Hes·SO VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Bereich Ingenieurwissenschaften
Rte du Rawyl 47
CH- 1950 Sion 2
Tel. +41 27 606 85 11
Fax +41 27 606 85 75
info@hevs.ch

www.hevs.ch

Studiengang Systemtechnik

Vertiefungsrichtung Infotronics

Diplom 2010

Dominic Furrer

TempArray

Dozent Joseph Moerschell

Experte Christian Briguet

Sitten, 12. Juli 2010

Es handelt sich um den Originalbericht des Studenten. Er wurde nicht korrigiert und kann folglich Fehler und ungenaue Angaben enthalten.

☒ FSI ☐ FTV	Année académique / *Studienjahr* **2009/10**	No TD / *Nr. DA* **it/2010/24**
Mandant / *Auftraggeber* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire	Etudiant / *Student* **Dominic Furrer**	Lieu d'exécution / *Ausführungsort* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire
Professeur / *Dozent* **Joseph Moerschell**	Expert / *Experte* *(données complètes)*	
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja [1] ☒ non / nein		

Titre / *Titel*

Thermische Bilanz durch verteilte Temperaturmessung

Description et Objectifs / *Beschreibung und Ziele*

Mit Hilfe einer hochauflösenden Temperaturmessung (<10-4K) mit einer Reihe verteilter Sensoren soll die Wärmeabgabe einer industriellen Anlage, im Beispiel eines Bioreaktors, charakterisiert werden.

Die Arbeit hat folgende Ziele:

— Realisierung einer hochaulösenden Temperaturmesskette mit mindestens 4 Kanälen
— Installation der Sensoren an einem Bioreaktor und Inbetriebnahme der Messkette
— Validierung eines thermischen Modells des Reaktors während einer Fermentation.

Délais / *Termine*

Attribution du thème / *Ausgabe des Auftrags:*
22.02.2010

Remise du rapport intermédiaire / *Zwischenbericht:*
07.05.2010, 17:00

Défense intermédiaire / *Zwischenverteidigung:*
21.05.2010

Remise du rapport / *Abgabe des Schlussberichts:*
12.07.2010, 12:00

Exposition publique / *Ausstellung Diplomarbeiten:*
27.08.2010

Défense orale / *Mündliche Verteidigung:*
Semaine 35 / *Woche 35*

Signature ou visa / *Unterschrift oder Visum*

Responsable de la filière
Leiter des Studiengangs: .....

[1] Etudiant/*Student*: Furrer Dominic

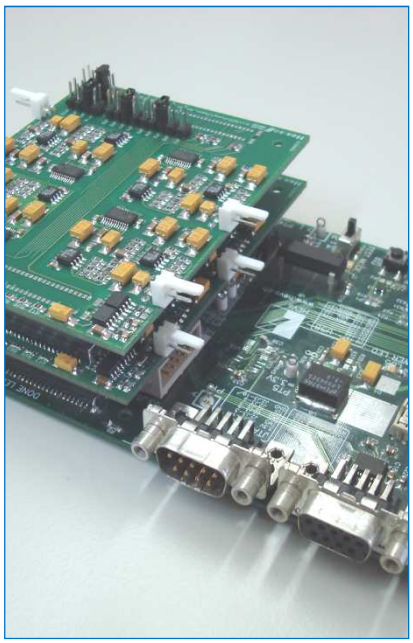
[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement la directive et le caractère confidentiel du travail de diplôme qui lui est confié et des informations mises à sa disposition.*
Durch seine Unterschrift verpflichtet sich der Student, die Richtlinie einzuhalten sowie die Vertraulichkeit der Diplomarbeit und der dafür zur Verfügung gestellten Informationen zu wahren.

Rapport reçu le / *Schlussbericht erhalten am* ..... Visa du secrétariat / *Visum des Sekretariats* .....

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

TempArray

Thermische Bilanz durch verteilte Temperaturmessung



Diplomand/in Dominic Furrer

Diplomarbeit
| 2010 |

Studiengang
Systemtechnik

Anwendungsbereich
Infotronic

Verantwortliche/r Dozent/in
Joseph Moerschell
moj@hevs.ch

Partner
-

Ziel des Projekts

Mit Hilfe einer hochauflösenden Temperaturmessung, mit einer Reiher verteilter Sensoren, wird die Wärmeabgabe einer industriellen Analage, im Beispiel eines Bioreaktors, charakterisiert.

Methoden / Experimente / Resultate

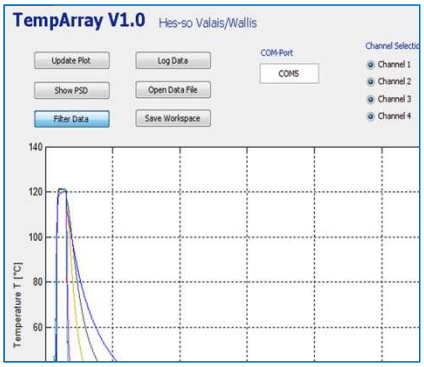
Das Projekt beinhaltet die Realisierung der Messkette, die Installation der Messkette an einem Bioreaktor und die Validierung eines thermischen Modells. Das Projekt reicht von analoger und digitaler Elektronik, über Programmierung mit Matlab bis zur Thermodynamik.

Ausgehend von einer bestehenden FPGA-Karte, wurden die Hardware und Firmware der Temperaturmessung, der Signalaufbereitung und der A/D-Umwandlung entwickelt. Es werden leistungsstarke A/D-Wandler verwendet um auch eine hochauflösende Temperaturmessung (10^{-4}K in -50℃…+150℃) durchführen zu können. Insbesondere kann die Temperaturverteilung innerhalb eines Bioreaktors mit einem dafür konzipierten modularen Sondenstab bestimmt werden. Am Ende der Messkette wird die Acquisition, Visualisierung und Analyse mit Matlab durchgeführt.

Die Messkette wurde an einem Bioreaktor installiert und in Betrieb genommen. Die aktuelle Ausführung der Messkette ermöglicht die Messung von 8 Kanälen.

Nach der Inbetriebnahme der Messkette wurden Fermentationen von Hefe durchgeführt welche dann mit der Messkette aufgezeichnet wurden. Die Validierung des thermischen Modells des Bioreaktors wurde dann mit diesen Daten durchgeführt.

Der gemessene und validierte Laborfermenter. Es handelt sich um das Modell NLF22 19 Liter von Bioengineering.



ese Abbildung zeigt einen sschnitt der entwickelten ftware zur Acquisition und alyse der Daten.

HES-SO Wallis
Route du Rawyl 47
1950 Sitten

Tel. 027 606 85 11
Web www.hevs.ch

Inhaltsverzeichnis

1. Einführung

1.1 Herkunft des Projektes

In der Biotechnologie werden biologische Materialien mit Hilfe von Fermentationen hergestellt oder umgewandelt. Die Qualität und der Erfolg der Prozesse hängen stark von der Temperatur ab. Aus dem Grund ist es wichtig, eine möglichst präzise Information über die Temperatur zu erhalten.

Die durch die Lebenmittelüberwachungsbehörde der Vereinigten Staaten (Food and Drug Administration, FDA) ins Leben gerufene Initiative namens Process Analytical Technology (PAT) erfordert eben solche Messungen, in diesem Fall die Temperaturmessung, um die überwachten Prozesse optimieren und analysieren zu können. So soll unter anderem die Produktqualität erhöht werden.

Da im Institut Life Technologies der Hes-so Valais / Wallis Sion auch solche Prozesse und Fermentationen durchgeführt werden entstand das Interesse an einer verteilten Temperaturmessung wie sie in Zuge dieses Projektes realisiert wird.

1.2 Auftrag und Zielsetzung

Der Auftrag für die Projektarbeit lautet folgendermassen.

Mit Hilfe einer hochauflösenden Temperaturmesssung (<10^{-4}K) mit einer Reiher verteilter Sensoren soll die Wärmeabgabe einer industriellen Anlage, im Beispiel eines Bioreaktors, charakterisiert werden.

Die Arbeit hat folgende Ziele:

- *Realisierung einer hochauflösenden Temperaturmesskette mit mindestens 4 Kanälen*
- *Installation der Sensoren an einem Bioreaktor und Inbetriebnahme der Messkette*
- *Validierung eines thermischen Modells des Reaktors während einer Fermentation*

Aus dem Auftrag ergaben sich also die folgenden Spezifikationen für die zu erstellende Messkette.

- Auflösung von maximal 10^{-4}K
- Temperaturmesskette mit mindestens 4 Kanälen

Während der ersten Phase der Projektarbeit werden noch weitere Spezifikationen erstellt. Diese sind an den entsprechenden Orten mit Erklärungen aufgezeigt.

Was allerdings vorweg noch als Ziel definiert werden konnte ist die Erstellung der Messkette von der Planung bis hin zu der gebrauchsfertigen Installation.

1.3 Rahmen des Projektes

Im Rahmen dieses Projektes wird eine Hardware erstellt, welche die Temperaturmessung durchführt. Zur Aufbereitung und Verarbeitung der Daten werden bestehende Elemente wie einem anwenderprogrammierbaren Logikbausteinen (Field Programmable Gate Array,

FPGA) auf einer bereits bestehenden Basiskarte und ein Einzelplatzrechner (Personal Computer, PC) verwendet.

Die Qualität der Schaltung soll bestimmt werden und eine Validierung eines thermischen Modells eines Bioreaktors wird durchgeführt. Hierzu muss die Installation an einem solchen Reaktor vorgenommen werden.

1.3.1 Planung

Das Projekt kann in zwei Phasen eingeteilt werden. Während dem 6. Semester konnte jeweils ein Tag pro Woche am Projekt gearbeitet werden. Dies ergab 7 Tage, welche im Vorfeld der Vollzeitarbeit genutzt werden konnten. Diese 7 Tage befanden sich in der Zeitspanne vom 24.02.2010 bis zum 07.05.2010. Am letzten Tag dieser Zeitspanne musste ein Zwischenbericht verfasst und abgegeben werden. Dieser Zwischenbericht, welcher eine sehr allgemeine Beschreibung des Projektes beinhaltet, befindet sich im *Anhang 14.1 Zwischenbericht*. Nach der Zeitspanne der Vorarbeit gab es auch eine Zwischenverteidigung.

Danach wurde während der gesamten zur Verfügung stehenden Zeit am Projekt gearbeitet. Dass heisst, vom 18.05.2010 bis zum 02.06.2010 wurde dann in Vollzeitarbeit gearbeitet.

Die Verteidigung der Diplomarbeit wird in der Woche vom 30.08.2010 bis zum 05.09.2010 durchgeführt.

Im Bericht wird nicht beschrieben zu welchem Zeitpunkt was gemacht wurde. Das bedeutet, der Chronologische Ablauf des Projektes ist im Bericht nicht wiedergegeben. Allerdings kann dieser der Planung des Projektes im Anhang *14.2 Planung* entnommen werden.

2. Funktionsprinzip

In diesem Kapitel wird die Grundidee der Messkette vermittelt. Die physikalischen Gegebenheiten wie auch die strukturellen und prinzipiellen Entscheide sollen aufgezeigt werden.

