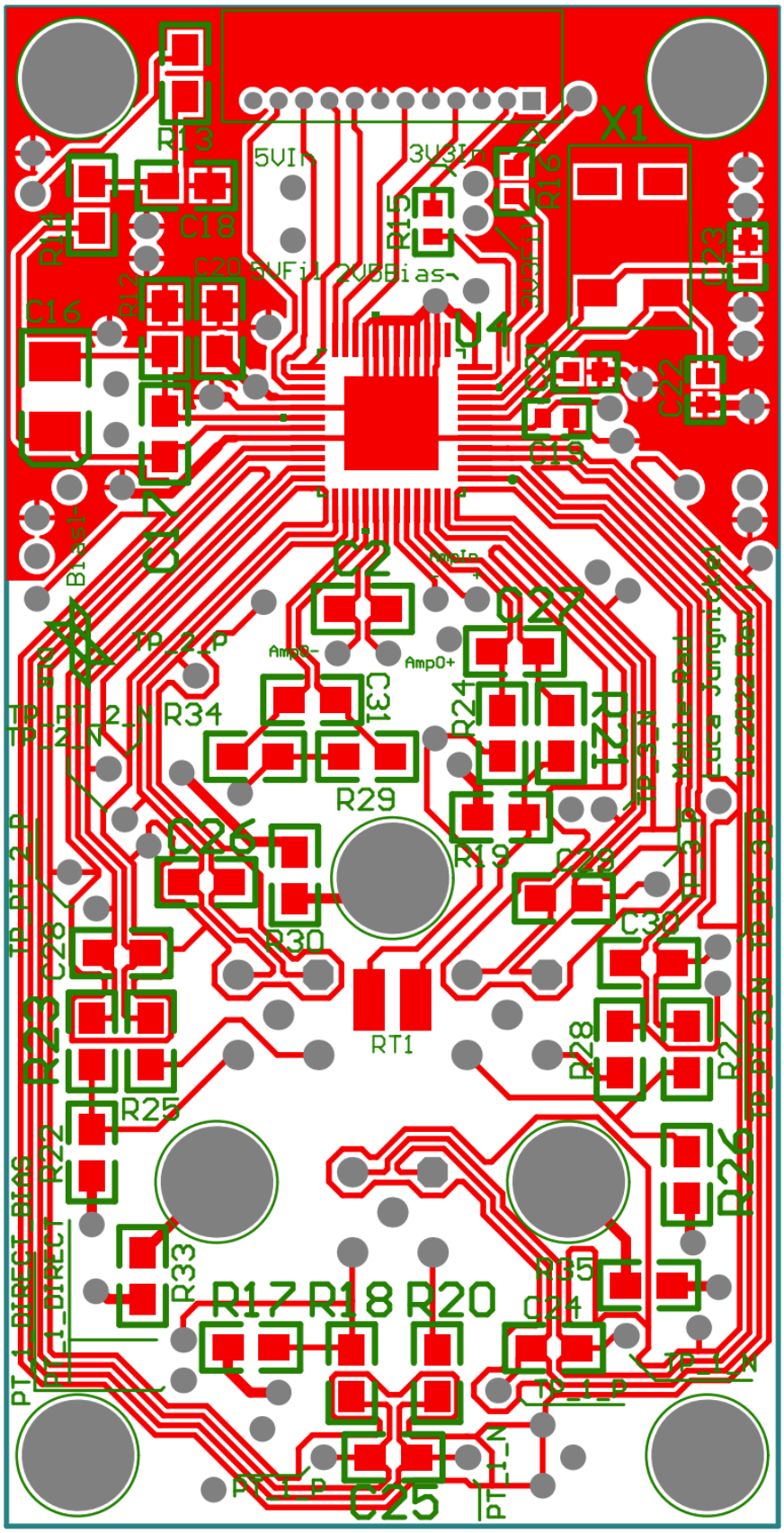

Testpoints are integrated on
the PCB with Vias:
5VIn (not filtered)
5VFil (Filtered 5V)
3V3In (not filtered)
3V3Fil (Filtered 3V3)
5V
POWER_AVDD_IN
MainSheet[4C]
Input
5V_In
L1
10uH
L2
10uH
#=5uH
L-Type LC Lowpass
f = 1 / (2*pi*(L*C)^-2) =
1 / (2*pi*(5u*5u)^-2) =
1 / (2*pi*5u) =
1 / (10u*pi) =
31.83 kHz
Use Tantalum
~=5uF
C11
10uF
15V
C12
10uF
15V
Snubber
R10
1R
C10
100nF
10V
POWER_AVDD_OUT
MainSheet[4B]
3V3
MainSheet[4C]
POWER_DVDD_IN
3V3_In
Input
L3
10uH
L4
10uH
Use Tantalum
C14
10uF
15V
C15
10uF
15V
Snubber
R11
1R
C13
100nF
10V
POWER_DVDD_OUT
MainSheet[4B]
POWER_GND
MainSheet[4C]
Drawn from MMX MiniRAD Power Filtering
Changes: Current Measurements were removed
Title
Power_Filter.SchDoc
Size
A4
Number
Revision
Date: 3.17.2023
Sheet of
File: D:_Daffler-H__\Power_Filter.SchDoc
Drawn By: Luca Jungnickel


Drawn from MMX with changes
Datasheet suggests: 10uF Tantalum + 100nF Cap
U4
ADS1258-EP
MUX
16:1
24 bit
ADC
FILTER
OSC
CTRL
SPI
GPIO
AVDD
AVSS
DVDD
DGND
VREFP
VREFN
AIN0
AIN1
AIN2
AIN3
AIN4
AIN5
AIN6
AIN7
AIN8
AIN9
AIN10
AIN11
AIN12
AIN13
AIN14
AIN15
AINCOM
ADCINP
ADCINN
MUXOUTP
MUXOUTN
CLKIO
CLKSEL
XTAL1
XTAL2
PLLCAP
GPIO0
GPIO1
GPIO2
GPIO3
GPIO4
GPIO5
GPIO6
GPIO7
!CS
!DRDY
SCLK
DIN
DOUT
START
!RESET
!PWDN
Heatsink
CLKENB=0 -> CLKIO internally grounded
CLKSEL=L for quartz at XTAL1/XTAL2
Crystal MC-206
place close
ADC_GND
+3V3
+5V
ADC_VREFN
Vrefn = GND
ADC_VREFP
Vrefp = 2.5V
VREF_N_FILTERED
VREF_P_FILTERED
ADC_IN_0
ADC_IN_1
ADC_IN_2
ADC_IN_3
ADC_IN_4
ADC_IN_5
ADC_IN_6
ADC_IN_7
ADC_IN_8
ADC_IN_9
ADC_IN_10
ADC_IN_11
ADC_IN_12
ADC_IN_13
ADC_IN_14
ADC_IN_15
ADC_AINCOM
ADC_ADCIN_N
ADC_ADCIN_P
ADC_MUXOUT_N
ADC_MUXOUT_P
ADC_SPI_CS
ADC_DRDY
ADC_SPI_SCLK
ADC_SPI_DIN
ADC_SPI_DOUT
ADC_START
ADC_RESET
CONN_SPI_CS
CONN_DRDY
CONN_SPI_SCLK
CONN_SPI_DIN
CONN_SPI_DOUT
CONN_START
RESET_R
51R
CONN_RESET
Title
ADC.SchDoc
Size
A4
Number
Revision
Date: 3.17.2023
File: D:_bafiler-H_\\ADC.SchDoc
Sheet of
Drawn By: Luca Jungnickel

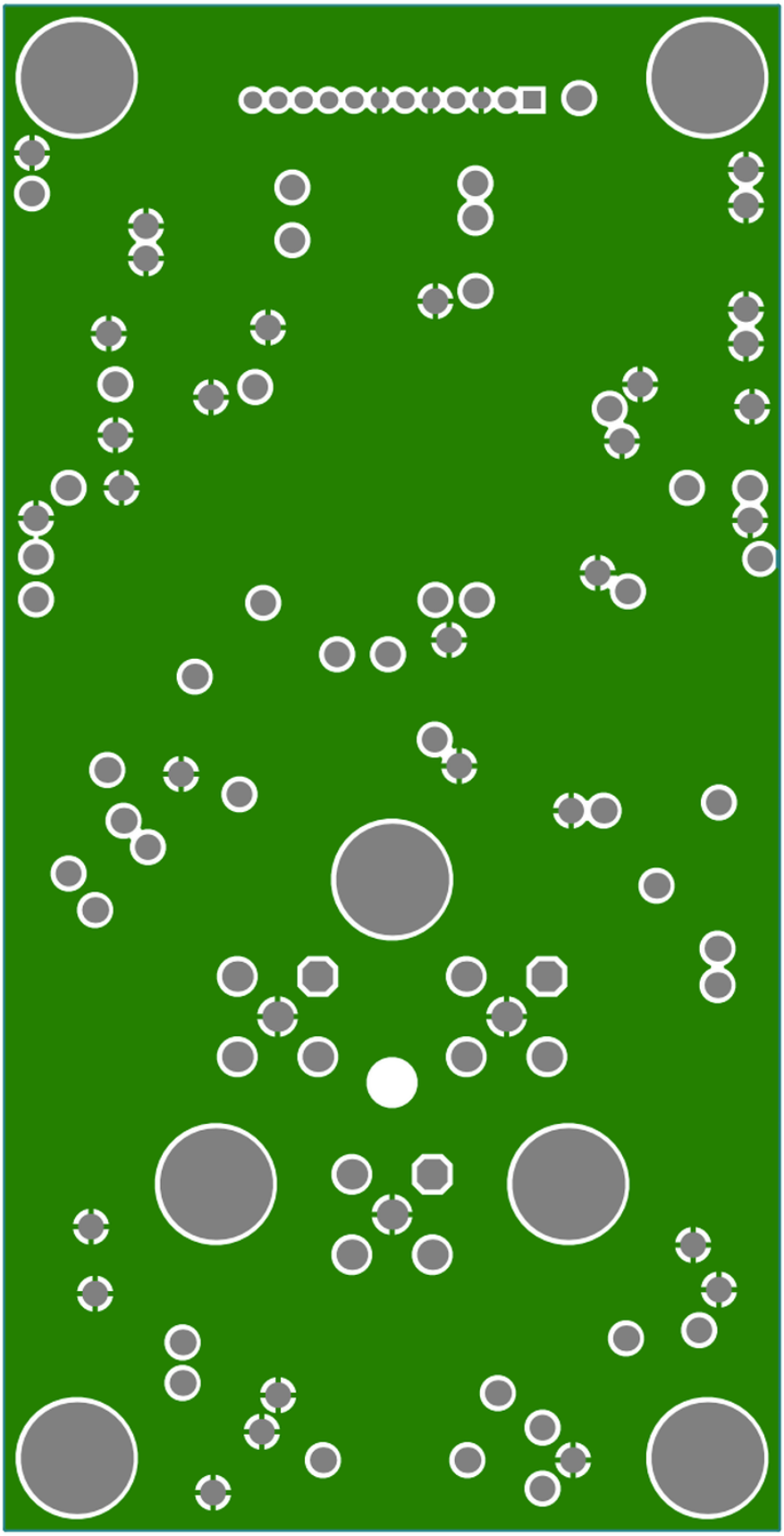

PT100
directly drawn from HP3 Rad
Power Pins
Offset Measuring
directly drawn from MMX miniRad
InSight Thermopile Sensors
directly drawn from HP3 Rad
Sensor 1 has additional testpoints next at the PT100, for more testing possibilities
Never place both resistors at the same time
Signal Harness for Sensors
Grounding is optional
Grounding of Aluminium Body
BIAS=2V5
DVDD=3V3
Sensor_And_Offset_Measuring_v2.SchDoc
Luca Jungnickel

Comment	Description	Designator	Footprint	LibRef	Quantity
470pF	Capacitor	C1, C4	0805-C	Cap	2
10nF	Capacitor	C2, C18	0805-C	Cap	2
100nF	Capacitor	C3, C6, C8, C9, C10, C13, C17, C20	0805-C	Cap	8
10uF	Polarized Capacitor (Surface Mount), Tantalum	C5, C7, C11, C12, C14, C15	C	Cap+	6
10uF	Polarized Capacitor (Surface Mount), Tantalum	C16	B	Cap+	1
100nF	Capacitor	C19	0603	Cap	1
22nF	Capacitor	C21	0603	Cap	1
4.7pF	Capacitor	C22, C23	0603	Cap	2
470nF	Capacitor	C24, C25, C26, C27, C28, C29, C30, C31	0805-C	Cap	8
Molex Picoblade 12pin Horizontal THT	Molex Part Number: 530481210	J1	CON_530481210_MOL	Molex Picoblade 12pin Horizontal THT	1
10uH	Inductor, SMD	L1, L2, L3, L4	INDC3216-1206	Inductor	4
0R	Resistor	R1, R2, R3, R4, R30, R31, R32, R33, R35	0805-C	Resistor SMD	9
20R	Resistor	R5, R9, R12, R13, R14	0805-C	Resistor SMD	5
10kPR	Resistor	R6, R8	0805-C	Resistor SMD	2
2kPR	Resistor	R7	0805-C	Resistor SMD	1
1R	Resistor	R10, R11	0805-C	Resistor SMD	2
51R	Resistor	R15	0603	Resistor SMD	1
0R	Resistor	R16	0603	Resistor SMD	1
47kPR	Resistor	R17, R19, R22, R26	0805-C	Resistor SMD	4
1kR	Resistor	R18, R20, R21, R23, R24, R25, R27, R28	0805-C	Resistor SMD	8
10kP	Resistor	R29, R34	0805-C	Resistor SMD	2
PT100		RT1	PT100	PT100 - 2 Wire	1
THERMOPILE		TH1, TH2, TH3	Insight Sensor	Insight Sensor	3
ISL28227		U1, U2	SO-8	ISL28227	2
REF43	REF43	U3	SO-8	REF43	1
ADS1258-EP	ADS1258-EP	U4	PHP48_4P6X4P6	ADS1258	1
Crystal MC-306		X1	MC-306 Crystal	Crystal MC-306	1