2.1 Messkette

2.1.1 Systemtechnischer Ansatz

Das Blockschaltbild in Abbildung 2-1 zeigt einen systemtechnischen Ansatz zur Erstellung der Messkette.

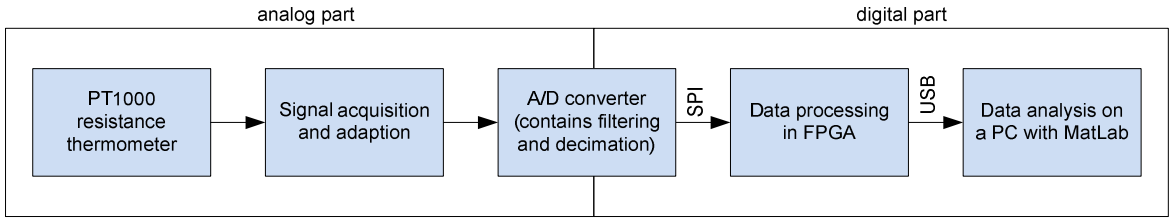


Abbildung 2-1: Systemtechnischer Ansatz der Messkette

Die Messkette beginnt mit einem PT1000-Widerstandsthermometer[1], welches die Temperatur misst. Das gemessene Signal wird mit einem analogen Schaltkreis aufgenommen und angepasst. Das angepasste Signal wird dann vom Analog-Digital-Wandler (Analog-to-Digital-Converter, ADC) in einen digitalen Wert, welcher wiederum dem gemessenen Signal entspricht, umgewandelt. Die so erhaltenen Werte werden über eine synchrone, serielle Schnittstelle (Serial Peripheral Interface, SPI) an die FPGA gesendet. In der FPGA werden die Daten weiter aufbereitet und zur Analyse mit dem Programm Matlab[2] vorbereitet. Die Daten gelangen über einen universellen seriellen Bus (Universal Serial Bus, USB) von der FPGA zum PC. Mit Hilfe von Matlab werden die Daten dann ausgewertet und analysiert.

Die Blöcke können ihrer Funktion nach in verschiedene Gruppen aufgeteilt werden. In der ersten Gruppe (Temperaturmessung) befindet sich der Block mit dem PT1000-Element. In der zweiten Gruppe (Datenaufnahme) befindet sich der analoge Schaltkreis, wie auch der A/D-Wandler. Die Datenverarbeitung, die dritte Gruppe, umfasst dann die FPGA. Und die Gruppe Datenauswertung beinhaltet den PC mit Matlab.

Die Unterteilung zwischen analogem und digitalem Teil ist auch zu erkennen. Diese ist für eine solch hochpräzise Anwendung notwendig.

Die einzelnen Teile des Systems werden in den folgenden Abschnitten erklärt.

[1] PT1000 Widerstandsthermometer: Temperaturfühler der auf der Widerstandsänderung von Platin durch den Temperatureinfluss basiert. Pt steht für Platinum und 1000 für den 0℃-Widerstand von 1k Ω

[2] Matlab (Eigenschreibweise: MATLAB): Software des Unternehmens The MathWorks, Inc. zur Lösung mathematischer, numerischer Probleme und zur grafischen Darstellung der Ergebnisse

2.1.2 Messtechnischer Ansatz

Die Messkette soll nun mit einem messtechnischen Ansatz gezeigt werden. Dabei handelt es sich um den PT1000-Widerstandsthermometer, die Signalanpassung und die A/D-Wandlung und die Datenverarbeitung und Analyse. Abbildung 2-2 zeigt diesen Ansatz.

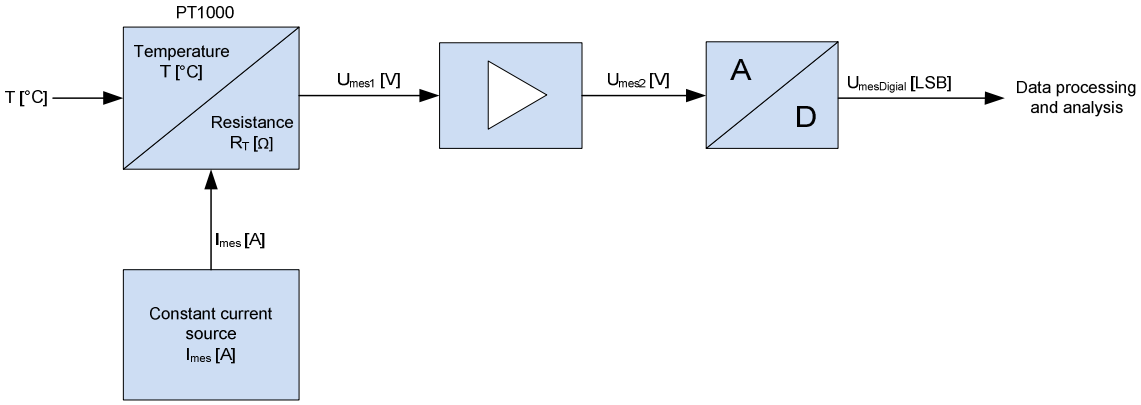


Abbildung 2-2: Messtechnische Ansatz der Messkette

Die Schnittstelle zwischen dem physikalischen Vorgang und dem elektrischen Kreis ist links oben zu sehen. Die danach folgende Signalbearbeitung und A/D-Wandlung sind auch ersichtlich. Zudem ist auch zu erkennen, dass die Information über die Temperatur, welche im Widerstand des PT1000-Widerstandsthermometers enthalten ist mit Hilfe einer Konstantstromquelle in eine Spannung umgewandelt wird. Detailliertere Informationen zu den einzelnen Blöcken der Messkette folgen in den nächsten Abschnitten.

2.1.3 Temperaturmessung

Da im Auftrag keine Vorgabe über den Messbereich vorhanden war, wird dieser zuerst definiert. Der Bereich wird auf -50℃ < T < 150℃ f estgelegt. Dieser Bereich wird gewählt, da es sich bei Gärungen usw. nicht um extremere Temperaturen als die gewählten handelt.

Die Messung der Temperatur wird mit PT1000-Widerstandsthermometern durchgeführt. Bei diesen Widerstandsthermometern handelt es sich um einen Platinwiderstand, welcher sich angenähert proportional zur Temperatur verhält.

PT1000-Widerstandsthermometer haben einen Nennwert von 1kΩ und verändern diesen anhand des Temperaturkoeffizienten α. Der Temperaturkoeffizient beträgt $3.85 \cdot 10^{-3}$℃ $^{-1}$ angegeben und besagt somit die Widerstandsänderung pro Grad Celsius.

Dieses Verhalten lässt sich mit der folgenden Formel beschreiben. Die Formel beschreibt die Widerstandsänderung in Funktion der Temperatur. Es handelt sich dabei jedoch um eine lineare Annäherung an das wirkliche Verhalten des Widerstands.

$$R_T = R_0 \cdot (1 + \alpha \cdot \Delta T)$$

ΔT beschreibt den Temperaturunterschied zum Nullpunkt 0℃.

Mit Hilfe dieser Formel kann die Kennlinie der Abbildung 2-3 erstellt werden.

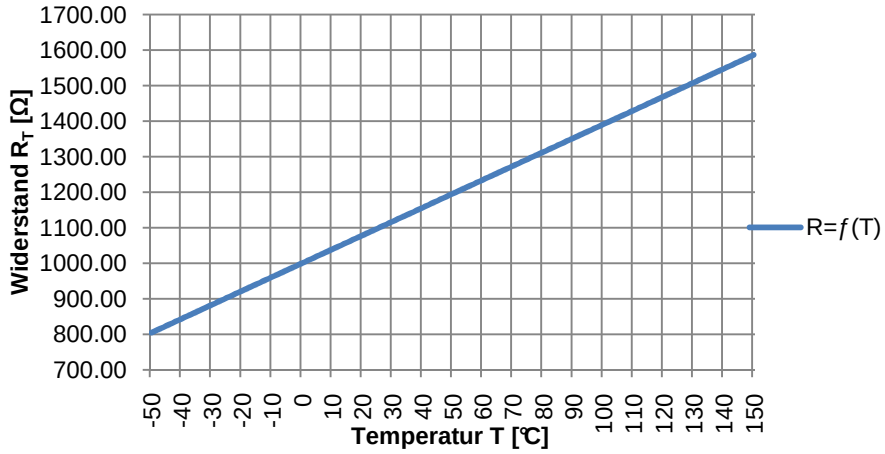


Abbildung 2-3: ideale Kennlinie des PT1000-Widerstandthermometers

Diese Funktion erlaubt es die Auswertung der Messwerte mit geringem rechnerischem Aufwand durchzuführen. Zu sehen ist auch der Widerstandsbereich, welcher durch den definierten Temperaturbereich vorgegeben wird. Der Widerstandsbereich beträgt $800\Omega < R_T < 1600\Omega$. Der Mittelpunkt des Bereiches beträgt $R_T = 1200\Omega$.

Um zu zeigen, dass das ideale anstelle des realen Verhaltens benutzt werden kann sollen die beiden Methoden folgend noch verglichen werden.

Falls das reale Verhalten beschrieben werden soll werden zwei Formeln benötigt. Die erste, welche für einen Bereich von 0℃<T<850℃ gültig ist , ist die folgende.

$$R_{T(0°C<T<850°C)} = R_0 \cdot (1 + \alpha \cdot \Delta T + \beta \cdot \Delta T^2)$$

Die zweite Formel ist für den Bereich -200℃<T<0℃.

$$R_{T(-200°C<T<0°C)} = R_0 \cdot (1 + \alpha \cdot \Delta T + \beta \cdot \Delta T^2 + \gamma \cdot (\Delta T - 100°C) \cdot \Delta T^3)$$

Des Weiteren sind noch die beiden Koeffizienten β und γ zu erwähnen. Dies sind auch materialabhängige Konstanten, und sie haben die folgenden Werte.

$$\beta = -5.775 \cdot 10^{-7}\ °C^{-2}$$

$$\gamma = -4.183 \cdot 10^{-12}\ °C^{-4}$$

Mit diesen Gleichungen lässt sich das Diagramm der Abbildung 2-4, in Welchem die reale und die ideale Kennlinie verglichen werden, erstellen.

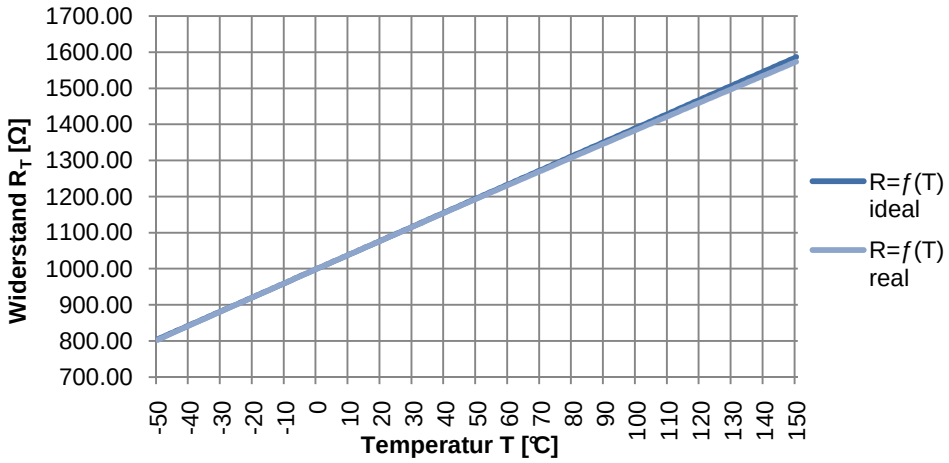


Abbildung 2-4: Vergleich zwischen idealer und realer Kennlinie des PT1000-Widerstandsthermometers

Wie zu sehen ist unterscheidet sich die Kennlinie des realen Verhaltens nur gering von dem des idealen. Aus diesem Grund wird mit der idealisierten Variante gearbeitet.