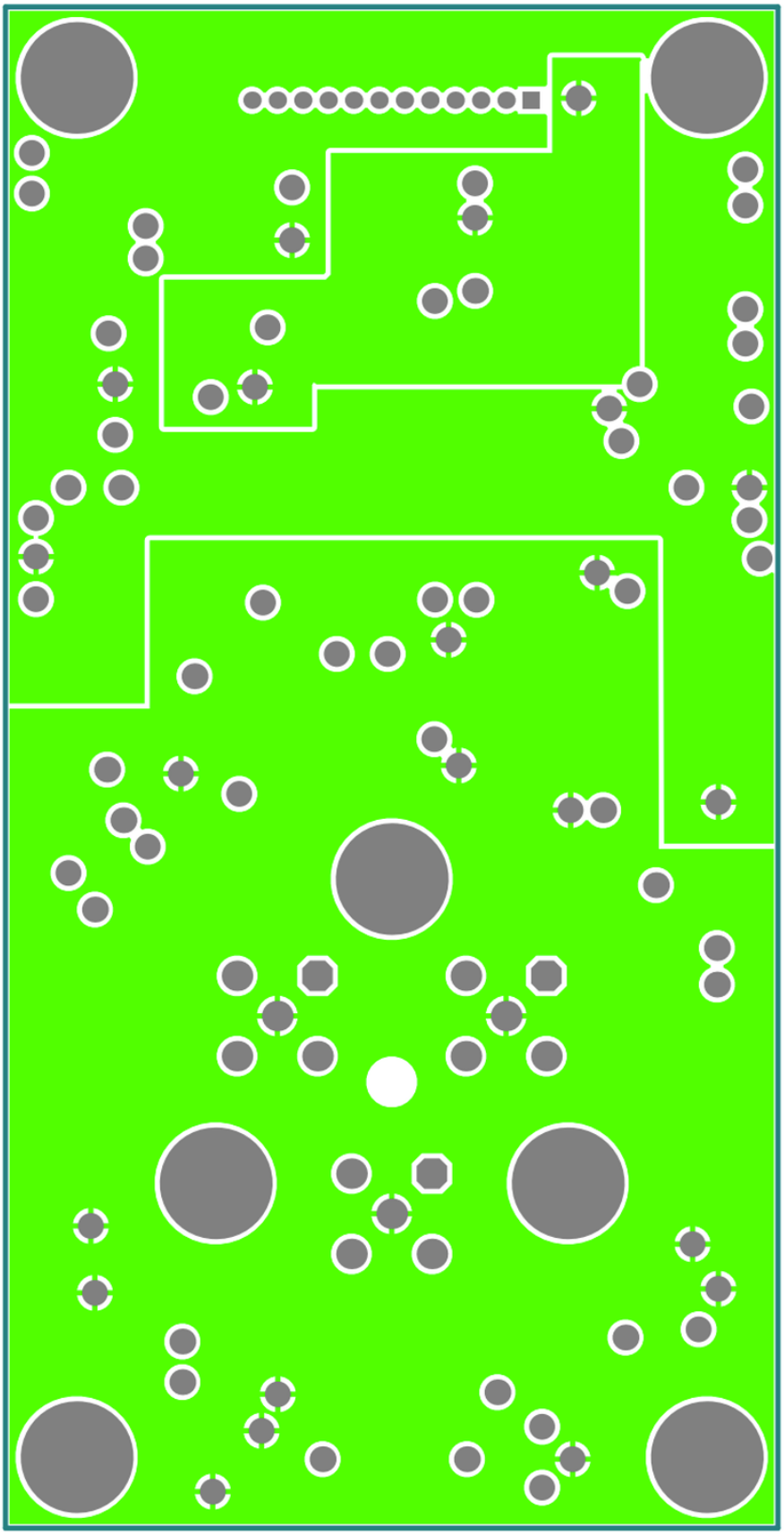
A.3.2. L1 - Top Layer



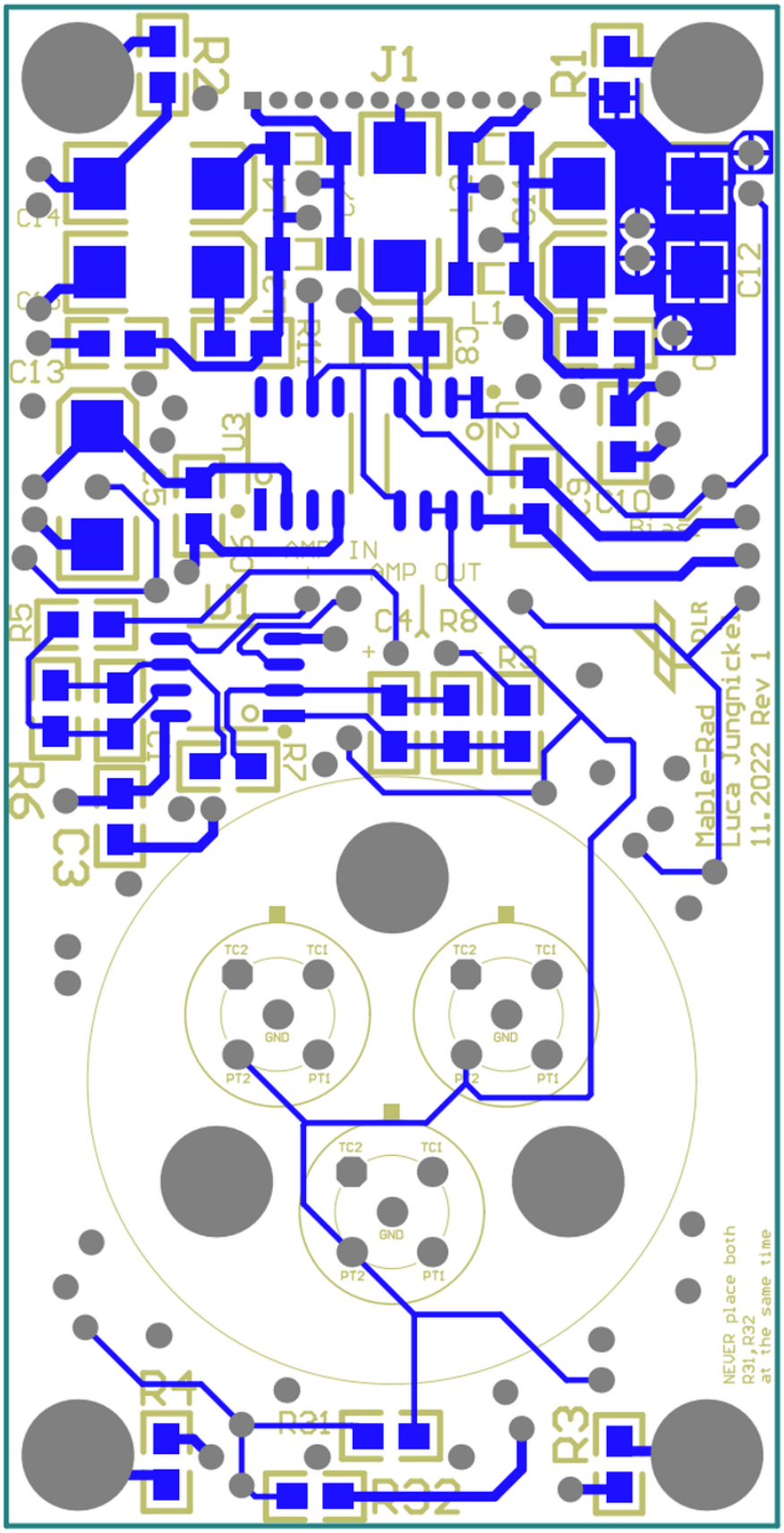
A.3.3. L2 - Ground Layer



A.3.4. L3 - Ground and Power Layer

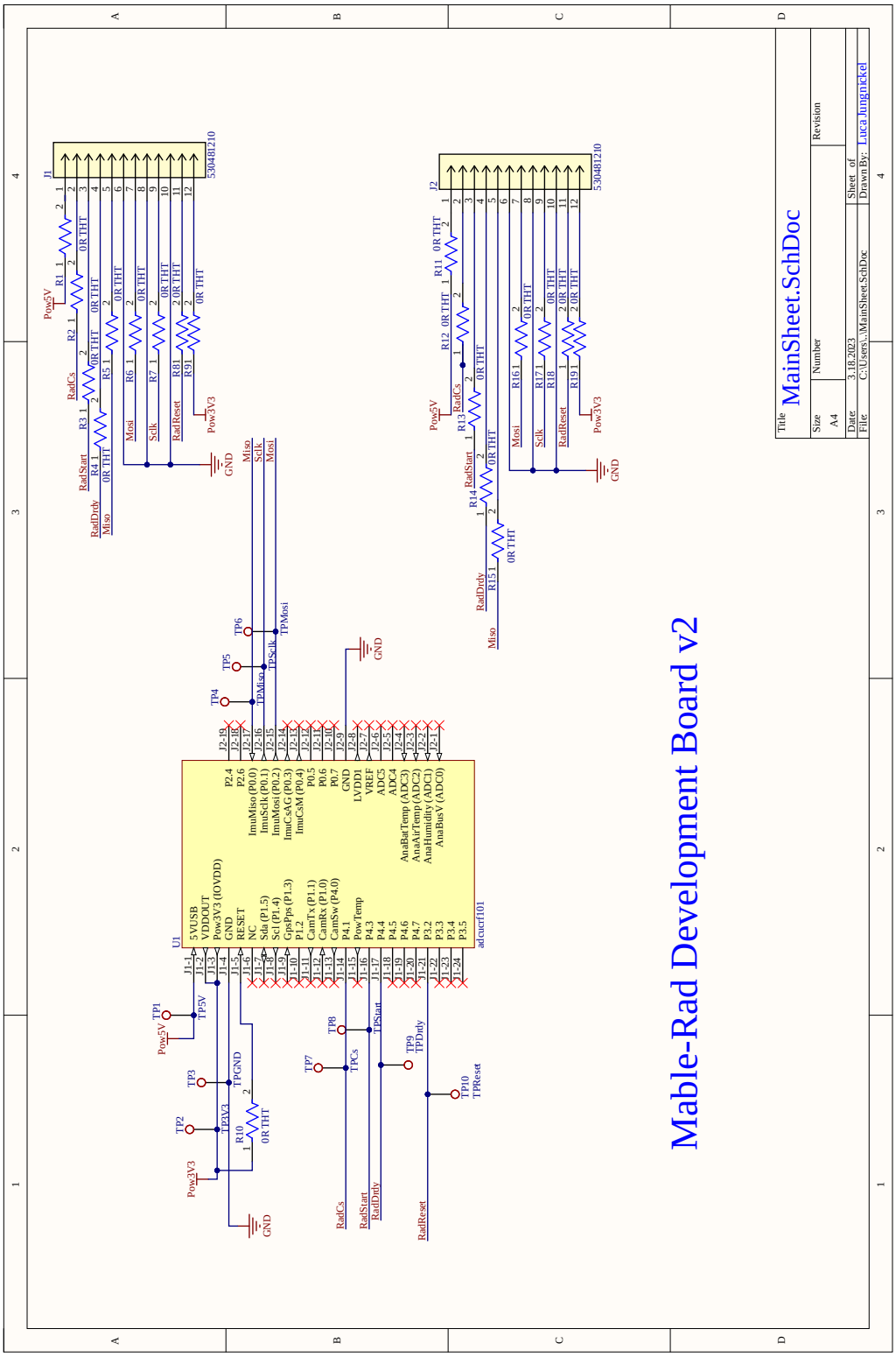


A.3.5. L4 - Bottom Layer

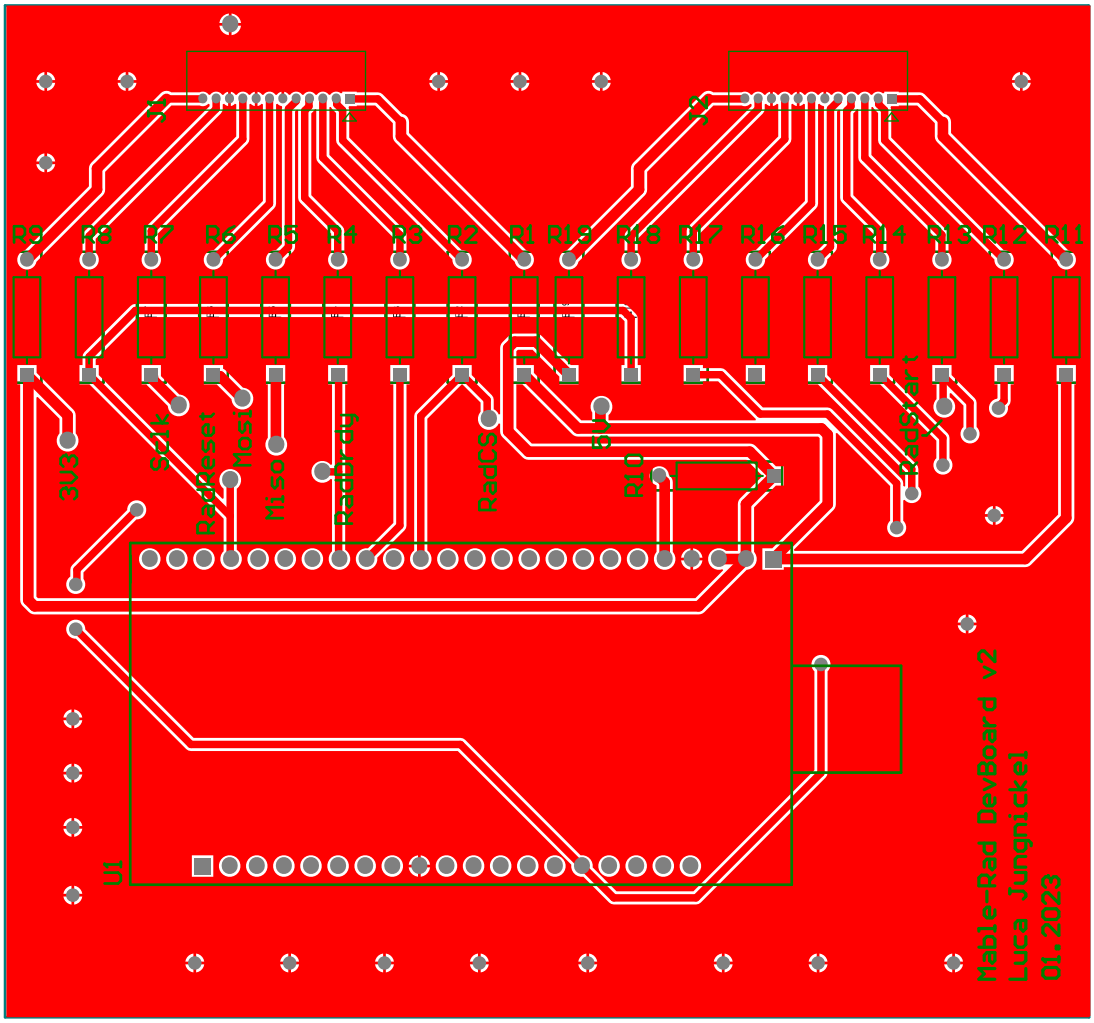


A.4. MableRad Development Board - Schematics and PCB Design

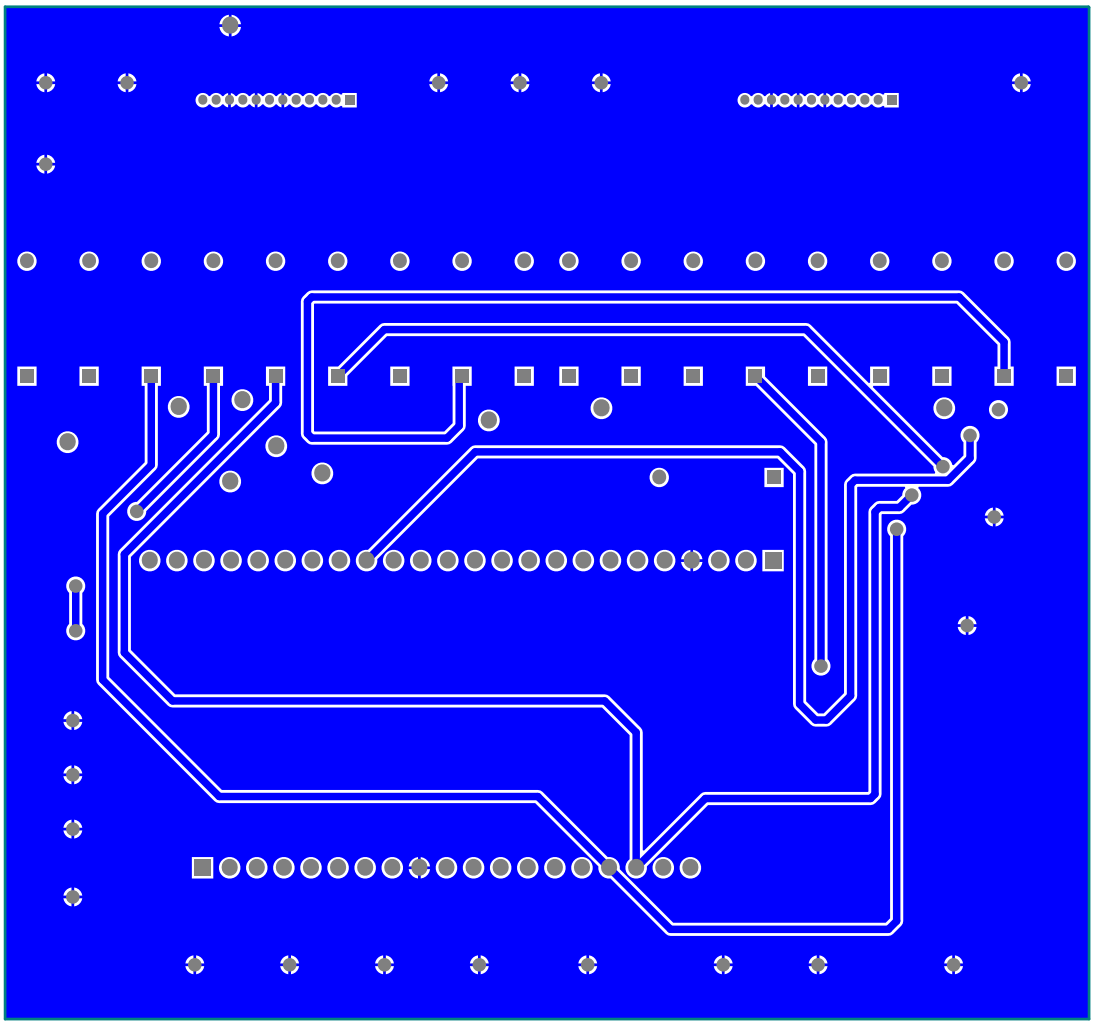
A.4.1. Schematics



A.4.2. L1 - Top Layer



A.4.3. L2 - Bottom Layer



A.5. Firmware Radiometer

A.5.1. Ads1258.h

```
1   /*
2    * ads1258.h
3    *
4    *  Created on: 02.02.2023
5    *      Author: Luca Jungnickel
6    *      DLR-PF
7    *
8    * Driver for the ADS1258. Core functionality of the ADS1258 is combined code from
9    * the driver from TI and own code which extends the sample driver.
10   * The original driver from TI can be found on
11   * https://www.ti.com/lit/zip/sbac222 (as of 02.02.2023)
12   *
13   *
14   * ADS1258 and ADS1258-ep are using the same drivers.
15   */
16
17   /* --COPYRIGHT--,BSD
18   * Copyright (c) 2018, Texas Instruments Incorporated
19   * All rights reserved.
20   *
21   * Redistribution and use in source and binary forms, with or without
22   * modification, are permitted provided that the following conditions
23   * are met:
24   *
25   * *  Redistributions of source code must retain the above copyright
26   *    notice, this list of conditions and the following disclaimer.
27   *
28   * *  Redistributions in binary form must reproduce the above copyright
29   *    notice, this list of conditions and the following disclaimer in the
30   *    documentation and/or other materials provided with the distribution.
31   *
32   * *  Neither the name of Texas Instruments Incorporated nor the names of
33   *    its contributors may be used to endorse or promote products derived
34   *    from this software without specific prior written permission.
35   *
36   * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
37   * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
38   * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
39   * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
40   * CONTRIBUTORS BE LIABLE FOR ANY direct, INDIRECT, INCIDENTAL, SPECIAL,
41   * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
42   * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
43   * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
44   * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
45   * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
46   * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
47   * --/COPYRIGHT--*/
48
49  #ifndef SRC_SENSORS_ADS1258_H_
50  #define SRC_SENSORS_ADS1258_H_
51
52  #include <hal/aducrf101/spi/spi.h>
53
54  namespace sensors
55  {
56  class Ads1258
57  {
58      public:
59          Ads1258(hal::adu::Spi& spiIn,
60          ADI_GPIO_TypeDef* drdyPortIn, const int drdyPinIn,
61          ADI_GPIO_TypeDef* csPortIn, const int csPinIn,
62          ADI_GPIO_TypeDef* startPortIn, const int startPinIn,
63          ADI_GPIO_TypeDef* resetPortIn, const int resetPinIn);
64
65
66
67      /**
68       *  Initialises the ADS1258.
69       *  Resets the chip, writes the given configuration and reads it's back for
         verification.
70       *  If all registers are correctly set, true will be returned.
71       *  If this function returns false, the function should be called again.
72       *
```

```
 * @param regConfig[]
 *  Array of configuration of the registers. Has to be the size of (numRegisters - 1).
 *  The order of configuration is: (corresponds to the address of the registers, starting from 0x0)
 *  regAddrConfig0
 *  regAddrConfig1
 *  regAddrMuxsch
 *  regAddrMuxdif
 *  regAddrMuxsg0
 *  regAddrMuxsg1
 *  regAddrSysred
 *  regAddrGpioc
 *  regAddrGpiod
 *
 * @param samplesDiscard
 *  Number of samples which will be discarded after a channel switch was performed.
 *  @see setChannel()
 *
 * @return true if successful,
 *         false if an error occured
 */
bool init(uint8_t regConfig[], const int samplesDiscard);


/**
 *
 * @brief Fetches one sample from the ADC and writes it to the given pointer.
 *
 * @param data pointer which will store the result
 *
 * @return true if the sampling was successful,
 *         false if an error occured
*
*/
bool getSample(int32_t& data);


/**
 * @brief Fetches the given amount of samples from the ADC, calculates the average (moving-average),
 * and writes the result to the given pointer.
 *
 * @param data pointer for the result
 * @param numSamples number of samples which will be performed
 *
 * @return true if the sampling was successful,
 *         false if an error occured
 */
bool getDataAndOversample(double& data, uint32_t numSamples);




/**
 * @brief   Sets the channel combination for measuring in differential mode.
 * The writing to the ADC1258 will not be read-back, it will be assumed that it worked and the ADC
 * was correctly initialised before.
 *
 * After a successful configuration, the amount of mNumSamplesDiscard of samples will be taken and discarded.
 * Therefore, the radiometer has to be active for reading data (rad.startConversions()).
 * @see mNumSamplesDiscard
 *
 * @param   channelCombination i.e. (MUXSCH_AINP_AIN7 | MUXSCH_AINN_AIN6) or CHANNELS_THERMOPILE_1_PT
 * @see     MUXSCH_AINP_... / MUXSCH_AINN_... / CHANNELS_...
 *
 * Call rad.init() before.
 * Call rad.startConversions() before.
```

```
138         *
139         * @return true if successful,
140         *         false if an error occured
141         */
142        bool setChannel(uint8_t channelCombination);
143
144
145
146        /**
147         * @brief   Activate conversions of the Radiometer.
148         *
149         * This will activate the ADS1258-ep with setting the START pin to HIGH.
150         * This function has to be called before any data is read from the Radiometer.
151         */
152        void startConversions(void);
153
154
155
156        /**
157         * @brief   Deactivate conversions of the Radiometer.
158         *
159         * This will deactivate the ADS1258-ep with setting the START pin to LOW.
160         * Following data reads will then result 0x0, until the conversions are started
            again.
161         */
162        void stopConversions(void);
163
164
165
166        /**
167         * @brief Getter function to access the set registers of the ADC without reading
            directly from it.
168         *
169         * A local copy of the currently set registers will be accessed for that.
170         *
171         * @param address register address, i.e. REG_ADDR_CONFIG0
172         * @return value of the register
173         */
174        uint8_t getRegisterValue(uint8_t address);
175
176
177
178        //--------------Inline Getters--------------
179        inline ADI_GPIO_TypeDef* getCsPort(void)
180        {
181            return mCsPort;
182        }
183        inline ADI_GPIO_TypeDef* getDrdyPort(void)
184        {
185            return mDrdyPort;
186        }
187        inline ADI_GPIO_TypeDef* getStartPort(void)
188        {
189            return mStartPort;
190        }
191        inline ADI_GPIO_TypeDef* getResetPort(void)
192        {
193            return mResetPort;
194        }
195        inline int getCsPin(void)
196        {
197            return mCsPin;
198        }
199        inline int getDrdyPin(void)
200        {
201            return mDrdyPin;
202        }
203        inline int getStartPin(void)
204        {
205            return mStartPin;
206        }
207        inline int getResetPin(void)
208        {
```

```
209         return mResetPin;
210     }
211     inline int getNumSamplesDiscard(void)
212     {
213         return mNumSamplesDiscard;
214     }
215
216
217
218     /* Command byte definition
219     *
        ---------------------------------------------------------------------------------
220     * |  Bit 7  |  Bit 6  |  Bit 5  |  Bit 4  |  Bit 3  |  Bit 2  |  Bit 1  |  Bit 0
        |
221     *
        ---------------------------------------------------------------------------------
222     * |            C[2:0]           |   MUL   |                A[3:0]
        |
223     *
        ---------------------------------------------------------------------------------
224     */
225
226     /* SPI Commands */
227     static const uint8_t opcodeReadDirect                   = 0x00;
228     static const uint8_t opcodeReadCommand                  = 0x30;
229     static const uint8_t opcodeRreg                         = 0x40;
230     static const uint8_t opcodeWreg                         = 0x60;
231     static const uint8_t opcodePulseConvert                 = 0x80;
232     static const uint8_t opcodeReset                        = 0xC0;
233
234     /* Commands byte masks */
235     static const uint8_t opcodeCMask                        = 0xE0;
236     static const uint8_t opcodeMulMask                      = 0x10;
237     static const uint8_t opcodeAMask                        = 0x0F;
238
239     /* Read mode enum */
240     typedef enum { direct, command } ReadMode;
241
242
243
244     //*****************************************************************************
245     //
246     // Status byte formatting
247     //
248     //*****************************************************************************
249
250     /* STATUS byte definition
251     *
        ---------------------------------------------------------------------------------
252     * |  Bit 7  |  Bit 6  |  Bit 5  |  Bit 4  |  Bit 3  |  Bit 2  |  Bit 1  |  Bit 0
        |
253     *
        ---------------------------------------------------------------------------------
254     * |   NEW   |   OVF   |  SUPPLY |                   CHID[4:0]
        |
255     *
        ---------------------------------------------------------------------------------
256     */
257
258     /* STATUS byte field masks */
259     static const uint8_t statusNewMask                      = 0x80;  /* Indicates new
        data */
260     static const uint8_t statusOvfMask                      = 0x40;  /* Indicates
        differential over-range condition */
261     static const uint8_t statusSupplyMask                   = 0x20;  /* Indicates low
        analog power-supply condition */
262     static const uint8_t statusChidMask                     = 0x1F;  /* Channel ID
        bits */
263
264     /* CHID field values */
265     static const uint8_t statusChidDiff0                    = 0x00;
266     static const uint8_t statusChidDiff1                    = 0x01;
267     static const uint8_t statusChidDiff2                    = 0x02;
```

```
268     static const uint8_t statusChidDiff3                    = 0x03;
269     static const uint8_t statusChidDiff4                    = 0x04;
270     static const uint8_t statusChidDiff5                    = 0x05;
271     static const uint8_t statusChidDiff6                    = 0x06;
272     static const uint8_t statusChidDiff7                    = 0x07;
273     static const uint8_t statusChidAin0                     = 0x08;
274     static const uint8_t statusChidAin1                     = 0x09;
275     static const uint8_t statusChidAin2                     = 0x0A;
276     static const uint8_t statusChidAin3                     = 0x0B;
277     static const uint8_t statusChidAin4                     = 0x0C;
278     static const uint8_t statusChidAin5                     = 0x0D;
279     static const uint8_t statusChidAin6                     = 0x0E;
280     static const uint8_t statusChidAin7                     = 0x0F;
281     static const uint8_t statusChidAin8                     = 0x10;
282     static const uint8_t statusChidAin9                     = 0x11;
283     static const uint8_t statusChidAin10                    = 0x12;
284     static const uint8_t statusChidAin11                    = 0x13;