Weitere Details zu den gewählten PT1000-Widerstandsthermometern wie auch zu deren Beschaltung sind unter *3.2.3.2 PT1000-Widerstandsthermometer* vorzufinden.

2.1.4 Datenaufnahme

In diesem Teil der Messkette werden die Messwerte aufgenommen angepasst und für die Verarbeitung und Auswertung vorbereitet. Dies umfasst einen analogen Schaltkreis, welcher es ermöglicht die Temperaturabhängigkeit des Widerstands des Thermometers in eine Temperaturabhängigkeit einer erzeugten Spannung umzuformen. Diese Spannung wird danach angepasst. Das so erhaltene elektrische Messsignal wird dann von einem Analog-Digital-Wandler in digitale Werte umgewandelt.

Die soeben erwähnten Schritte erfordern mehrere Schaltungsteile. Als erstes wird eine Konstantstromquelle benötigt, welche mit Hilfe des PT1000-Widerstandthermometers eine zur Temperatur proportionale Spannung generiert. Der Konstantstrom beträgt ca. 1mA.

Die erzeugte Spannung wird dann mit einem Verstärker auf den für den Eingang des A/D-Wandlers erwünschten Wertebereich angepasst. Nebst diesen Verstärkern werden noch Filter benötigt, welche das Rauschen reduzieren sollen und so auch eine höhere Auflösung ermöglichen.

Weitere Informationen dazu sind unter *3.2.3.3 Signalanpassung* zu finden.

Der A/D-Wandler erstellt aus der angepassten Spannung einen digitalen Wert, welcher der gemessenen Grösse entspricht. Beim A/D-Wandler handelt es sich um einen Delta-Sigma-Wandler (ΔΣ-Wandler) von Texas Instruments, welcher sich durch einen grossen Störabstand wie auch durch andere gute Leistungen auszeichnet. Des Weiteren beinhaltet der A/D-Wandler noch drei Filterstufen. Die so erhaltenen Daten werden dann über eine SPI-Schnittstelle an die FPGA übermittelt. Weitere Daten zu dem gewählten A/D-Wandler und dessen Beschaltung befinden sich unter *3.3 Analog-Digital-Wandler*.

2.1.5 Datenverarbeitung

Die Datenverarbeitung wird mit einer FPGA realisiert. Es wird eine FPGA und kein Prozessor verwendet, da mehrere Messwerte zur selben Zeit, also parallel, verarbeiten werden müssen.

Demzufolge werden in der FPGA so viele parallele SPI-Schnittstellen wie Messkanäle vorhanden sind. Diese Schnittstellen sind da um die seriell empfangenen Daten zu parallelisieren. Danach werden die so erhaltenen Werte arithmetisch gemittelt, wodurch der Einfluss von fehlerhaften Messungen gemindert werden kann. Nachdem die Durchschnittswerte verarbeitet wurden, werden sie an eine USB-Schnittstelle übertragen, von wo aus sie dann an den PC weiter gesendet werden.

Die soeben aufgezeigten Operationen werden in der Abbildung 2-5 schematisch als Blockschaltbild dargestellt.

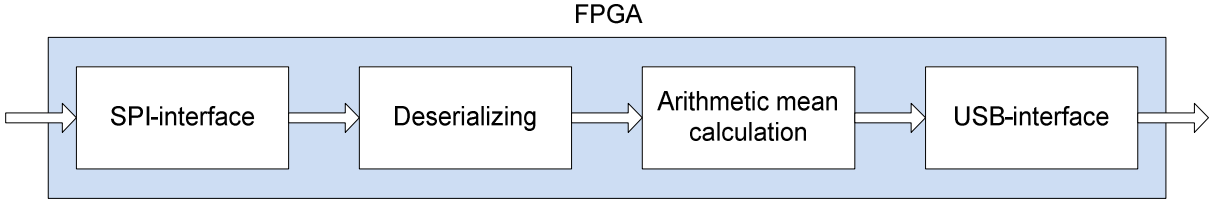


Abbildung 2-5: Funktionsprinzip der Datenverarbeitung in der FPGA

Die Idee wäre auch die FPGA als so genannten Controller zu verwenden. D.h. dass die FPGA für die Konfiguration der A/D-Wandler zuständig sein wird. Natürlich soll die Mittelung der Werte in der FPGA auch angepasst werden können.

Weitere Informationen im Bezug auf die FPGA sind unter *3.5 FPGA* vorzufinden.

2.1.6 Datenauswertung

De Datenauswertung und Analyse geschieht mit Hilfe eines PCs oder Laptops. Die USB-Verbindung erscheint auf dem PC als serieller Port. So ist es in Matlab mit geringem Aufwand möglich die Werte entgegenzunehmen.

Die Messwerte welche über USB empfangen werden befinden sich bereits im Zweierkomplement. Die Integer-Variablen in Matlab sind auch im Zweierkomplement kodiert. So können die Werte einfach aufgenommen und direkt genutzt werden.

Von den Spannungsmesswerten wird zurück auf die Temperatur gerechnet. Dies ergibt T_D. Diese Werte, oder auch die Spannungsmesswerte, können nun im Zeitbereich wie auch im Frequenzbereich dargestellt werden. Zudem erlaubt es Matlab die Messdaten in Form von Dateien abzuspeichern. Dabei kann es sich, je nach Bedarf, um Matlab-, Text- oder XML-Dateien handeln. So können die Daten zu einem späteren Zeitpunkt wieder verwendet werden.

Detailliertere Informationen und Erklärungen zu der USB Verbindung und zu Matlab können unter *3.6 USB-Verbindung* und unter *5 Entwicklung der Software* nachgeschlagen werden.

2.2 Herstellung

Wie die Messkette aufgebaut ist wurde soeben erklärt. Nun soll noch gezeigt werden, wie die Hardware hergestellt bzw. erstellt wird. Da eine FPGA verwendet wird kann die FPGA-EBS-Karte der Hes-so Valais / Wallis Sion benutzen. Dabei handelt es sich um eine Karte, welche von der Hochschule zu Ausbildungszwecken entwickelt wurde. Sie bietet eine FPGA mit einem ROM und verschiedenen Schnittstellen, wie z.B. auch die USB-Schnittstelle die benötigt wird.

Dies ist interessant da so ein grosser Teil der Hardware bereits vorhanden ist: die FPGA und die USB-Schnittstelle.

Durch die Verwendung dieser FPGA-EBS-Karte war das Format der Erweiterungskarte, welche die analoge Schaltung so wie auch die A/D-Wandler (ADC) und den Anschluss des PT1000-Widerstandsthermometers umfasst, bereits vorgegeben. Die Erweiterungskarte wird nach dem an der Hochschule verwendeten Mezzanine-Prinzip aufgebaut. Beim Mezzanine-Prinzip handelt es sich um die Erweiterung der FPGA-EBS-Karte durch das Stapeln von Erweiterungskarten. Die Verbindung wird dabei über die beiden Mezzanine-Stecker (Mezza A & B) hergestellt. Diese Steckverbindung stellt gleichzeitig die mechanische wie auch die elektrische Verbindung her.

Das Prinzip der Erweiterungskarten ermöglicht es dieses Projekt erweiterbar zu gestallten. Pro Erweiterungskarte werden vier Kanäle implementiert. Danach können solche Erweiterungskarten aufeinander gestapelt werden. Diese Erweiterungsmöglichkeit ist in dieser Anwendung auf 4 Mezzanine-Karten begrenzt, da die FPGA-EBS-Karte auch begrenzte Ressourcen aufweist. Weitere Details zu diesem modularen Prinzip und zum Mechanismus, welcher dies ermöglich, sind unter *4 Herstellung*. vorhanden. Dieses System der Erweiterungskarten wurde von der Fachhochschule entwickelt und nennt sich Mezzanine.

Das Model in Abbildung 2-6 soll dieses Prinzip zeigen. Zu sehen ist die FPGA-EBS-Karte, welche die Funktion der Basiskarte übernimmt. Die Mezzanine-Erweiterungskarten so wie die PT1000-Widerstandsthermometer sind auch ersichtlich.

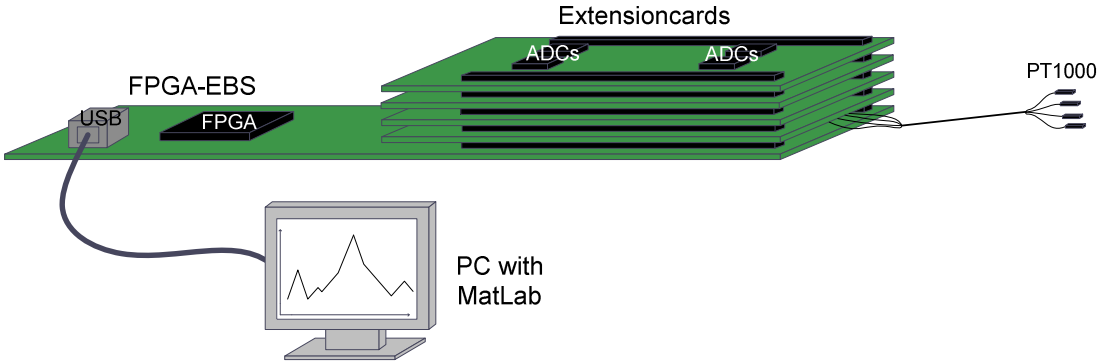


Abbildung 2-6: Hardwaremodel

Weitere Details zur Herstellung der Hardware sind unter *4 Herstellung* zu finden.

3. Entwicklung der Hardware

Dieses Kapitel enthält Erklärungen zur entwickelten Hardware der Erweiterungskarte. Des Weiteren wird der verwendete A/D-Wandler und die erstellte FPGA-Schaltung erklärt.

Für die Erstellung der elektrischen Schemas wurde die Software P-CAD Schematic von Altium in der Version 2006 verwendet.

3.1 Übersicht

Die gesamte Hardware lässt sich mit Hilfe des Blockschemas in Abbildung 3-1 erklären.