285     static const uint8_t statusChidAin12                    = 0x14;
286     static const uint8_t statusChidAin13                    = 0x15;
287     static const uint8_t statusChidAin14                    = 0x16;
288     static const uint8_t statusChidAin15                    = 0x17;
289     static const uint8_t statusChidOffset                   = 0x18;
290     static const uint8_t statusChidVcc                      = 0x1A;
291     static const uint8_t statusChidTemp                     = 0x1B;
292     static const uint8_t statusChidGain                     = 0x1C;
293     static const uint8_t statusChidRef                      = 0x1D;
294     static const uint8_t statusChidFixedchmode              = 0x1F; /* ID for
        fixed-channel mode */
295
296
297     //*****************************************************************************
298     //
299     // Register definitions
300     //
301     //*****************************************************************************
302
303
304     /* Register 0x00 (CONFIG0) definition
305     *
        ---------------------------------------------------------------------------------
306     * |  Bit 7  |  Bit 6  |  Bit 5  |  Bit 4  |  Bit 3  |  Bit 2  |  Bit 1  |  Bit 0
        |
307     *
        ---------------------------------------------------------------------------------
308     * |    0    |  SPIRST |  MUXMOD |  BYPAS  |  CLKENB |   CHOP  |   STAT  |    0
        |
309     *
        ---------------------------------------------------------------------------------
310     */
311
312     /** CONFIG0 register address */
313     static const uint8_t regAddrConfig0                     = 0x00;
314     /** CONFIG0 default (reset) value */
315     static const uint8_t config0Default                     = 0x0A;
316     /* CONFIG0 register field masks */
317     static const uint8_t config0SpirstMask                  = 0x40;
318     static const uint8_t config0MuxmodMask                  = 0x20;
319     static const uint8_t config0BypasMask                   = 0x10;
320     static const uint8_t config0ClkenbMask                  = 0x08;
321     static const uint8_t config0ChopMask                    = 0x04;
322     static const uint8_t config0StatMask                    = 0x02;
323
324
325     /* Register 0x01 (CONFIG1) definition
326     *
        ---------------------------------------------------------------------------------
327     * |  Bit 7  |  Bit 6  |  Bit 5  |  Bit 4  |  Bit 3  |  Bit 2  |  Bit 1  |  Bit 0
        |
328     *
        ---------------------------------------------------------------------------------
329     * |  IDLMOD |          DLY[2:0]          |     SCBCS[1:0]    |     DRATE[0:1]
        |
330     *
```

```
        ------------------------------------------------------------------------------
331     */
332
333     /** CONFIG1 register address */
334     static const uint8_t regAddrConfig1                     = 0x01;
335     /** CONFIG1 default (reset) value */
336     static const uint8_t config1Default                     = 0x83;
337     /* CONFIG1 register field masks */
338     static const uint8_t config1IdlmodMask                  = 0x80;
339     static const uint8_t config1DlyMask                     = 0x70;
340     static const uint8_t config1ScbcsMask                   = 0x0C;
341     static const uint8_t config1DrateMask                   = 0x03;
342
343     /* DLY field values */
344     static const uint8_t config1Dly0us                      = 0x00;
345     static const uint8_t config1Dly8us                      = 0x10;
346     static const uint8_t config1Dly16us                     = 0x20;
347     static const uint8_t config1Dly32us                     = 0x30;
348     static const uint8_t config1Dly64us                     = 0x40;
349     static const uint8_t config1Dly128us                    = 0x50;
350     static const uint8_t config1Dly256us                    = 0x60;
351     static const uint8_t config1Dly384us                    = 0x70;
352
353     /* SCBCS field values */
354     static const uint8_t config1ScbcsOff                    = 0x00;
355     static const uint8_t config1Scbcs15ua                   = 0x40;
356     static const uint8_t config1Scbcs24ua                   = 0xC0;
357
358     /* DRATE field values (fixed-channel DRs shown) */
359     static const uint8_t config1Drate1953sps                = 0x00;
360     static const uint8_t config1Drate7813sps                = 0x01;
361     static const uint8_t config1Drate31250sps               = 0x02;
362     static const uint8_t config1Drate125000sps              = 0x03;
363
364
365     /* Register 0x02 (MUXSCH) definition
366     *
        ------------------------------------------------------------------------------
367     * |  Bit 7  |  Bit 6  |  Bit 5  |  Bit 4  |  Bit 3  |  Bit 2  |  Bit 1  |  Bit 0
        |
368     *
        ------------------------------------------------------------------------------
369     * |               AINP[3:0]               |               AINN[3:0]
        |
370     *
        ------------------------------------------------------------------------------
371     */
372
373     /** MUXSCH register address */
374     static const uint8_t regAddrMuxsch                      = 0x02;
375
376     /** MUXSCH default (reset; value */
377     static const uint8_t muxschDefault                      = 0x00;
378
379     /* MUXSCH register field masks */
380     static const uint8_t muxschAinpMask                     = 0xF0;
381     static const uint8_t muxschAinnMask                     = 0x0F;
382
383     /* AINP field values */
384     static const uint8_t muxschAinpAin0                     = 0x00;
385     static const uint8_t muxschAinpAin1                     = 0x10;
386     static const uint8_t muxschAinpAin2                     = 0x20;
387     static const uint8_t muxschAinpAin3                     = 0x30;
388     static const uint8_t muxschAinpAin4                     = 0x40;
389     static const uint8_t muxschAinpAin5                     = 0x50;
390     static const uint8_t muxschAinpAin6                     = 0x60;
391     static const uint8_t muxschAinpAin7                     = 0x70;
392     static const uint8_t muxschAinpAin8                     = 0x80;
393     static const uint8_t muxschAinpAin9                     = 0x90;
394     static const uint8_t muxschAinpAin10                    = 0xA0;
395     static const uint8_t muxschAinpAin11                    = 0xB0;
396     static const uint8_t muxschAinpAin12                    = 0xC0;
397     static const uint8_t muxschAinpAin13                    = 0xD0;
```

```
398     static const uint8_t muxschAinpAin14                    = 0xE0;
399     static const uint8_t muxschAinpAin15                    = 0xF0;
400
401     /* AINN field values */
402     static const uint8_t muxschAinnAin0                     = 0x00;
403     static const uint8_t muxschAinnAin1                     = 0x01;
404     static const uint8_t muxschAinnAin2                     = 0x02;
405     static const uint8_t muxschAinnAin3                     = 0x03;
406     static const uint8_t muxschAinnAin4                     = 0x04;
407     static const uint8_t muxschAinnAin5                     = 0x05;
408     static const uint8_t muxschAinnAin6                     = 0x06;
409     static const uint8_t muxschAinnAin7                     = 0x07;
410     static const uint8_t muxschAinnAin8                     = 0x08;
411     static const uint8_t muxschAinnAin9                     = 0x09;
412     static const uint8_t muxschAinnAin10                    = 0x0A;
413     static const uint8_t muxschAinnAin11                    = 0x0B;
414     static const uint8_t muxschAinnAin12                    = 0x0C;
415     static const uint8_t muxschAinnAin13                    = 0x0D;
416     static const uint8_t muxschAinnAin14                    = 0x0E;
417     static const uint8_t muxschAinnAin15                    = 0x0F;
418
419     /* Register 0x03 (MUXDIF) definition
420     *
        ---------------------------------------------------------------------------------
421     * |  Bit 7  |  Bit 6  |  Bit 5  |  Bit 4  |  Bit 3  |  Bit 2  |  Bit 1  |  Bit 0
        |
422     *
        ---------------------------------------------------------------------------------
423     * |  DIFF7  |  DIFF6  |  DIFF5  |  DIFF4  |  DIFF3  |  DIFF2  |  DIFF1  |  DIFF0
        |
424     *
        ---------------------------------------------------------------------------------
425     */
426
427     /** MUXDIF register address */
428     static const uint8_t regAddrMuxdif                      = 0x03;
429
430     /** MUXDIF default (reset; value */
431     static const uint8_t muxdifDefault                      = 0x00;
432
433     /* MUXDIF register field masks */
434     static const uint8_t muxdifDiff7Enable                  = 0x80;
435     static const uint8_t muxdifDiff6Enable                  = 0x40;
436     static const uint8_t muxdifDiff5Enable                  = 0x20;
437     static const uint8_t muxdifDiff4Enable                  = 0x10;
438     static const uint8_t muxdifDiff3Enable                  = 0x08;
439     static const uint8_t muxdifDiff2Enable                  = 0x04;
440     static const uint8_t muxdifDiff1Enable                  = 0x02;
441     static const uint8_t muxdifDiff0Enable                  = 0x01;
442
443
444
445     /* Register 0x04 (MUXSG0) definition
446     *
        ---------------------------------------------------------------------------------
447     * |  Bit 7  |  Bit 6  |  Bit 5  |  Bit 4  |  Bit 3  |  Bit 2  |  Bit 1  |  Bit 0
        |
448     *
        ---------------------------------------------------------------------------------
449     * |   AIN7  |   AIN6  |   AIN5  |   AIN4  |   AIN3  |   AIN2  |   AIN1  |   AIN0
        |
450     *
        ---------------------------------------------------------------------------------
451     */
452
453     /** MUXSG0 register address */
454     static const uint8_t regAddrMuxsg0                      = 0x04;
455
456     /** MUXSG0 default (reset; value */
457     static const uint8_t muxsg0Default                      = 0xFF;
458
459     /* MUXSG0 register field masks */
460     static const uint8_t muxsg0Ain7Enable                   = 0x80;
```

```
461     static const uint8_t muxsg0Ain6Enable                   = 0x40;
462     static const uint8_t muxsg0Ain5Enable                   = 0x20;
463     static const uint8_t muxsg0Ain4Enable                   = 0x10;
464     static const uint8_t muxsg0Ain3Enable                   = 0x08;
465     static const uint8_t muxsg0Ain2Enable                   = 0x04;
466     static const uint8_t muxsg0Ain1Enable                   = 0x02;
467     static const uint8_t muxsg0Ain0Enable                   = 0x01;
468
469
470
471     /* Register 0x05 (MUXSG1) definition
472     *
        ---------------------------------------------------------------------------------
473     * |  Bit 7  |  Bit 6  |  Bit 5  |  Bit 4  |  Bit 3  |  Bit 2  |  Bit 1  |  Bit 0
        |
474     *
        ---------------------------------------------------------------------------------
475     * |  AIN15  |  AIN14  |  AIN13  |  AIN12  |  AIN11  |  AIN10  |   AIN9  |   AIN8
        |
476     *
        ---------------------------------------------------------------------------------
477     */
478
479     /** MUXSG1 register address */
480     static const uint8_t regAddrMuxsg1                      = 0x05;
481
482     /** MUXSG1 default (reset; value */
483     static const uint8_t muxsg1Default                      = 0xFF;
484
485     /* MUXSG1 register field masks */
486     static const uint8_t muxsg1Ain15Enable                  = 0x80;
487     static const uint8_t muxsg1Ain14Enable                  = 0x40;
488     static const uint8_t muxsg1Ain13Enable                  = 0x20;
489     static const uint8_t muxsg1Ain12Enable                  = 0x10;
490     static const uint8_t muxsg1Ain11Enable                  = 0x08;
491     static const uint8_t muxsg1Ain10Enable                  = 0x04;
492     static const uint8_t muxsg1Ain9Enable                   = 0x02;
493     static const uint8_t muxsg1Ain8Enable                   = 0x01;
494
495
496
497     /* Register 0x06 (SYSRED) definition
498     *
        ---------------------------------------------------------------------------------
499     * |  Bit 7  |  Bit 6  |  Bit 5  |  Bit 4  |  Bit 3  |  Bit 2  |  Bit 1  |  Bit 0
        |
500     *
        ---------------------------------------------------------------------------------
501     * |    0    |    0    |   REF   |   GAIN  |   TEMP  |   VCC   |     0   |  OFFSET
        |
502     *
        ---------------------------------------------------------------------------------
503     */
504
505     /** SYSRED register address */
506     static const uint8_t regAddrSysred                      = 0x06;
507
508     /** SYSRED default (reset; value */
509     static const uint8_t sysredDefault                      = 0x00;
510
511     /* SYSRED register field masks */
512     static const uint8_t sysredRefEnable                    = 0x20;
513     static const uint8_t sysredGainEnable                   = 0x10;
514     static const uint8_t sysredTempEnable                   = 0x08;
515     static const uint8_t sysredVccEnable                    = 0x04;
516     static const uint8_t sysredOffsetEnable                 = 0x01;
517
518
519
520     /* Register 0x07 (GPIOC) definition
521     *
        ---------------------------------------------------------------------------------
522     * |  Bit 7  |  Bit 6  |  Bit 5  |  Bit 4  |  Bit 3  |  Bit 2  |  Bit 1  |  Bit 0
```

```
|
523     *
        ---------------------------------------------------------------------------
524     * |                                   CIO[7:0]
        |
525     *
        ---------------------------------------------------------------------------
526     */
527
528     /** GPIOC register address */
529     static const uint8_t regAddrGpioc                       = 0x07;
530
531     /** GPIOC default (reset; value */
532     static const uint8_t gpiocDefault                       = 0xFF;
533
534     /* GPIOC register field masks */
535     static const uint8_t gpiocGpio7Input                    = 0x80;
536     static const uint8_t gpiocGpio6Input                    = 0x40;
537     static const uint8_t gpiocGpio5Input                    = 0x20;
538     static const uint8_t gpiocGpio4Input                    = 0x10;
539     static const uint8_t gpiocGpio3Input                    = 0x08;
540     static const uint8_t gpiocGpio2Input                    = 0x04;
541     static const uint8_t gpiocGpio1Input                    = 0x02;
542     static const uint8_t gpiocGpio0Input                    = 0x01;
543
544
545
546     /* Register 0x08 (GPIOD) definition
547     *
        ---------------------------------------------------------------------------
548     * |  Bit 7  |  Bit 6  |  Bit 5  |  Bit 4  |  Bit 3  |  Bit 2  |  Bit 1  |  Bit 0
        |
549     *
        ---------------------------------------------------------------------------
550     * |                                   DIO[7:0]
        |
551     *
        ---------------------------------------------------------------------------
552     */
553
554     /** GPIOD register address */
555     static const uint8_t regAddrGpiod                       = 0x08;
556
557     /** GPIOD default (reset; value */
558     static const uint8_t gpiodDefault                       = 0x00;
559
560     /* GPIOD register field masks */
561     static const uint8_t gpiodGpio7High                     = 0x80;
562     static const uint8_t gpiodGpio6High                     = 0x40;
563     static const uint8_t gpiodGpio5High                     = 0x20;
564     static const uint8_t gpiodGpio4High                     = 0x10;
565     static const uint8_t gpiodGpio3High                     = 0x08;
566     static const uint8_t gpiodGpio2High                     = 0x04;
567     static const uint8_t gpiodGpio1High                     = 0x02;
568     static const uint8_t gpiodGpio0High                     = 0x01;
569
570
571
572     /* Register 0x09 (ID) definition
573     *
        ---------------------------------------------------------------------------
574     * |  Bit 7  |  Bit 6  |  Bit 5  |  Bit 4  |  Bit 3  |  Bit 2  |  Bit 1  |  Bit 0
        |
575     *
        ---------------------------------------------------------------------------
576     * |                                   ID[7:0]
        |
577     *
        ---------------------------------------------------------------------------
578     */
579
580     /** ID register address */
581     static const uint8_t regAddrId                          = 0x09;
```

```
582
583     /* ID register field masks */
584     static const uint8_t idId4Mask                          = 0x10;
585
586     /* ID4 field values */
587     static const uint8_t idId4Ads1258                       = 0x00;
588     static const uint8_t idId4Ads1158                       = 0x10;
589
590
591     //*****************************************************************************
592     //
593     // Constants
594     //
595     //*****************************************************************************
596     static const uint8_t numRegisters                       = 10;
597     /**
598      * Timeout in milliseconds of the function getSample().
599      * If the ADS1258 doesn't respond within this time with a data ready signal,
600      * the function will abort with an error.
601      */
602     static const uint32_t timeoutGetSample                   = 2000;
603
604
605 private:
606
607     //internal register map for recalling configuration
608     uint8_t mRegisterMap[numRegisters];
609
610     //spi object for communication with ADS1258-ep
611     hal::adu::Spi& mSpi;
612
613     //ports:
614     //can not be const, since they are changed in the DioLib. The port configuration
        shouldn't be changed and regarded as const
615     ADI_GPIO_TypeDef* mDrdyPort;
616     const int mDrdyPin;
617     ADI_GPIO_TypeDef* mCsPort;
618     const int mCsPin;
619     ADI_GPIO_TypeDef* mStartPort;
620     const int mStartPin;
621     ADI_GPIO_TypeDef* mResetPort;
622     const int mResetPin;
623
624
625     /**
626      * Settings of number of samples which will be thrown away after a channel switch.
627      * Not const, because a second ads1258.init() could write another configuration
         and change it.
628      */
629     int mNumSamplesDiscard;
630
631
632
633     /**
634      * @brief Sends the read command and retrieves STATUS (if enabled) and data
635      *
636      * @param status[] pointer to address where STATUS byte will be stored
637      * @param data[] pointer to starting address where data bytes will be stored
638      * @param mode ReadMode typedef to select which read operation to use
639      * @return 32-bit sign-extended conversion result (data only)
640      */
641     int32_t readData(uint8_t status[], uint8_t data[], ReadMode mode);
642
643
644
645     /**
646      *
647      * @brief Reads contents of a single register at the specified address
648      *
649      * @param addr address of the register to read
650      * @return 8-bit register read result
651      */
652     uint8_t readSingleRegister(uint8_t address);
```

```
653
654
655
656     /**
657      * @brief Reads a group of registers starting at the specified address
658      *
659      * @param startAddress register address from which we start reading
660      * @param count number of registers we want to read
661      *
662      * NOTE: Use getRegisterValue() to retrieve the read values
663      */
664     void    readMultipleRegisters(uint8_t startAddress, uint8_t count);
665
666
667
668     /**
669      *
670      * @brief Writes data to a single register at the specified address
671      *
672      * @param address The address of the register to write
673      * @param data The 8-bit data to write to the register
674      */
675     void    writeSingleRegister(uint8_t address, uint8_t data);
676
677
678
679     /**
680      * @brief Writes data to a group of registers
681      *
682      * @param startAddress register address from which we start write
683      * @param count number of registers we want to write to
684      * @param regData Array that holds the data to write, where element zero is the
         data to write to the starting address.
685      * NOTES:
686      * - Use getRegisterValue() to retrieve the written values.
687      * - Registers should be re-read after a write operaiton to ensure proper
         configuration.
688      */
689     void    writeMultipleRegisters(uint8_t startAddress, uint8_t count, const uint8_t
        regData[]);
690
691
692
693     /**
694      * @brief Sends the specified SPI command to the ADC
695      *
696      * @param opCode SPI command byte
697      */
698     void    sendCommand(uint8_t opCode);
699
700
701
702     /**
703      * @brief Helper function for checking that after powerup all registers are at
         default setting.
704      *
705      * If an error occurs here, most likely there is an error with the SPI interface.
706      */
707     bool    checkPowerup(void);
708
709
710
711     /**
712      * @brief Updates the mRegisterMap[] array to its default values.
713      *
714      * NOTES:
715      * - If the MCU keeps a copy of the ADC register settings in memory,
716      * then it is important to ensure that these values remain in sync with the
717      * actual hardware settings. In order to help facilitate this, this function
718      * should be called after powering up or resetting the device (either by
719      * hardware pin control or SPI software command).
720      *
721      * - Reading back all of the registers after resetting the device will
```

```
722         * accomplish the same result.
723         */
724        void    restoreRegisterDefaults(void);
725
726
727        //Internal Pin Set/Cls
728        inline void setCsPin(void)
729        {
730            DioSet(mCsPort, mCsPin);
731        }
732
733        inline void clearCsPin(void)
734        {
735            DioClr(mCsPort, mCsPin);
736        }
737
738        inline void setStartPin(void)
739        {
740            DioSet(mStartPort, mStartPin);
741        }
742
743        inline void clearStartPin(void)
744        {
745            DioClr(mStartPort, mStartPin);
746        }
747
748        inline void setResetPin(void)
749        {
750            DioSet(mResetPort, mResetPin);
751        }
752
753        inline void clearResetPin(void)
754        {
755            DioClr(mResetPort, mResetPin);
756        }
757
758        /**
759         * @brief Internal Pin Read of data read pin
760         * @return 0 if drdy pin is LOW,
761         *          not 0 if drdy pin HIGH
762         */
763        inline int isDrdyPin(void)
764        {
765            return DioRd(mDrdyPort) & mDrdyPin;
766        }
767
768        /** Register bit checking macros, helper functions.
769         *
770         *  Return true if register bit is set (since last read or write).
771         */
772        inline bool isMuxmodSet(void)
773        {
774            return (getRegisterValue(regAddrConfig0) & config0MuxmodMask);
775        }
776        inline bool isStatSet(void)
777        {
778            return (getRegisterValue(regAddrConfig0) & config0StatMask);
779        }
780
781    }; //end class Ads1258
782    }  // namespace sensors
783
784
785    #endif //SRC_SENSORS_ADS1258_H_
```