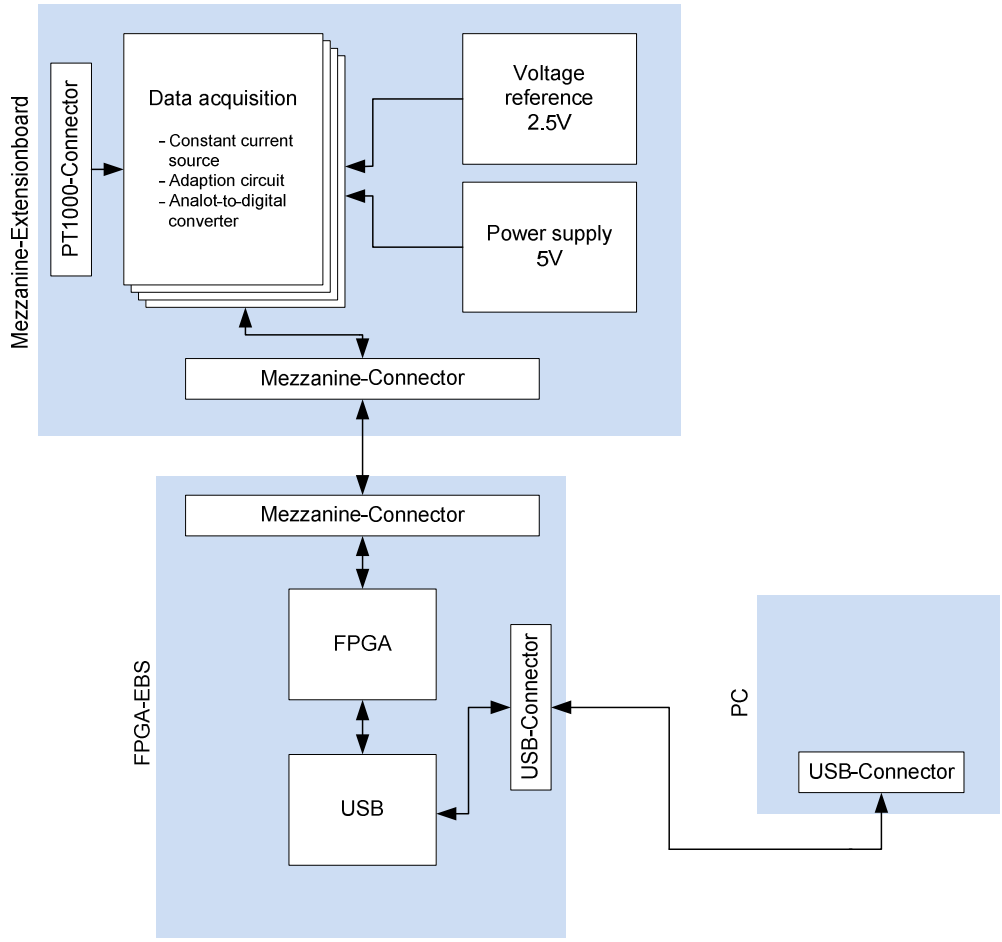


Abbildung 3-1: Übersicht der Hardware

Auf diesem Schema ist ersichtlich, dass die Hardware in drei Teile zerlegt werden kann. Die drei Teile sind die Mezzanine-Erweiterungskarte, die FPGA-EBS-Karte und der PC.

Mezzanine-Erweiterungskarte
Wie aus dem Schema zu entnehmen ist umfasst die Erweiterungskarte die Datenaufnahme, eine Spannungsversorgung, eine Spannungsreferenz und die Anschlüsse welche die Verbindung mit den Sensoren und der FPGA herstellen. Auch ersichtlich ist, dass sich auf einer Erweiterungskarte vier Kanäle befinden.

Die Erweiterungskarte enthält die gesamte analoge Schaltung für jeden Kanal. Zudem befinden sich die A/D-Wandler auch auf diesen. Dies bedeutet, dass sich auf der Erweiterungskarte analoge, wie auch digitale Schaltungsteile befinden. Dies ist insofern wichtig, da beim Erstellen des Schemas und des Designs darauf geachtet werden muss, um Probleme zu vermeiden und um eine maximale Qualität zu erreichen.

Auch ersichtlich ist, dass maximal vier Erweiterungskarten genutzt werden können.

FPGA-EBS
Dies ist eine Karte, welche eine FPGA und die nötige Schaltung beinhaltet. Diese Karte war bereits vorhanden. Es handelt sich dabei um die Karte FPGA-EBS in der Version 1.0 Die Karte beinhaltet nebst der FPGA auch eine Mezzanine-Schnittstelle zur Erweiterungskarte, ein EEPROM und eine USB-Schnittstelle.

PC
Der PC, welcher das Ende der Hardware bildet, verfügt auch über eine USB-Schnittstelle, welche die Konfiguration der Acquisition und den Empfang der Daten ermöglicht.

In den folgenden Kapiteln werden die Einzelnen Teile detailliert beschrieben. Dabei wird allerdings nur erklärt, was im Laufe dieses Projektes realisiert bzw. erstellt wurde.

3.2 Mezzanine-Erweiterungskarte

Beim analogen Teil handelt es sich um den Teil der Datenaufnahme und den ersten Teil der Datenverarbeitung. Diese Hardware wird für jede Erweiterungskarte benötigt.

Das elektrische Schema des der Erweiterungskarte ist im Anhang *14.3.2 Elektrisches Schema der Erweiterungskarte (Version 1.2)* wiedergegeben. Die nun im Einzelnen beschriebenen Teile sind da zu finden. Das Prinzip des analogen Teils der Erweiterungskarte wurde von einem bereits bestehenden Projekt [2] übernommen.

Das hier erklärte Schema entspricht dem Schema der Version 1.2. Im Anhang befindet sich auch das Schema der Version 1.12. Diese wurde auch beigefügt, da die Erweiterungskarte nach diesem hergestellt wurde. Der einzige Unterschied zwischen beiden Versionen besteht beim Spannungsregler. Im entsprechenden Kapitel wird auf den Unterschied hingewiesen. Auf der beigelegten CD unter *pcad* und auf dem Server der Hes-so Valais / Wallis Sion unter *P:\PCB\Students\TSTD\Furrer D\PCB* sind auch beide Versionen des Schemas vorhanden.

3.2.1 Spannungsversorgung

Die Mezzanine-Erweiterungskarte benötigt eine 5V Spannungsversorgung. Diese wird für die Operationsverstärkerschaltungen und den A/D-Wandler benötigt. Mit einer externen Spannungsquelle wird mit Hilfe eines Spannungsreglers die 5V Spannungsversorgung erzeugt. Diese externe Spannungsquelle muss im Bereich zwischen 6V und 16V liegen. Die vom FPGA-EBS zur Verfügung gestellte 5V-Spannungsversorgung wird nicht verwendet, da die Versorgung des analogen von der des digitalen Teils getrennt werden soll.

Die Spannungsversorgung ist mit einem Regulator realisiert der auf niedrige Verlustleistung, tiefes Rauschen und kleines Drop-Out[3] ausgerichtet ist. Es handelt sich dabei um den integrierten Schaltkreis (Integrated Circuit, IC) LP2985 welcher von National Semiconductors hergestellt wird. Dieser bietet die benötigten Eigenschaften bei einer geringen externen Schaltung. Lediglich drei zusätzliche Kondensatoren werden benötigt. Hierbei muss auf die Bedingungen, welche im Datenblatt des Reglers spezifiziert sind geachtet werden. In erster Linie ist es wichtig, dass die Ausgangskapazität C61 einen passenden Ersatzwiderstand (low ESR[4]) aufweist. Dies ist nötig, da der Regulator ansonsten nicht stabil betrieben werden kann. Low ESR bedeutet in diesem Fall, dass der Ersatzwiderstand zwischen 0.005Ω und 0.5Ω liegt. Daher wird ein Kondensator mit einem ESR von 0.125Ω verbaut. Der serielle Ersatzwiderstand in Funktion des Ausgangsstromes und der stabile Bereich werden in der

[3] Drop-Out: Als Drop-Out wird der Spannungsunterschied zwischen Eingang und Ausgang bezeichnet.

[4] Low ESR: Low equivalent series resistance (zu Deutsch: geringer gleichwertiger serieller Widerstand)

Abbildung 3-2 dargestellt. Diese Abbildung stammt aus dem Datenblatt des LP2985 von welchem sich ein Auszug im Anhang *14.4.1 LP2985* befindet.

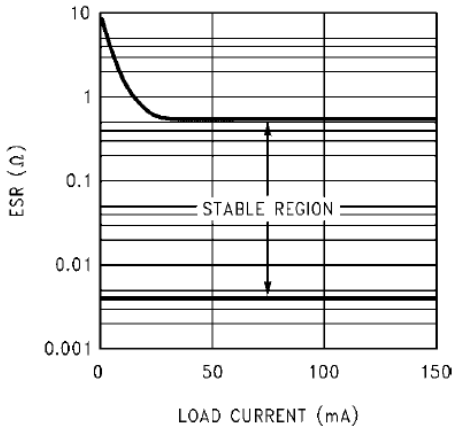


Abbildung 3-2: Ersatzwiderstand ESR in Funktion des Ausgangsstromes I_{Load}: ESR = $f(I_{Load})$

Die Spannungsversorgung liefert einen maximalen Strom von 150mA. Die analoge Schaltung mit den A/D-Wandlern benötigt allerdings maximal 30mA.

Die Berechnung des Verbrauchs der Erweiterungskarte befindet sich im *14.3.5 Berechnung des Stromverbrauch* der Erweiterungskarte.

Die Abbildung 3-3 zeigt das Schema der 5V-Spannungsversorgung.

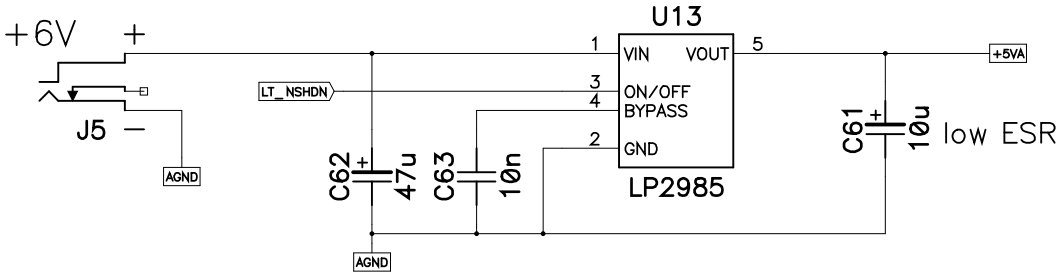


Abbildung 3-3: Schema der Spannungsversorgung

Der Kondensator C62 dämpft das Eingangssignal für höhere Frequenzen, damit der Spannungsregler stabil betrieben werden kann. Der Bypass-Kondensator[5] C63 ist da um das Rauschen am Ausgang des Spannungsreglers zu reduzieren.

Bei C63 und C61 wird mit einem Keramikkondensator bzw. einem low ESR Kondensator bereits ein passendes Bauteil verwendet. Um die Leistung noch zu verbessern müsste am Eingang anstelle eines Tantalkondensators einen Keramikkondensator verwendet werden.

Es wird eine Spannungsversorgung pro Erweiterungskarte benötigt.

Das Schema der Version 1.12 enthält nicht denselben Spannungsregler. Dies erfordert eine kleine Anpassung der Beschaltung. Was genau gemacht werden muss wird unter *6.2.1 Spannungsversorgung und Spannungsreferenz* erklärt.