A.5.2. Ads1258.cpp

```
1   /*
2    * ads1258.cpp
3    *
4    *  Created on: 02.02.2023
5    *      Author: Luca Jungnickel
6    *      DLR-PF
7    */
8
9   #include "ads1258.h"
10  extern "C"
11  {
12  #include <hal/aducrf101/PRINTF/PRINTF.h>
13  #include <hal/aducrf101/clock.h>
14  }
15
16  /**
17   * Activates / deactivates the PRINTFs in this file for better debugging.
18   * Can be removed in production / optimized away by compiler
19   */
20  const bool debugPrints = false;
21
22
23
24  sensors::Ads1258::Ads1258(hal::adu::Spi& spiIn,
25          ADI_GPIO_TypeDef* drdyPortIn,   const int drdyPinIn,
26          ADI_GPIO_TypeDef* csPortIn,     const int csPinIn,
27          ADI_GPIO_TypeDef* startPortIn,  const int startPinIn,
28          ADI_GPIO_TypeDef* resetPortIn,  const int resetPinIn) :
29          mSpi(spiIn),
30          mDrdyPort(drdyPortIn),      mDrdyPin(drdyPinIn),
31          mCsPort(csPortIn),          mCsPin(csPinIn),
32          mStartPort(startPortIn),    mStartPin(startPinIn),
33          mResetPort(resetPortIn),    mResetPin(resetPinIn),
34          mNumSamplesDiscard(0)
35  {
36  }
37
38
39
40  /**
41   *  Initialises the ADS1258.
42   *  Resets the chip, writes the given configuration and reads it's back for
     verification.
43   *  If all registers are correctly set, true will be returned.
44   *  If this function returns false, the function should be called again.
45   *
46   * @param regConfig[]
47   *  Array of configuration of the registers. Has to be the size of (numRegisters - 1).
48   *  The order of configuration is: (corresponds to the address of the registers,
     starting from 0x0)
49   *  regAddrConfig0
50   *  regAddrConfig1
51   *  regAddrMuxsch
52   *  regAddrMuxdif
53   *  regAddrMuxsg0
54   *  regAddrMuxsg1
55   *  regAddrSysred
56   *  regAddrGpioc
57   *  regAddrGpiod
58   *
59   * @param samplesDiscard
60   *  Number of samples which will be discarded after a channel switch was performed.
61   *  @see setChannel()
62   *
63   * @return true if successful,
64   *         false if an error occured
65   */
66  bool sensors::Ads1258::init(uint8_t regConfig[], const int samplesDiscard)
67  {
68      /*
69       * ADC has a startup time of maximum 2^18 / f_clk = 17ms.
70       * (Datasheet p.25, Power-up Timing, Table 7, Worst Case if power rails are
         powered)
```

```
71          *
72          * Therefore, 20ms are chosen for the startup delay.
73          * The time can be decreased if this init() function is not called immediately
            after to µC powers up, 20ms is the worst case time.
74          */
75         delayMsec(20);
76
77         mNumSamplesDiscard = samplesDiscard;
78
79         if (debugPrints)
80         {
81             PRINTF("Init Routine\n");
82         }
83
84         //Toggle Reset: (reset pin has inverted logic)
85         clearResetPin();
86         delayMsec(1);
87         setResetPin();
88
89         // Ensure internal register array is initialized
90         restoreRegisterDefaults();
91         setCsPin(); //deactivate SPI bus
92
93         delayMsec(5); //maybe a lower delay is also possible, but this is currently not
           tested
94
95         //check if default registers are correctly set
96         if (!checkPowerup()) return false;
97         if (debugPrints)
98         {
99             PRINTF("Standard register read test successful\n");
100        }
101        //write configuration
102        writeMultipleRegisters(regAddrConfig0, numRegisters - 1, regConfig);
103
104        //Read settings and assert that they are correct with the written settings
105        readMultipleRegisters(regAddrConfig0, numRegisters);
106        for (int i = 0; i < numRegisters; i++)
107        {
108            if (regConfig[i] != getRegisterValue(i)) //error
109            {
110                if (debugPrints)
111                {
112                    PRINTF("Wrong data in reg %i, is %x, should %x\n", i, getRegisterValue
                       (i), regConfig[i]);
113                    PRINTF("Bootup with errors\n");
114                }
115                return false;
116            }
117        }
118
119        if (debugPrints)
120        {
121            PRINTF("Bootup successful\n");
122        }
123
124        delayMsec(1);
125
126        return true;
127    }
128
129
130
131    /**
132     *
133     * @brief Fetches one sample from the ADC and writes it to the given pointer.
134     *
135     * @param data pointer which will store the result
136     *
137     * @return true if the sampling was successful,
138     *         false if an error occured
139     *
140     */
```

```
141  bool sensors::Ads1258::getSample(int32_t& data)
142  {
143      clearCsPin(); //activate SPI bus
144
145
146      const uint32_t timeLimit = getMsSinceBoot() + timeoutGetSample;
147
148      while (isDrdyPin() != 0)
149      {
150          if (getMsSinceBoot() >= timeLimit) //timeout
151          {
152              setCsPin(); //deactivate SPI bus
153              return false;
154          }
155          else
156          {
157             asm("nop");
158          }
159      }
160
161      /**
162       * WARNING: currently only the mode command is working. The other mode, direct,
          is currently
163       * producing bad data and should not be used.
164       *
165       * See Gitlab Issue #7.
166       *
167       */
168      const ReadMode mode     = command;
169      uint8_t pStatus[0]   = {}; //status array won't be given back
170      uint8_t pData[0]     = {}; //data array won't be given back
171      int32_t dataRead = readData(pStatus, pData, mode);
172
173      setCsPin(); //deactivate SPI bus
174
175      if (debugPrints)
176      {
177          PRINTF("Got sample: 0x%x\n", dataRead);
178      }
179      data = dataRead;
180      /**
181       * In the current implementation the function will always return true, but
          further versions could
182       * also return false when an error occur.
183       */
184      return true;
185  }
186
187
188
189  /**
190   * @brief Fetches the given amount of samples from the ADC, calculates the average
       (moving-average),
191   * and writes the result to the given pointer.
192   *
193   * @param data pointer for the result
194   * @param num_samples number of samples which will be performed
195   *
196   * @return true if the sampling was successful,
197   *         false if an error occured
198      */
199  bool sensors::Ads1258::getDataAndOversample(double& data, uint32_t num_samples)
200  {
201      if (debugPrints)
202      {
203          PRINTF("Start oversampling\n");
204      }
205
206      int64_t result = 0;
207      for (uint32_t i = 0; i < num_samples; i++)
208      {
209          int32_t r = 0;
210          if (!getSample(r))
```

```
211         { //getSample failed
212             if (debugPrints)
213             {
214                 PRINTF("getSample results in an error\n");
215             }
216             data = 0;
217             return false;
218         }
219         result = result + static_cast<int64_t>(r);
220     }
221     double divided = static_cast<double>(result) / static_cast<double>(num_samples);
222     if (debugPrints)
223     {
224         PRINTF("Result now: %lld (0x%x)\n", result, result);
225         PRINTF("Divided with %i its %lf (0x%x)\n", num_samples, divided, result);
226     }
227     data = divided;
228     return true;
229 }
230
231
232
233 /**
234  * @brief   Sets the channel combination for measuring in differential mode.
235  * The writing to the ADC1258 will not be read-back, it will be assumed that it
     worked and the ADC
236  * was correctly initialised before.
237  *
238  * After a successful configuration, the amount of mNumSamplesDiscard of samples will
     be taken and discarded.
239  * Therefore, the radiometer has to be active for reading data
     (rad.startConversions()).
240  * @see mNumSamplesDiscard
241  *
242  * @param   channelCombination i.e. (MUXSCH_AINP_AIN7 | MUXSCH_AINN_AIN6)
243  * @see     MUXSCH_AINP_... / MUXSCH_AINN_... / CHANNELS_...
244  *
245  * Call rad.init() before.
246  * Call rad.startConversions() before.
247  *
248  * @return true if successful,
249  *         false if an error occured
250     */
251 bool sensors::Ads1258::setChannel(uint8_t channelCombination)
252 {
253     writeSingleRegister(regAddrMuxsch, channelCombination);
254     const uint8_t back = readSingleRegister(regAddrMuxsch);
255     if (back != channelCombination)
256     {
257         if (debugPrints)
258         {
259             PRINTF("Wrong data, is %x, should %x\n", back, channelCombination);
260         }
261         return false;
262     }
263     else
264     {
265         if (debugPrints)
266         {
267             PRINTF("Wrote successfully %x to register MuxSch\n", back);
268         }
269     }
270
271     //Throws first samples away
272     for (int i = 0; i < mNumSamplesDiscard; i++)
273     {
274         int32_t d = 0;
275         if (!getSample(d))
276         { //getSample failed
277             if (debugPrints)
278             {
279                 PRINTF("getSample failed after a channel switch\n");
280             }
```

```
281                 return false;
282             }
283         }
284
285         return true;
286     }
287
288
289
290     /**
291      * @brief   Activate conversions of the Radiometer.
292      *
293      * This will activate the ADS1258-ep with setting the START pin to HIGH.
294      * This function has to be called before any data is read from the Radiometer.
295      */
296     void sensors::Ads1258::startConversions()
297     {
298         setStartPin();
299     }
300
301
302
303     /**
304      * @brief   Deactivate conversions of the Radiometer.
305      *
306      * This will deactivate the ADS1258-ep with setting the START pin to LOW.
307      * Following data reads will then result 0x0, until the conversions are started again.
308      */
309     void sensors::Ads1258::stopConversions()
310     {
311         clearStartPin();
312     }
313
314
315
316     /**
317      * @brief Getter function to access the set registers of the ADC without reading
         directly from it.
318      *
319      * A local copy of the currently set registers will be accessed for that.
320      *
321      * @param address register address, i.e. regAddrConfig0
322      * @return value of the register
323      */
324     uint8_t sensors::Ads1258::getRegisterValue(uint8_t address)
325     {
326         if (debugPrints) {
327             if (address >= numRegisters) PRINTF("ERROR: address 0x%x in getRegisterValue
             Out of Bounds.\n", address);
328         }
329
330         return mRegisterMap[address];
331     }
332
333
334
335     //*****************************************************************************
336     //
337     // Private Functions
338     //
339     //*****************************************************************************
340
341     /**
342      * @brief Sends the read command and retrieves STATUS (if enabled) and data
343      *
344      * @param status[] pointer to address where STATUS byte will be stored
345      * @param data[] pointer to starting address where data bytes will be stored
346      * @param mode ReadMode typedef to select which read operation to use
347      * @return 32-bit sign-extended conversion result (data only)
348      */
349     int32_t sensors::Ads1258::readData(uint8_t status[], uint8_t data[], ReadMode mode)
350     {
351         uint8_t DataTx[5] = { 0 };    // Initialize all array elements to 0
```

```
352     uint8_t DataRx[5] = { 0 };    // Relies on C99 [$6.7.8/10], [$6.7.8/21]
353     uint8_t byteLength          = 0;
354     uint8_t dataPosition        = 0;
355     bool    statusByteEnabled = false;
356
357     /* Build TX array and send it */
358     if (mode == direct)
359     {
360         DataTx[0] = opcodeReadDirect;
361
362         /* STATUS byte enable depends on MUXMOD and STAT (see table 17 in datasheet)
            */
363         statusByteEnabled = isStatSet() & !isMuxmodSet();
364         byteLength = (statusByteEnabled ? 4 : 3);
365         dataPosition = (statusByteEnabled ? 1 : 0);
366     }
367     else if (mode == command) //WORKING
368     {
369         DataTx[0] = opcodeReadCommand | opcodeMulMask;
370
371         /* STATUS byte always enabled, but undefined in fixed-channel mode (see table
            17 in datasheet) */
372         statusByteEnabled = true;
373         byteLength = 5;
374         dataPosition = 2;
375     }
376     //No final else needed: only two modes exist
377
378     clearCsPin(); //activate SPI bus
379
380     /* Send and receive bytes from SPI */
381     for (int i = 0; i < byteLength; i++)
382     {
383         mSpi.send(DataTx[i]);
384
385         while ((SpiSta(pADI_SPI1) & SPISTA_RXFSTA_MSK) == SPISTA_RXFSTA_EMPTY) { asm(
            "NOP"); }
386         DataRx[i] = mSpi.receive();
387     }
388
389     setCsPin(); //deactivate SPI bus
390
391     //
392     // Parse returned SPI data
393     //
394
395     /* Check if STATUS byte is enabled and if we have a valid "status" memory pointer
        */
396     if (statusByteEnabled && status)
397     {
398         status[0] = DataRx[dataPosition - 1];
399     }
400
401     /* Check for "data" pointer before making assignments */
402     if (data)
403     {
404         data[0] = DataRx[dataPosition + 0];
405         data[1] = DataRx[dataPosition + 1];
406         data[2] = DataRx[dataPosition + 2];
407     }
408
409     /* Return the 32-bit sign-extended conversion result */
410     int32_t signByte;
411     if (DataRx[dataPosition] & 0x80u)   { signByte = 0xFF000000; }
412     else                                { signByte = 0x00000000; }
413
414     const int32_t upperByte   = ( static_cast<int32_t>(DataRx[dataPosition + 0]) &
        0xFF) << 16;
415     const int32_t middleByte  = ( static_cast<int32_t>(DataRx[dataPosition + 1]) &
        0xFF) << 8;
416     const int32_t lowerByte   = ( static_cast<int32_t>(DataRx[dataPosition + 2]) &
        0xFF) << 0;
417
```

```
418         return (signByte | upperByte | middleByte | lowerByte);
419     }
420
421
422
423     /**
424      *
425      * @brief Reads contents of a single register at the specified address
426      * @param addr address of the register to read
427      * @return 8-bit register read result
428      */
429     uint8_t sensors::Ads1258::readSingleRegister(uint8_t address)
430     {
431         clearCsPin(); //activate SPI bus
432
433         /* Initialize arrays */
434         uint8_t DataTx[2] = { 0 };
435         uint8_t DataRx[2] = { 0 };
436         /* Build TX array and send it */
437         DataTx[0] = (opcodeRreg | (address & opcodeAMask));
438
439         mSpi.send(DataTx[0]);
440         mSpi.send(DataTx[1]);
441
442         while ((SpiSta(pADI_SPI1) & SPISTA_RXFSTA_MSK) == SPISTA_RXFSTA_EMPTY) { asm("NOP"
            ); }
443         DataRx[0] = mSpi.receive();
444         while ((SpiSta(pADI_SPI1) & SPISTA_RXFSTA_MSK) == SPISTA_RXFSTA_EMPTY) { asm("NOP"
            ); }
445         DataRx[1] = mSpi.receive();
446
447         setCsPin(); //deactivate SPI bus
448
449         /* Update register array and return read result*/
450         mRegisterMap[address] = DataRx[1];
451
452         return DataRx[1];
453     }
454
455
456
457     /**
458      * \fn void readMultipleRegisters(uint8_t startAddress, uint8_t count)
459      * \brief Reads a group of registers starting at the specified address
460      * \param startAddress register address from which we start reading
461      * \param count number of registers we want to read
462      * NOTE: Use getRegisterValue() to retrieve the read values
463      */
464     void sensors::Ads1258::readMultipleRegisters(uint8_t startAddress, uint8_t count)
465     {
466
467         clearCsPin(); //activate SPI bus
468         // first send command byte
469         uint8_t dataTx = opcodeRreg | opcodeMulMask | (startAddress & opcodeAMask);
470         mSpi.send(dataTx);
471
472         while ((SpiSta(pADI_SPI1) & SPISTA_RXFSTA_MSK) == SPISTA_RXFSTA_EMPTY) { asm("NOP"
            ); }
473         mSpi.receive();
474
475         // receive bytes after that
476         for (uint8_t i = startAddress; i < startAddress + count; i++)
477         {
478             // Read register data bytes
479             mSpi.send(0x0); //payload doesn't matter
480             while ((SpiSta(pADI_SPI1) & SPISTA_RXFSTA_MSK) == SPISTA_RXFSTA_EMPTY) { asm(
                "NOP"); }
481             mRegisterMap[i] = mSpi.receive();
482         }
483
484         setCsPin(); //deactivate SPI bus
485     }
486
```

```
487
488  /**
489   *
490   * @brief Writes data to a single register at the specified address
491   *
492   * @param address The address of the register to write
493   * @param data The 8-bit data to write to the register
494   */
495  void sensors::Ads1258::writeSingleRegister(uint8_t address, uint8_t data)
496  {
497      clearCsPin(); //activate SPI bus
498
499      /* Initialize arrays */
500      uint8_t DataTx[2];
501
502      /* Build TX array and send it */
503      DataTx[0] = ( opcodeWreg | (address & opcodeAMask) );
504      DataTx[1] = data;
505
506      mSpi.send(DataTx[0]);
507      mSpi.send(DataTx[1]);
508
509      while ((SpiSta(pADI_SPI1) & SPISTA_RXFSTA_MSK) == SPISTA_RXFSTA_EMPTY) { asm("NOP"
         ); }
510      mSpi.receive();
511      while ((SpiSta(pADI_SPI1) & SPISTA_RXFSTA_MSK) == SPISTA_RXFSTA_EMPTY) { asm("NOP"
         ); }
512      mSpi.receive();
513
514      setCsPin(); //deactivate SPI bus
515
516      /* Update register array */
517      mRegisterMap[address] = DataTx[1];
518  }
519
520
521
522  /**
523   * @brief Writes data to a group of registers
524   *
525   * @param startAddress register address from which we start write
526   * @param count number of registers we want to write to
527   * @param regData Array that holds the data to write, where element zero is the data
      to write to the starting address.
528   * NOTES:
529   * - Use getRegisterValue() to retrieve the written values.
530   * - Registers should be re-read after a write operaiton to ensure proper
      configuration.
531   */
532  void sensors::Ads1258::writeMultipleRegisters(uint8_t startAddress, uint8_t count,
     const uint8_t regData[])
533  {
534      //
535      // SPI communication
536      //
537      clearCsPin(); //activate SPI bus
538      uint8_t dataTx = opcodeWreg | opcodeMulMask | (startAddress & opcodeAMask);
539
540      mSpi.send(dataTx);
541
542      while ((SpiSta(pADI_SPI1) & SPISTA_RXFSTA_MSK) == SPISTA_RXFSTA_EMPTY) { asm("NOP"
         ); }
543      mSpi.receive();
544
545      for (uint8_t i = startAddress; i < startAddress + count; i++)
546      {
547          mSpi.send(regData[i]);
548          // write register data bytes
549          while ((SpiSta(pADI_SPI1) & SPISTA_RXFSTA_MSK) == SPISTA_RXFSTA_EMPTY) { asm(
             "NOP"); }
550          mSpi.receive();
551
552          /* Update register array */
```

```
553             mRegisterMap[i] = regData[i];
554         }
555
556         setCsPin(); //deactivate SPI bus
557     }
558
559
560
561     /**
562      * @brief Sends the specified SPI command to the ADC
563      *
564      * @param opCode SPI command byte
565      */
566     void sensors::Ads1258::sendCommand(uint8_t opCode)
567     {
568         /* SPI communication */
569         clearCsPin(); //activate SPI bus
570
571         mSpi.send(opCode);
572
573         while ((SpiSta(pADI_SPI1) & SPISTA_RXFSTA_MSK) == SPISTA_RXFSTA_EMPTY) { asm("NOP"
        ); }
574         mSpi.receive();
575
576         setCsPin(); //deactivate SPI bus
577
578         // Check for RESET command
579         if (opcodeReset == opCode)
580         {
581             /* Update register array to keep software in sync with device */
582             restoreRegisterDefaults();
583         }
584     }
585
586
587
588     /**
589      * @brief Helper function for checking that after powerup all registers are at
         default setting.
590      *
591      * If an error occurs here, most likely there is an error with the SPI interface.
592      */
593     bool sensors::Ads1258::checkPowerup()
594     {
595         //Read default register data and assert that they are correct
596         uint8_t initRegisterMap[numRegisters];
597
598         initRegisterMap[regAddrConfig0]   =   config0Default;
599         initRegisterMap[regAddrConfig1]   =   config1Default;
600         initRegisterMap[regAddrMuxsch]    =   muxschDefault;
601         initRegisterMap[regAddrMuxdif]    =   muxdifDefault;
602         initRegisterMap[regAddrMuxsg0]    =   muxsg0Default;
603         initRegisterMap[regAddrMuxsg1]    =   muxsg1Default;
604         initRegisterMap[regAddrSysred]    =   sysredDefault;
605         initRegisterMap[regAddrGpioc]     =   gpiocDefault;
606         initRegisterMap[regAddrGpiod]     =   gpiodDefault;
607         initRegisterMap[regAddrId]        =   0x8b;
608
609         bool success = true;
610
611         readMultipleRegisters(regAddrConfig0, numRegisters - 1);
612
613         for (int i = 0; i < numRegisters - 1; i++)
614         {
615             if (mRegisterMap[i] != initRegisterMap[i])
616             {
617                 if (debugPrints)
618                 {
619                     PRINTF("Wrong default data in reg %i, is %x, should %x\n", i,
                    mRegisterMap[i], initRegisterMap[i]);
620                 }
621                 success = false;
622             }
```

```
623         }
624
625         if (!success)
626         {
627             if (debugPrints) PRINTF("Bootup with errors\n");
628         }
629         return success;
630     }
631
632
633
634     /**
635      * @brief Updates the mRegisterMap[] array to its default values.
636      *
637      * NOTES:
638      * - If the MCU keeps a copy of the ADC register settings in memory,
639      * then it is important to ensure that these values remain in sync with the
640      * actual hardware settings. In order to help facilitate this, this function
641      * should be called after powering up or resetting the device (either by
642      * hardware pin control or SPI software command).
643      *
644      * - Reading back all of the registers after resetting the device will
645      * accomplish the same result.
646      */
647     void sensors::Ads1258::restoreRegisterDefaults()
648     {
649         mRegisterMap[regAddrConfig0]   =   config0Default;
650         mRegisterMap[regAddrConfig1]   =   config1Default;
651         mRegisterMap[regAddrMuxsch]    =   muxschDefault;
652         mRegisterMap[regAddrMuxdif]    =   muxdifDefault;
653         mRegisterMap[regAddrMuxsg0]    =   muxsg0Default;
654         mRegisterMap[regAddrMuxsg1]    =   muxsg1Default;
655         mRegisterMap[regAddrSysred]    =   sysredDefault;
656         mRegisterMap[regAddrGpioc]     =   gpiocDefault;
657         mRegisterMap[regAddrGpiod]     =   gpiodDefault;
658         mRegisterMap[regAddrId]        =   0x00; // Value of 0x00 indicates that we have
            not yet read the ID register
659     }
```

A.5.3. Rad.h

```
1   /*
2    * rad.h
3    *
4    *  Created on: 20.12.2022
5    *      Author: Luca Jungnickel
6    *      DLR-PF
7    *
8    *  Combines the adapted driver from TI with own functionality for the radiometer
     Mable-Rad.
9    */
10
11
12
13
14  #ifndef SRC_SENSORS_RAD_H_
15  #define SRC_SENSORS_RAD_H_
16
17  #include <sensors/radiometer/ads1258.h>
18
19
20
21
22
23
24
25  namespace sensors
26  {
27  class Rad
28  {
29  public:
30
31      Rad(sensors::Ads1258& ads1258In);
32
33
34      /*
35      * These channel combinations should be used for setting the channels of the next
        measurement.
36      */
37      static const uint8_t channelsThermopile1        = ( Ads1258::muxschAinpAin0   |
        Ads1258::muxschAinnAin1  );
38      static const uint8_t channelsThermopile2        = ( Ads1258::muxschAinpAin11  |
        Ads1258::muxschAinnAin10 );
39      static const uint8_t channelsThermopile3        = ( Ads1258::muxschAinpAin4   |
        Ads1258::muxschAinnAin5  );
40      static const uint8_t channelsThermopile1Pt      = ( Ads1258::muxschAinpAin14  |
        Ads1258::muxschAinnAin15 );
41      static const uint8_t channelsThermopile2Pt      = ( Ads1258::muxschAinpAin12  |
        Ads1258::muxschAinnAin13 );
42      static const uint8_t channelsThermopile3Pt      = ( Ads1258::muxschAinpAin2   |
        Ads1258::muxschAinnAin3  );
43      static const uint8_t channelsPtExtern           = ( Ads1258::muxschAinpAin7   |
        Ads1258::muxschAinnAin6  );
44      static const uint8_t channelsOffset             = ( Ads1258::muxschAinpAin8   |
        Ads1258::muxschAinnAin9  );
45
46
47
48
49      const uint8_t measuringSequenceSize             = 8;
50      /*
51       * Sequence which channels the radiometer will measure when calling the measure()
         function.
52       */
53      const uint8_t measuringSequence[8]              = {
54                                                          channelsThermopile1,
                                                            channelsThermopile1Pt,
55                                                          channelsThermopile2,
                                                            channelsThermopile2Pt,
56                                                          channelsThermopile3,
                                                            channelsThermopile3Pt,
57                                                          channelsPtExtern,
                                                            channelsOffset
58                                                      };
```

```
59
60      /**
61       * Initializes the radiometer.
62       * Writes the configuration to the ADC with calling ads1258.init().
63       *
64       * @return true if successful,
65       *         false if an error occured
66       *
67       */
68      bool init();
69
70
71
72      /**
73       * @brief Performs the preprogrammed measurement procedure.
74       *
75       * Should be used to gather data from the radiometer.
76       *
77       * Each channel of the radiometer will be measured and integrated 1s, then the
         channel will be switched.
78       * The execution of this command should take around 8s.
79       *
80       * @param data pointer to array which will store the results. The array has to be
         minimum 8 elements big.
81       * @see measuringSequence, to get the order of the stored samples
82       *
83       * @param timestamps pointer to array which will store the timestamps of the
         corresponding sample.
84       *      The array has to be minimum 8 (measuringSequenceSize) elements big.
85       *
86       * @return true if the sampling was successful,
87       *         false if an error occured. Then all data should be discarded.
88       */
89      bool measure(double data[], uint32_t timestamps[]);
90
91
92  private:
93      sensors::Ads1258& mAds1258;
94
95  }; //end class Rad
96  }  // namespace sensors
97
98
99
100 #endif /* SRC_SENSORS_RAD_H_ */
101
```