3.2.2 Spannungsreferenz

Die Spannungsreferenz wird für verschiedene Anwendungen benötigt. Hierzu gehören die Erstellung des konstanten Stromes, die Messsignalanpassung und die A/D-Wandlung. Die Spannungsreferenz dient als mathematischer Fixpunkt für die Signalverarbeitung.

[5] Bypass-Kondensator: (bypass: ableiten oder umleiten) Kondensator der unerwünschte Signale von einem Pin eines ICs ableitet.

Demzufolge ist eine Referenz für die Messung wie für die A/D-Wandlung vorhande, welche sehr genau und überall verfügbar ist.

Um die Spannungsreferenz von 2.5V zu generieren wird ein Schaltkreis, welcher von Maxim hergestellt wird, verwendet. Das Model MAX6126 wird verwendet. Dabei handelt es sich um eine ultra-hoch-präzise, rauscharme Spannungsreferenz. Diese zeichnet sich durch einen kleinen Temperaturkoeffizienten, durch eine hohe Genauigkeit und schwaches Rauschen aus.

Die Abbildung 3-4 zeigt das Schema der Spannungsreferenz

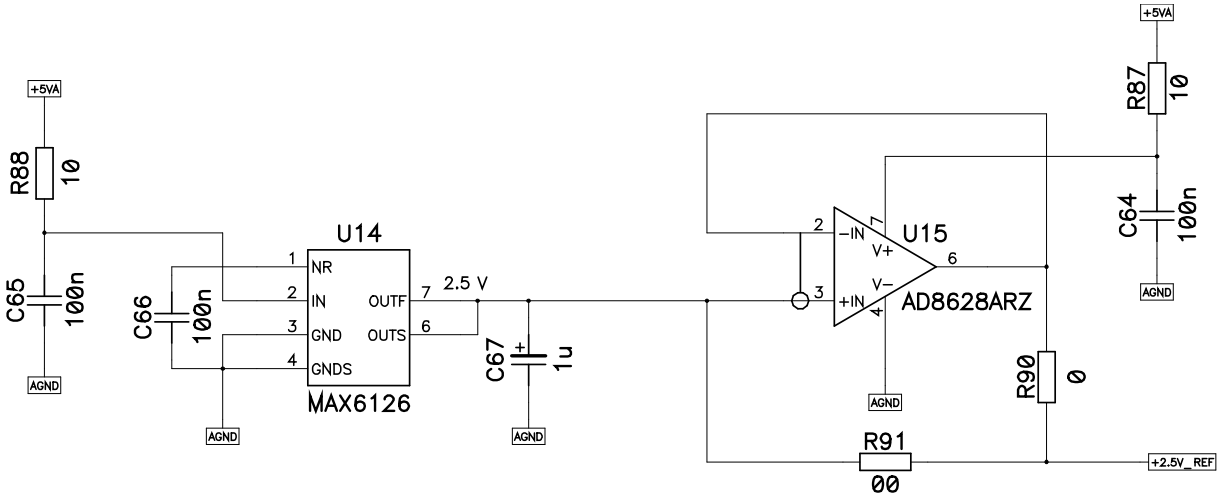


Abbildung 3-4: Schema der Referenzspannung

Im linken Teil ist die Spannungsreferenz MAX6126 ersichtlich. Zu sehen ist C66, welcher verbaut wird um das Rauschen zu reduzieren.

Des Weiteren ist am IN-Pin ein RC-Tiefpass-Filter zu sehen, welcher vorhanden ist um das Eingangssignal, aus welchem die Referenzspannung erstellt wird, zu Filtern. Es handelt sich dabei um einen RC-Filter erster Ordnung mit der folgenden Grenzfrequenz.

$$f_C = \frac{1}{2 \cdot \pi \cdot R \cdot C} = \frac{1}{2 \cdot \pi \cdot 10\Omega \cdot 100nF} = 159.15kHz$$

Dies bedeutet nun, dass alle Signal die über dieser Grenzfrequenz liegen herausgefiltert werden, was wiederum zu einem besseren Rauschverhalten führt.

Nach der Spannungsreferenz folgt ein Impedanzwandler. Diese Schaltung wird verwendet, um die Spannungsreferenz unabhängiger gegenüber der Last zu machen und so die Spannung von 2.5V zu garantieren. Auch hier ist ein RC-Tiefpass-Filter an der Spannungsversorgung ersichtlich.

Beim Operationsverstärker des Impedanzwandlers handelt es sich um einen Operationsverstärker welcher einen sehr kleinen Drift[6] wie auch ein sehr geringes Offset aufweist. Der Operationsverstärker hebt auch das Offset automatisch auf. Es ist das Model AD8628 von Analog Devices.

Des Weiteren sind die Widerstände R90 und R91 zu sehen. Diese werden zu Testzwecken eingesetzt. Der Wert 00 des Widerstandes R91 bedeutet, dass dieser unbestückt ist. Allerdings kann an dieser Stelle ein Widerstand eingesetzt werden, um den Einfluss bzw. die Funktionalität des Impedanzwandlers zu testen.

Für eine Analyse des Rauschens fehlen die nötigen Daten. D.h. die Rauschspektren, welche vom Hersteller angegeben werden gehen nicht bis in den Frequenzbereich in welchem sich das zu messende Signal befindet.

[6] Drift: Dies bezeichnet die Veränderung der Eigenschaften in Funktion der Zeit oder der Temperatur

Es wird eine Spannungsreferenz pro Erweiterungskarte benötigt.

3.2.3 Messschaltung

Dies ist die Schaltung, des analogen Teils, welche die Messung durchführt. Diese wird für jeden einzelnen Kanal benötigt. Die Signale usw. sind stets mit "_1" gekennzeichnet. Dies zeigt den Kanal auf, zu welchen das Signal gehört. Es wird lediglich ein Kanal erklärt, da alle Kanäle identisch sind.

3.2.3.1 Konstantstromquelle

Um die Temperaturmessung, welche in Form eines temperaturabhängigen Widerstandes (R_T) realisiert wird, bewerkstelligen zu können wird ein konstanter Strom benötigt. Dieser wird benötigt, damit die Messgrösse in ein Signal umgewandelt werden kann, welches danach verarbeitet wird. Die neue Messgrösse wird nach dem Ohmschen Gesetz eine Spannung sein.

Der konstante Strom wird so gewählt, dass die Selbsterwärmung des PT1000-Widerstandsthermometers nicht zu stark wird. Zudem muss darauf geachtet werden, dass die erzeugte Spannung nicht zu tief ist, damit die Anfälligkeit gegenüber Störungen gering ist. So wird ein Strom I_{Mes} von ca. 1mA gewählt.

Bei der Schaltung, welche gewählt wurde, handelt es sich um einen differentiellen Spannung/Strom-Wandler. Die Abbildung 3-5 zeigt diese.

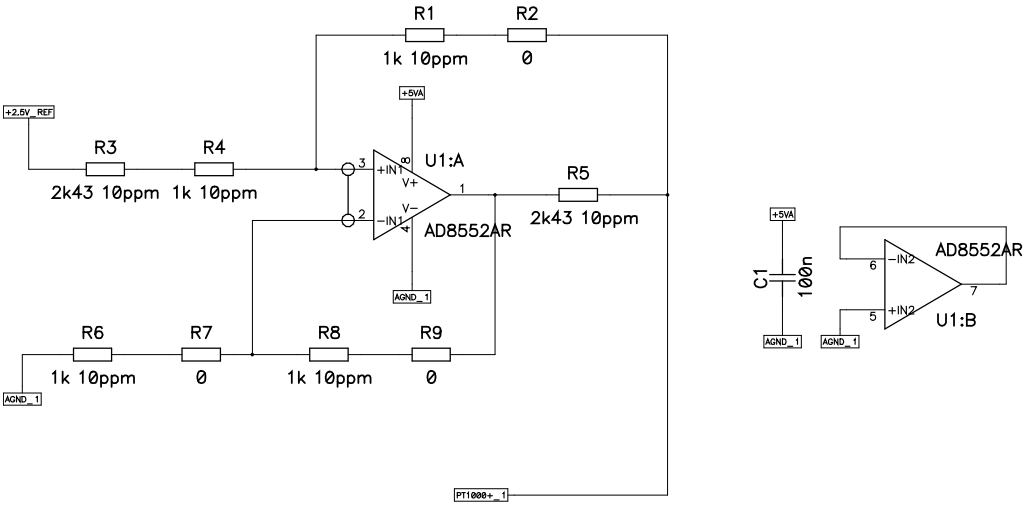


Abbildung 3-5: Konstantstromquelle

Es handelt sich dabei um eine Stromquelle, welche mit Hilfe eines Operationsverstärkers erstellt wird. Beim Operationsverstärker handelt es sich, wie auch beim Impedanzwandler bei der Spannungsreferenz, um ein Model mit geringem Drift und Offset. Dies ist wichtig, damit sich der Strom möglichst wenig in Funktion der Zeit und der Temperatur verändert. Hier wird das Model AD8552 von Analog Devices verwendet. Des Weiteren bietet dieser Operationsverstärker noch die Eigenschaft, dass das Offset automatisch aufgehoben wird.

Dies ist ein IC, welcher zwei Operationsverstärker enthält. Wie zu sehen ist wird der zweite Operationsverstärker nicht verwendet, um einen Einfluss auf die Stromquelle zu verhindern. Hierzu wird der eine Eingang (+IN2) auf Masse gelegt und der Ausgang wird zum anderen Eingang (-IN2) rückgekoppelt.

Wie die Stromquelle dimensioniert wurde ist im *14.3.3 Berechnung* der Stromquelle ersichtlich. Hier soll lediglich das Resultat dieser Schaltung aufgezeigt werden. Dabei

handelt es sich um eine Gleichung, welcher den erzeugten Strom in Funktion der Referenzspannung und des Shuntwiderstandes[7] R_3 und R_5 beschreibt.

$$I_{Mes} = \frac{V_{REF}}{R_5}$$

Um den Mittelpunkt der mit dem PT1000-Widerstandsthermometer erstellten Spannung knapp unter $\frac{V_{REF}}{2}$ zu fixieren beträgt der Strom nicht genau 1mA sondern 1.029mA. Warum dies gemacht wird, wird im nächsten Kapitel erklärt.

$$I_{Mes} = \frac{2.5V}{2.43k\Omega} = 1.029mA$$

Zu sehen ist auch die Verbindung PT1000+_1, mit welcher der Strom zum PT1000-Widerstandsthermometer geführt wird.

Die Widerstände mit dem Wert 0Ω sind da, falls in der fertigen Schaltung Widerstandswerte korrigiert oder angepasst werden sollen.

Für eine Analyse des Rauschens fehlen die nötigen Daten. D.h. die Rauschspektren, welche vom Hersteller angegeben werden gehen nicht bis in den Frequenzbereich in welchem sich das zu messende Signal befindet.

Des Weiteren ermöglicht die Stromquelle die Korrektur der nicht linearen Stromquelle und hat geringe Verluste. Mehr dazu ist im Anhang unter *14.3.3 Berechnung der Stromquelle* zu finden.

Die Stromquelle wurde mit Hilfe von [7] erstellt.

3.2.3.2 PT1000-Widerstandsthermometer

Der unter *2.1.3 Temperaturmessung* erklärte Widerstandsbereich beträgt 800Ω < R_T < 1600Ω. Mit dem vorhin festgelegten Strom erhält ergibt dies eine Spannung U_{Mes1} von 0.823V < U_{Mes1} < 1.646V.