A.5.4. Rad.cpp

```
/*
 * rad.cpp
 *
 *  Created on: 20.12.2022
 *      Author: Luca Jungnickel
 *      DLR-PF
 */

#include "rad.h"

extern "C"
{
#include <hal/aducrf101/PRINTF/PRINTF.h>
#include <hal/aducrf101/clock.h>
}



sensors::Rad::Rad(sensors::Ads1258& ads1258In) : mAds1258(ads1258In)
{
}


/**
 * Activates / deactivates the PRINTFs in this file for better debugging.
 * Can be removed in production / optimized away by compiler
 */
const bool debugPrints = false;

/**
 * Initializes the radiometer.
 * Writes the configuration to the ADC with calling ads1258.init().
 *
 * @return true if successful,
 *         false if an error occured
 *
 */
bool sensors::Rad::init()
{

    //Configuration for radiometer is here:
    uint8_t config[mAds1258.numRegisters];
    config[mAds1258.regAddrConfig0]   =   mAds1258.config0MuxmodMask | mAds1258.config0BypasMask | mAds1258.config0ChopMask; //Fixed-Channel, Bypass Active, Choping Active
    config[mAds1258.regAddrConfig1]   =   mAds1258.config1Dly384us | mAds1258.config1Drate1953sps; //Highest Delay, lowest data rate
    config[mAds1258.regAddrMuxsch]    =   channelsThermopile1;
    config[mAds1258.regAddrMuxdif]    =   mAds1258.muxdifDefault; //not available when MUXMOD=1(fixed-channel mode)
    config[mAds1258.regAddrMuxsg0]    =   mAds1258.muxsg0Default; //not available when MUXMOD=1(fixed-channel mode)
    config[mAds1258.regAddrMuxsg1]    =   mAds1258.muxsg1Default; //not available when MUXMOD=1(fixed-channel mode)
    config[mAds1258.regAddrSysred]    =   mAds1258.sysredDefault; //not available when MUXMOD=1(fixed-channel mode)
    config[mAds1258.regAddrGpioc]     =   0x0; //all GPIOs are output
    config[mAds1258.regAddrGpiod]     =   mAds1258.gpiodDefault; //all GPIOs are 0 / not driven
    config[mAds1258.regAddrId]        =   0x8B;           // Read-only register, 0x8B is default
    const int samplesDiscard = 32;

    if (debugPrints)
    {
        PRINTF("ads1258 will be inited\n");
    }
    return mAds1258.init(config, samplesDiscard);

}
```

```
65  /**
66   * @brief Performs the preprogrammed measurement procedure.
67   *
68   * Should be used to gather data from the radiometer.
69   *
70   * Each channel of the radiometer will be measured and integrated 1s, then the
     channel will be switched.
71   * The execution of this command should take around 8s.
72   *
73   * @param data pointer to array which will store the results. The array has to be
     minimum 8 elements big.
74   * @see measuringSequence, to get the order of the stored samples
75   *
76   * @return true if the sampling was successful,
77   *         false if an error occured. Then all data should be discarded.
78      */
79  bool sensors::Rad::measure(double data[], uint32_t timestamps[])
80  {
81
82      /*
83       * For configuration:
84       *  Chop = 1
85       *  Drate = 0x00 (slowest data rate)
86       *  Delay = 0x111 (highest data rate)
87       *  External Oscillator -> 15.729MHz internal clock
88       *
89       * Leads to 528.457 SPs (formula 3 datasheet)
90       * -> 1.89230 ms / S
91       *
92       * If first 32 samples will be thrown away:
93       *
94       *
95       * -> 496 samples should be taken to get around 1s of total time on one channel
96       *      -> take 500 samples
97       */
98
99
100     const uint32_t oversampleRate = 500;
101     for (int i = 0; i < measuringSequenceSize; i++ )
102     {
103         if (!mAds1258.setChannel(measuringSequence[i]))
104         {
105             return false; //error
106         };
107         if (!mAds1258.getDataAndOversample(data[i], oversampleRate))
108         {
109             return false; //error
110         }
111         if (debugPrints)
112         {
113             PRINTF("data[%i] = %x\n", i, data[i]);
114         }
115         timestamps[i] = getMsSinceBoot();
116     }
117
118     return true;
119 }
```