Der Mittelpunkt des Bereiches beträgt mit einem Widerstand R_T von 1200Ω somit 1.2346V. Dies liegt gerade unter Hälfte der Referenzspannung und ermöglicht eine vereinfachte Weiterverarbeitung des gemessenen Signals. Warum nicht genau die Hälfte der Referenzspannung wird im nächsten Kapitel erklärt.

Die Abbildung 3-6 zeigt den Anschluss des PT1000-Widerstandsthermometers und die Filterung am Eingang der Schaltung. Diese Filterkondensatoren (C6 und C10) werden benötigt um das gemessene Signal zu glätten und von Störungen zu befreien.

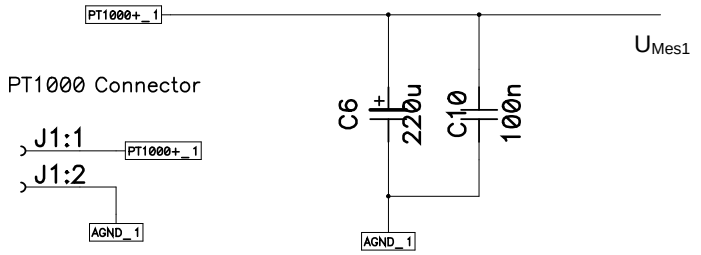


Abbildung 3-6: Anschluss und Beschaltung des PT1000-Widerstandsthermometers

Die Verbindung die nach rechts geht ist der Anschluss an die Schaltung welche das Signal anpasst.

[7] Shuntwiderstand: Dies bezeichnet den Widerstand, welcher für die Strommessung in der Stromquelle verwendet wird

3.2.3.3 Signalanpassung

Das Signal, welches von dem PT1000-Widerstandsthermometer her kommt, muss nun noch auf den gewünschten Bereich angepasst werden.

Bei diesem Bereich handelt es sich um einen Spannungsbereich von $\pm\frac{V_{REF}}{2}$. Weitere Erklärungen hierzu befinden sich im nächsten Kapitel.

Um das Signal auf diesen Bereich zu bringen genügt es nicht, einen normalen Verstärker mit einem Operationsverstärker zu verwenden. Aus diesem Grund wird ein differentielles Verstärkerpaar verwendet.

Das folgende Schema zeigt die Verstärkerschaltung.

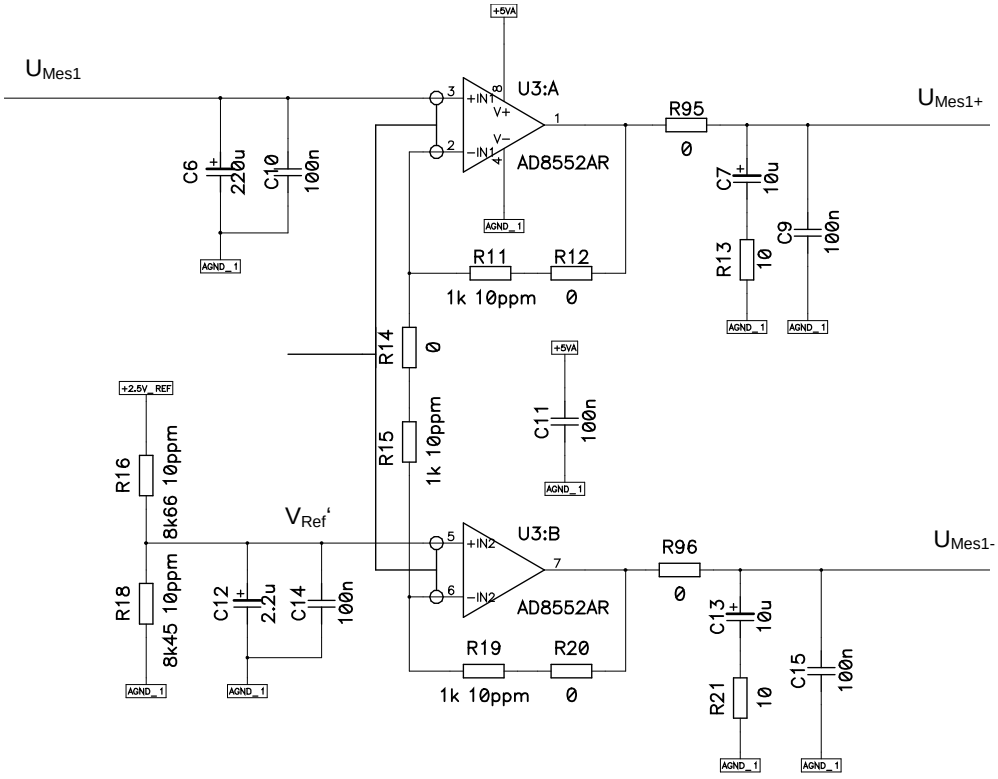


Abbildung 3-7: Schema der Verstärkerschaltung der Signalanpassung

Wie bereits erwähnt handelt es sich um ein differentielles Verstärkerpaar. Es sind also zwei Differenzverstärker vorhanden, welche Spannungen erzeugen, deren Differenz dem gewünschten Eingangswertebereich entspricht.

Die Spannung welche nebst U_{Mes1} für diese Operation auch noch benötigt wird entspricht dem vorher definierten Mittelpunkt des Bereichs der gemessenen Spannung und wird mit dem Spannungsteiler am Eingang des unteren Operationsverstärkers erstellt. Diese Spannung wird mit V_{REF}‘ bezeichnet und beträgt 1.2346V.

Der obere Operationsverstärker berechnet also die gewichtete Differenz zwischen U_{Mes1} und V_{REF}‘. Dies lässt sich vereinfacht mit der folgenden Formel ausdrücken.

$U_{Mes1+} = 2 \cdot U_{Mes1} - V_{Ref}{}'$ wobei $Vref = V_{Ref} \cdot \frac{R_{18}}{R_{16}+R_{18}}$

Diese Spannung wird über die rechte obere Verbindung zum positiven Eingang des A/D-Wandlers geführt.

Der untere Operationsverstärker berechnet dementsprechend dieselbe gewichtete Differenz, wobei andere Vorzeichen vorhanden sind. Die folgende vereinfachte Formel beschreibt diesen Zusammenhang.

$$U_{Mes1-} = 2 \cdot V_{Ref}{}' - U_{Mes1}$$

Diese Spannung wird über die rechte untere Verbindung zum negativen Eingang des A/D-Wandlers geführt.

Da der A/D-Wandler die Differenz der bedien Eingänge erstellt ergibt sich die folgende Formel für die effektiv umgewandelte Spannung.

$$U_{Mes2} = U_{Mes1+} - U_{Mes1-} = 3 \cdot (U_{Mes1} - V'_{REF})$$

Die genau Entwicklung und Dimensionierung dieser Stufe befindet sich im Anhang *14.3.4 Berechnung* der Verstärkerstufe.

Nach dieser Stufe ist somit einen neuen Wertebereich für das gemessene und umgewandelte Signal vorhanden. Dieser beträgt -1.212V < U_{Mes2} < 1.165V.

Da dieses Signal das Ende des analogen Teiles darstellt soll die Kennlinie, welche am Eingang des A/D-Wandlers vorhanden ist noch kurz aufgezeigt werden. Abbildung 3-8 zeigt diese Kennlinie.

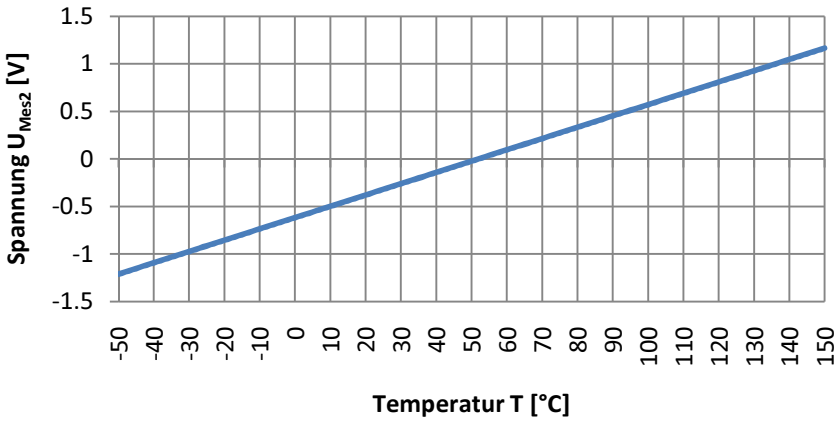


Abbildung 3-8: Kennlinie der Umgewandelten Spannung ($U_{Mes2}=f(T)$)

Auf dem Schema in Abbildung 3-7 sind auch die Glättungskondensatoren am Eingang (C6, C10, C12 und C14) der beiden Operationsverstärker ersichtlich. Diese entfernen Störungen aus den Signalen und glätten das Signal.

Zudem ist am Ausgang der Verstärker jeweils eine Filterung vorhanden. Die 10µF-Kondensatoren (C7 und C13) filtern die eher niederfrequenten Störungen ab und die 100nF-Kondensatoren (C9 und C15) filtern die eher hochfrequenten Störungen ab. Die Widerstände mit dem Wert von 10Ω sind vorhanden, damit die Ausgänge der Operationsverstärker nicht kapazitiv überlastet werden.

Der Kondensator C11 dient zur Entstörung der Spannungsversorgung. Die 0Ω-Widerstände geben die Möglichkeit eventuelle Wertkorrekturen oder Anpassungen vorzunehmen.

Bei den Operationsverstärkern handelt es sich um dieselben, welche auch bei der Stromquelle verwendet werden.

Für eine Analyse des Rauschens fehlen die nötigen Daten. D.h. die Rauschspektren, welche vom Hersteller angegeben werden gehen nicht bis in den Frequenzbereich in welchem sich das zu messende Signal befindet.

3.3 Analog-Digital-Wandler

In diesem Kapitel soll dieser A/D-Wandler und seine Funktion innerhalb der Messkette erklärt werden. Der A/D-Wandler gehört auch zur Erweiterungskarte, erhält aber wegen seiner Wichtigkeit ein eigenes Kapitel

Da eine möglichst hohe Genauigkeit erreicht werden soll werden die wohl momentan leistungsstärksten A/D-Wandler verwendet, welche auf dem Markt zu finden sind. Es handelt sich dabei um das Model ADS1281 von Texas Instruments. Dies ist ein Delta-Sigma-Wandler (ΔΣ-Wandler).

3.3.1 Allgemeines

Hier soll der ΔΣ-Wandler an und für sich erklärt werden.

3.3.1.1 Funktionsprinzip

Das Funktionsprinzip des ΔΣ-Wandler soll anhand des Blockschemas in Abbildung 3-9, welches aus dem Datenblatt des ADS1281 von Texas Instrument [1] stammt, erklärt werden.

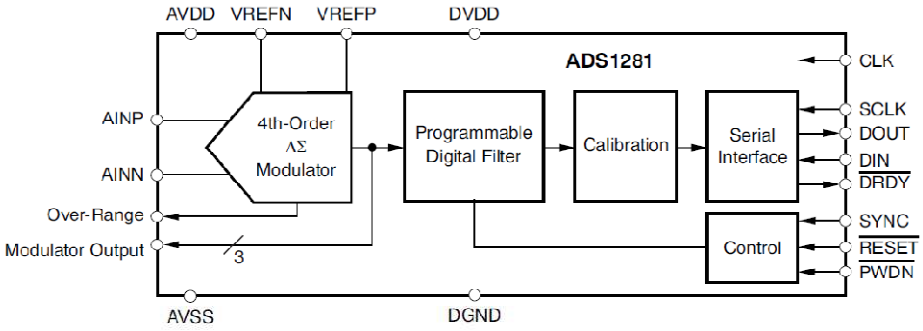


Abbildung 3-9: Blockschaltbild des ADS1281 [1]

Der ΔΣ-Wandler beinhaltet einen Delta-Sigma-Modulator (ΔΣ-Modulator) 4. Ordnung. Dieser quantisiert die Daten. Der enthaltene konfigurierbare Filter filtert die Daten. Ein Kalibrierungs-Block ermöglicht die Offset- und Verstärkungskalibrierung um Offset- oder Verstärkungsfehler zu korrigieren. Die schlussendlichen Daten werden über die serielle Schnittstelle versendet. Ein Kontroll-Block ermöglicht die Steuerung des A/D-Wandlers.

Ersichtlich sind auch die Anschlüsse der Referenzspannung und der zu messenden Spannung. Und auch die restlichen Anschlüsse, wie z.B. die SPI-Schnittstelle, sind zu sehen.

3.3.1.2 Eigenschaften

Der Analog-Digital-Wandler wandelt das erhaltene Signal (U_{Mes2}) in eine binäre Zahl um.

Mit der vorhandenen Referenzspannung ergibt sich ein Eingangswertebereich von ±1.25V.

Beim A/D-Wandler ADS1281handelt es sich um einen 32Bit-ΔΣ -Wandler. Allerdings werden nur die 24 werthöchsten Bits verwendet. Warum dies so gemacht wird, wird im nachfolgenden Kapitel *3.3.1.3 Delta-Sigma-Modulator* erklärt.

Da nur mit 24Bit gearbeitet wird ergibt dies nach der Konvertierung einen digitalen Wertebereich von -2^{23} bis $2^{23}-1$. Diese digitalen Werte werden mit U_{MesD} bezeichnet und sind im Zweierkomplement kodiert. Die erhaltenen Werte werden dann über das vorhandene SPI-Interface weitergesendet.

Die Tabelle 3-1 soll die Werte, welche durch den gegebenen Eingangswertebereich gegeben sind, aufzeigen.

Eingangssignal U_{Mes2} [V]	**digitaler 24bit-Wert U_{MesD} [ ]**
$\geq \frac{V_{REF}}{2}$	$7FFFFF_h$
$0 < U_{Mes2} < \frac{V_{REF}}{2}$	$000000_h < U_{MesD} < 7FFFFF_h$
0	000000_h
$\frac{-V_{REF}}{2} < U_{Mes2} < 0$	$800000_h < U_{MesD} < 000000_h$
$\leq \frac{-V_{REF}}{2}$	800000_h

Tabelle 3-1: Digitale 32bit-Werte im Bezug auf das Eingangssignal

Anhand der Tabelle lässt sich erkennen, dass die Zahlen, wie bereits erwähnt, im Zweierkomplement kodiert sind. Zudem ist ersichtlich, was für Werte bei einer Bereichsüberschreitung vorhanden sind.

Der ΔΣ-Wandler zeichnet sich aus durch hervorragende Rauscheigenschaften. Bei einer tiefen Datenrate (250SPS) beträgt der Störabstand SNR_{dB} 130dB. Demzufolge wird diese Datenrate gewählt.

Bei den enthaltenen digitalen Filtern handelt es sich um einen Sinc-Filter, einen FIR und einen IIR-Filter, wobei durch die Konfiguration bestimmt werden kann welche der Stufen oder ob überhaupt eine Stufe verwendet werden soll.

Der ΔΣ-Wandler ist, wie bereits erwähnt, in gewissen Bereichen konfigurierbar.

3.3.1.3 Delta-Sigma-Modulator

Es handelt sich dabei um einen ΔΣ-Modulator 4. Ordnung. Die Abbildung 3-9 zeigt den Aufbau dieses ΔΣ-Modulators.

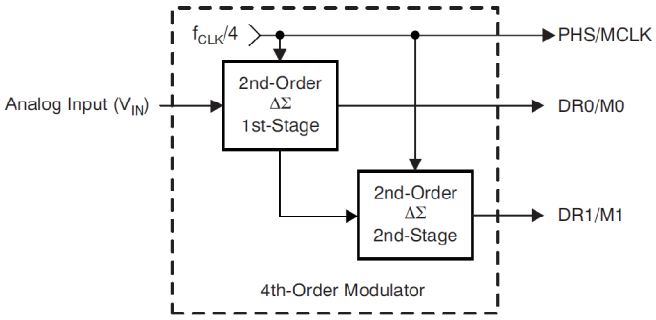


Abbildung 3-10: Aufbau des ΔΣ -Modulators 4. Ordnung [1]

Zu sehen ist, dass es sich bei dem Modulator 4. Ordnung um die Kaskadierung zweier Modulatoren 2. Ordnung handelt. Hierbei nimmt die erste Stufe die Konvertierung des analogen Signals in ein PCM-Signal vor. Die zweite Stufe hat in erster Linie die Aufgabe das Quantisierungsrauschen der ersten Stufe aufzuheben. Der an M0 und M1 resultierende Bitstrom wird dann kombiniert und zu den Filtern übertragen.

Ein Modulator dieser Ordnung ermöglicht eine sehr hohe Auflösung da wenig Quantisierungsrauschen vorhanden ist. Zudem können Resttöne, welche durch eine nicht ideale Konvertierung entstehen, bei Modulatoren dieser Ordnung vernachlässigt werden.

Das Quantisierungsrauschen soll folgend für die ganzen 32 Bits bestimmt werden.

Vorweg soll die Grösse einer Quantisierungsstufe bestimmt werden.

$$U_{\Delta} = \frac{U_{Mes2}}{2^n} = \frac{2.5V}{2^{32}} = 582.1 \cdot 10^{-12}V$$

Der Effektivwert des Quantisierungsfehlers beträgt dann

$$e_{rms}^2 = \frac{{U_{\Delta}}^2}{12} = \frac{(582.1 \cdot 10^{-12}V)^2}{12} = 28.23 \cdot 10^{-21}V^2$$

Das Spektrum des Quantisierungsfehlers nach der Wandlung erstreckt sich von 0Hz bis 512kHz, was $f_s/2$ entspricht.

Wie zu sehen ist, ist der Quantisierungsfehler sehr gering und kann vernachlässigt werden.

Falls der ΔΣ-Wandler nun ideal ist beträgt der Störabstand

$$SNR_{ideal} = 20 \cdot \log\left(\frac{2^n \cdot \sqrt{12}}{2 \cdot \sqrt{2}}\right) = 20 \cdot \log\left(\frac{2^{32} \cdot \sqrt{12}}{2 \cdot \sqrt{2}}\right) = 194.42dB$$

Da aber im Analogteil der Schaltung auch ein Rauschen vorhanden ist beträgt der Störabstand lediglich 130dB, wie dies im Datenblatt im Anhang *14.4.2 ADS1281* auch angegeben wird.

Anhand der oben verwendeten Formel für den Störabstand lässt sich die Anzahl stabiler Bits bestimmen.

$$n = \frac{log\left(\frac{10^{\frac{SNR_{real}}{20}}}{\sqrt{12}}\right)}{\log(2)} = \frac{log\left(\frac{10^{\frac{130dB}{20}}}{\sqrt{12}}\right)}{\log(2)} = 19.8 \cong 19$$

Es wird abgerundet, da 19Bits stabil sind.

Wie weiter oben zu sehen war ist die Quantisierungsstufe mit $582.1 \cdot 10^{-12}$V sehr klein. Falls für die Eingangskapazität der analogen Eingänge des A/D-Wandlers ein Richtwert von 1pF angenommen wird beträgt die elektrische Ladung die einer Quantisierungsstufe entspricht

$$Q = C \cdot U_{\Delta} = 1pF \cdot 582.1 \cdot 10^{-12}V = 582.1 \cdot 10^{-24}C$$

Die elektrische Ladung eines Elektrons beträgt $1.6022 \cdot 10^{-19}$C.

Die Ladung welche am Eingang des A/D-Wandlers erkennt werden sollte beträgt also weniger als die Ladung eines Elektrons. Dies macht insofern keinen Sinn, da sich zumindest ein Elektron bewegen muss, damit eine potentielle Veränderung erkannt werden kann.

Die Ladung eines Elektrons entspricht ungefähr der Ladung von 275 Quantisierungsstufen, wodurch ersichtlich wird, dass die wertniedrigsten 8Bits keine Information beinhalten. Dies wird durch die weiter oben vorgenommene Berechnung der Anzahl stabilen Bits bestätigt.

Es werden also nur die 24 werthöchsten Bits verwendet. Dies hat eine neue Grösse der Quantisierungsstufe zur Folge.

$$U_{\Delta 24Bit} = \frac{U_{Mes2}}{2^n} = \frac{2.5V}{2^{24}} = 149.01 \cdot 10^{-9}V$$

Der Effektivwert des Quantisierungsfehlers beträgt dann

$$e_{rms}^2 = \frac{{U_{\Delta 24}}^2}{12} = \frac{(149.01 \cdot 10^{-9}V)^2}{12} = 1.85 \cdot 10^{-15}V^2$$

Der Quantisierungsfehler ist immer noch so gering, dass er vernachlässigt werden kann.

Noch erwähnt werden sollte die vom Modulator gelieferte Datenrate, welche $f_{data} = f_{clk} / 4$ entspricht. Die Datenrate f_{Data} entspricht der Abtastrate f_s.

3.3.1.4 Digitale Filter

Übersicht

Im A/D-Wandler sind drei digitale Filter vorhanden. Dies sind ein Sinc, ein FIR-Tiefpass-Filter und ein IIR-Hochpass-Filter. Das Blockschema in Abbildung 3-11 erklärt die Filterstufe des ΔΣ–Wandlers.

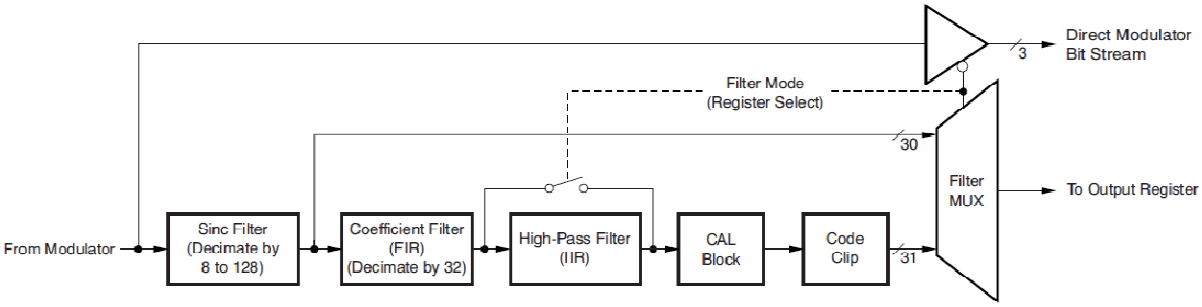


Abbildung 3-11: Filterstufe im ΔΣ –Wandler [1]

Wie zu sehen ist, ist diese Stufe komplett konfigurierbar. Welche Filter verwendet werden kann gewählt werden. Bei dieser Anwendung, der Temperaturmessung, werden nur die ersten zwei Stufen, der Sinc-Filter und der FIR-Tiefpass-Filter verwendet, da ein Hochpass-Filter wenig Sinn machen würde, da sich Temperatursignale auch mit sehr tiefen Frequenzen ändern können.