A.5.5. Mable-rad-complete-test.cpp

```
/**
 * This integration test measures all channels of the Radiometer and formats it's output as a CSV table.
 *
 * Each channel will be integrated around 1s, this will be repeated as configured in num_measurements.
 * The result will be formatted as an CSV table and printed via the serial interface.
 * It takes around 8s for one output dump of measured data.
 *
 * When everything is done, the radiometer will be set to stop converting and the program will stay in
 * an infinity loop without additional output.
 *
 *
 * This test represents the measurement procedure which should be followed in production.
 *
 * @author  Luca Jungnickel
 *          DLR-PF
 *          30.01.2023
 */

extern "C"
{
#include <hal/aducrf101/WdtLib.h>
#include <hal/aducrf101/PRINTF/PRINTF.h>
#include <hal/aducrf101/clock.h>
}

#include <debug.h>
#include <hal/aducrf101/serial/serial.h>
#include <sensors/radiometer/rad.h>
#include <sensors/radiometer/ads1258.h>
#include <hal/aducrf101/port_config/port_config.h>

static void csvPrintHeader()
{
    PRINTF("Time [mS];");
    PRINTF("Channel P;");
    PRINTF("Channel N;");
    PRINTF("DATA HEX;");
    PRINTF("DATA INT;");
    PRINTF("VOLTAGE;");
    PRINTF("END\n");
}

static void csvPrintData(uint8_t channels, double data, uint32_t timestamp, sensors::Ads1258& ads1258)
{
    PRINTF("%u;", timestamp);
    uint8_t channelP =  (channels & ads1258.muxschAinpMask) >> 0x4;
    uint8_t channelN =  (channels & ads1258.muxschAinnMask) >> 0x0;
    PRINTF("0x%x;", channelP);
    PRINTF("0x%x;", channelN);
    PRINTF("0x%x;", data);
    PRINTF("%lf;", data);
    double voltage = (2.5f / 0x780000) * data;
    PRINTF("%lfV;", voltage);
    PRINTF("\n");
}

int main()
{
    /**
     *
     *        CONFIGURATION
     *      of this testcase
     *
     */
    int num_measurements = 50000; //Number of measurements which will be taken, 1 takes around 8s


```

```
68      WdtGo (T3CON_ENABLE_DIS);
69      SystemInit ();
70      SystemSetClockDivider (0);
71      clockInit (SystemGetClockFrequency ());
72      hal::adu::PortConfig portConfig;
73      portConfig.set (); //configures all ports, also radiometer ports
74      // Serial
75      hal::adu::Serial serial;
76      serial.init (115200);
77
78
79
80      // SPI (setup in spi.init() )
81      //J2-15 P0.2        Mosi (DIN for Mable-Rad)
82      //J2-16 P0.1        Sclk
83      //J2-17 P0.0        Miso (DOUT for Mable-Rad)
84      hal::adu::Spi spi;
85      spi.init ();
86
87      //setup Radiometer Pins
88      //J1-17 P4.4 RadDRDY
89      //J1-16 P4.3 RadStart
90      //J1-21 P3.2 RadReset
91      //J1-14 P4.1 RadCs
92      sensors::Ads1258 ads1258 (spi,
93          pADI_GP4, (1 << PIN4),//Drdy Pin
94          pADI_GP4, (1 << PIN1), //CS Pin
95          pADI_GP4, (1 << PIN3), //Start Pin
96          pADI_GP3, (1 << PIN2)  //reset Pin
97          );
98
99
100     sensors::Rad rad(ads1258);
101
102
103     if (!rad.init()) {
104         PRINTF("Error while initialising the ADS1258\n");
105         return 1;
106     }
107
108     //Print config
109     PRINTF("Config:\n");
110     for (int i = 0; i < ads1258.numRegisters - 1; i++)
111     {
112         PRINTF("Reg[%i] = 0x%x\n", i, ads1258.getRegisterValue(i));
113     }
114     PRINTF("Num Samples Discard = 0x%x\n", ads1258.getNumSamplesDiscard());
115     PRINTF("\n");
116
117     csvPrintHeader();
118
119     ads1258.startConversions();
120
121     for (int i = 0; i < num_measurements; i++)
122     {
123         double data[8]          = {0};
124         uint32_t timestamps[8]  = {0};
125
126         if (!rad.measure(data, timestamps))
127         {
128             PRINTF("Error, measuring didn't work\n");
129         }
130         else
131         {
132             for (int i2 = 0; i2 < rad.measuringSequenceSize; i2++)
133             {
134                 csvPrintData(rad.measuringSequence[i2], data[i2], timestamps[i2],
                    ads1258);
135             }
136             PRINTF("\n");
137         }
138     }
139     PRINTF("\nDONE!\n");
```

```
140
141         ads1258.stopConversions();
142
143         while (1) { }
144
145
146
147
148
149
150         return 0;
151     }
```

A.5.6. Mable-rad-one-channel-test.cpp

```
1   /**
2    * This integration test measures one channels of the Radiometer and formats it's
     output as a CSV table.
3    *
4    * When everything is done, the radiometer will be set to stop converting and the
     program will stay in
5    * an infinity loop without additional output.
6    *
7    * @author  Luca Jungnickel
8    *          DLR-PF
9    *          31.01.2023
10   */
11
12   extern "C"
13  {
14      #include <hal/aducrf101/WdtLib.h>
15      #include <hal/aducrf101/PRINTF/PRINTF.h>
16      #include <hal/aducrf101/clock.h>
17      #include <stdbool.h>
18  }
19  #include <debug.h>
20  #include <sensors/radiometer/rad.h>
21  #include <sensors/radiometer/ads1258.h>
22  #include <hal/aducrf101/ADUCRF101.h>
23  #include <hal/aducrf101/serial/serial.h>
24  #include <hal/aducrf101/port_config/port_config.h>
25
26  static void csvPrintHeader()
27  {
28      PRINTF("Time [mS];");
29      PRINTF("Channel P;");
30      PRINTF("Channel N;");
31      PRINTF("DATA HEX;");
32      PRINTF("DATA INT;");
33      PRINTF("VOLTAGE;");
34      PRINTF("END\n");
35  }
36
37  static void csvPrintData(uint8_t channels, double data, uint32_t timestamp, sensors::
    Ads1258& ads1258)
38  {
39      PRINTF("%u;", timestamp);
40      uint8_t channelP =  (channels & ads1258.muxschAinpMask) >> 0x4;
41      uint8_t channelN =  (channels & ads1258.muxschAinnMask) >> 0x0;
42      PRINTF("0x%x;", channelP);
43      PRINTF("0x%x;", channelN);
44      PRINTF("0x%x;", data);
45      PRINTF("%lf;", data);
46      double voltage = (2.5f / 0x780000) * data;
47      PRINTF("%lfV;", voltage);
48      PRINTF("\n");
49  }
50
51  int main()
52  {
53      /**
54       *
55       *        CONFIGURATION
56       *      of this testcase
57       *
58       */
59      const uint8_t channel = sensors::Rad::Rad::channelsPtExtern; //Select channel here
60
61      /*
62       * Number of samples per channel for oversampling.
63       * This amount will be measured and averaged.
64       *
65       * 500 are around 1s.
66       */
67      const uint32_t numSamples = 500;
68      const int numMeasurements = 6000; //Number of measurements which will be taken
69
70      WdtGo (T3CON_ENABLE_DIS);
```

```
71      SystemInit ();
72      SystemSetClockDivider (0);
73      clockInit (SystemGetClockFrequency ());
74      hal::adu::PortConfig portConfig;
75      portConfig.set (); //configures all ports, also radiometer ports
76      // Serial
77      hal::adu::Serial serial;
78      serial.init (115200);
79
80      // SPI (setup in spi.init() )
81      //J2-15 P0.2        Mosi (DIN for Mable-Rad)
82      //J2-16 P0.1        Sclk
83      //J2-17 P0.0        Miso (DOUT for Mable-Rad)
84      hal::adu::Spi spi;
85      spi.init ();
86
87      //setup Radiometer Pins
88      //J1-17 P4.4 RadDRDY
89      //J1-16 P4.3 RadStart
90      //J1-21 P3.2 RadReset
91      //J1-14 P4.1 RadCs
92      sensors::Ads1258 ads1258 (spi,
93          pADI_GP4, (1 << PIN4),//Drdy Pin
94          pADI_GP4, (1 << PIN1), //CS Pin
95          pADI_GP4, (1 << PIN3), //Start Pin
96          pADI_GP3, (1 << PIN2)  //reset Pin
97          );
98
99      sensors::Rad rad(ads1258);
100
101     //initialises the radiometer
102     if (!rad.init()) {
103         PRINTF("Error while initialising the ADS1258\n");
104         return 1;
105     }
106
107     //Print config
108     PRINTF("Config:\n");
109     for (int i = 0; i < ads1258.numRegisters - 1; i++)
110     {
111         PRINTF("Reg[%i] = 0x%x\n", i, ads1258.getRegisterValue(i));
112     }
113     PRINTF("Num Samples Discard = 0x%x\n", ads1258.getNumSamplesDiscard());
114     PRINTF("\n");
115
116
117     csvPrintHeader();
118
119     ads1258.startConversions();
120
121     if (!ads1258.setChannel(channel))
122     { //error
123         PRINTF("Error, channel switching didn't work\n");
124     }
125     for (int i = 0; i < numMeasurements; i++)
126     {
127
128         double data = 0;
129         if (!ads1258.getDataAndOversample(data, numSamples))
130         {
131             PRINTF("Error, measuring didn't work\n");
132         }
133         csvPrintData(channel, data, getMsSinceBoot(), ads1258);
134
135         PRINTF("\n");
136     }
137     PRINTF("\nDONE!\n");
138
139     ads1258.stopConversions();
140
141     while (1) { }
142
143
```

```
144
145
146         return 0;
147     }
148
```