Der CAL Block ermöglicht eine Kalibrierung der Filterstufe. Der Code Clip Block schneidet den Wert auf einen 32Bit-Wert zu. Falls erwünscht schneidet dieser auch die wertniedrigsten 8 Bits ab, was in Erwägung gezogen werden kann da diese, wie vorhin erklärt, instabil sind.

Sinc-Filter

Der Sinc-Filter hebt idealerweise alle Frequenzanteile ausserhalb einer gegebenen Bandbreite auf. Dies ist in der Realität nicht möglich. Allerdings kommt der vorhandene Sinc-Filter 5. Ordnung nahe an diese Vorgabe heran. Die Abbildung 3-12 zeigt die Frequenzantwort des Filters.

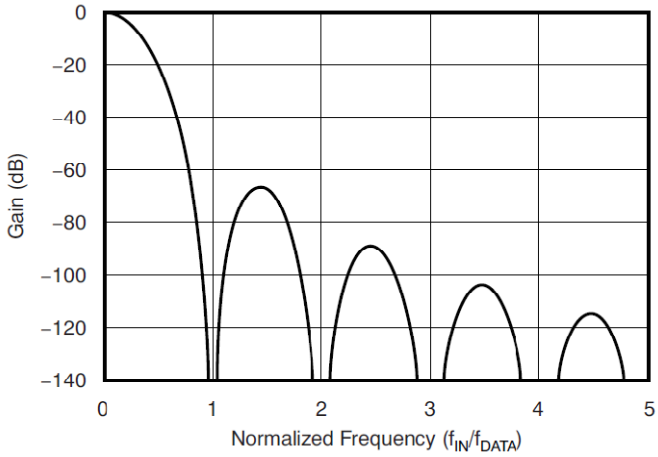


Abbildung 3-12: Frequenzantwort des Sinc-Filters [1]

Wie zu sehen ist sind Nullstellen bei der gewählten Datenrate (250SPS) und den ganzen Vielfachen dieser vorhanden. Das Netzsignal mit 50Hz, welches auch zu Fehlern führen kann, wird bereits um ca. 5dB abgeschwächt.

Mit der Standardeinstellung wird eine Dezimation mit dem Faktor 32 durchgeführt. Dies wird so beibehalten. Am Ausgang des Sinc-Filters ist in der Standardkonfiguration eine Datenrate von $f_{data} = (f_{clk} / 4) / 32 = f_{clk} / 128$ vorhanden.

FIR-Filter

Bei der FIR-Filter-Stufe handelt es sich auch um einen Tiefpass-Filter. Dieser ergänzt den Sinc-Filter und führt eine Dezimation um den Faktor 32 durch. Die Frequenzantwort des Filters ist in Abbildung 3-13 dargestellt.

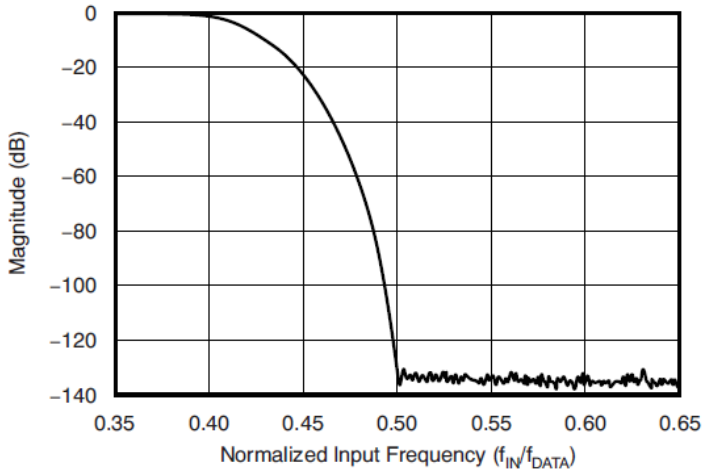


Abbildung 3-13: Frequenzantwort des FIR-Filters [1]

Die Signale mit einer Frequenz die grösser als die Hälfte der Datenrate ist werden um 130dB abgeschwächt. Auch ersichtlich ist die Tatsache, dass bereits Signale mit einer Frequenz die kleiner ist als ein Drittel der Datenrate nicht mehr abgeschwächt werden. Diese werden jedoch teilweise vom Sinc-Filter abgeschwächt. Zu erwähnen ist auch noch, dass sich die Frequenzantwort des Durchlassbereiches bei Vielfachen der Modulatorfrequenz wiederholt. Falls das Signal nun genau solche Frequenzen beinhalten sollte könnten diese zurück in den Bereich des Nutzsignals fallen. Da im analogen Teil der Schaltung genug Filter vorhanden sind gibt es hier keinen Grund Vorkehrungen zu treffen.

Die FIR-Filter-Stufe nimmt in der Standardkonfiguration wiederum eine Dezimation mit dem Faktor 32 vor. Die Schlussendliche Datenrate beträgt dann $f_{data} = (f_{clk} / 128) / 32 = f_{clk} / 4096$.

Datenrate

Wie soeben zu sehen war, beträgt die Datenrate in der Standardkonfiguration $f_{data} = f_{clk} / 4096$. Standardkonfiguration bedeutet unter anderem, dass mit dem nominellen Takt von $f_{clk} = 4096kHz$ gearbeitet wird. In dieser Standardkonfiguration beträgt die Datenrate 1000SPS. Da allerdings eine Datenrate von 250SPS gewählt wurde, aber die Standardkonfiguration ansonsten passend ist, wird der Takt auf $f_{clk} = 1024kHz$ festgelegt. So ergibt sich die gewünschte Datenrate von 250SPS und zudem kann so der Energieverbrauch des A/D-Wandlers noch gesenkt werden.

3.3.1.5 Kalibrierung

Ein Offset kann eingestellt werden. Dies ist nützlich falls am analogen Eingang ein unerwünschtes Offset vorhanden ist oder falls absichtlich mit einem Offset gearbeitet wird um Resttöne zu verhindern.

Der Offset-Kalibrierungswert wird in ein Register geschrieben und ist im Zweierkomplement kodiert. Der Standardwert der Offsetkalibrierung beträgt 0.

Mit der Vollausschlags-Kalibrierung kann ein möglicher Verstärkungsfehler korrigiert werden. Bei dieser Kalibrierung gibt es vier mögliche Werte, welche in ein Register geschrieben werden können. Diese entsprechen den Verstärkungskorrekturen 0, 0.5, 1.0 oder 2.0. Das bedeutet, die umgewandelten Werte können mit diesen Faktoren multipliziert werden bevor sie an den Ausgang gelangen. Standardmässig ist der Wert 1.0 eingestellt welcher keiner Veränderung entspricht.

3.3.1.6 Serielle Schnittstelle

Bei der seriellen Schnittstelle handelt es sich um eine übliche SPI-Schnittstelle. Dies bedeutet, vom ADS1281 aus gesehen gibt es die Verbindungen DIN, DOUT und SCLK. Hierbei wird die Übertragung mit dem seriellen Takt SCLK synchronisiert. Der Eingang DIN dient zum schreiben der Register. Der Ausgang DOUT wird zum Senden der Konvertierungsdaten wie auch zum Lesen der Register verwendet.

Die SPI-Schnittstelle benötigt zudem über eine digitale 3.3V-Spannungsversorgung.

Die Spezifikationen der SPI-Schnittstelle sind im Auszug des Datenblatt des ADS1281 im Anhang *14.4.2 ADS1281*.

3.3.2 Beschaltung

Das Schema in Abbildung 3-14 zeigt den A/D-Wandler und dessen Beschaltung.

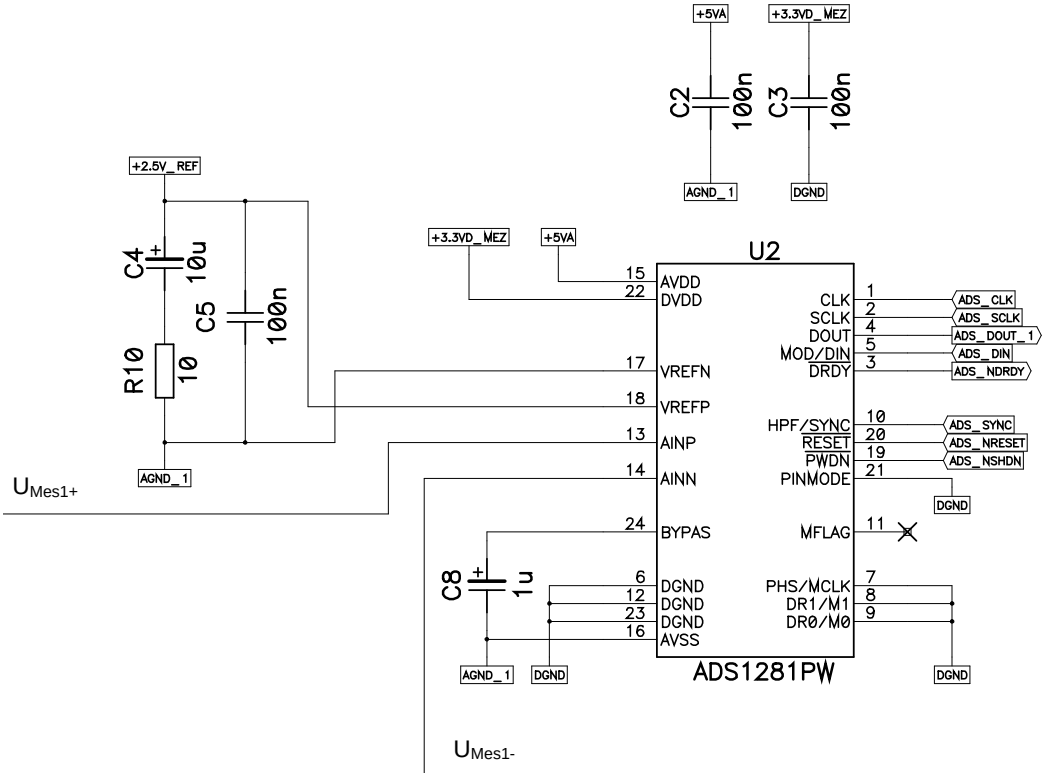


Abbildung 3-14: Schema des A/D-Wandlers ADS1281

Zu sehen sind, nebst dem ΔΣ-Wandler, die zwei Eingangsleitungen U_{Mes1+} und U_{Mes1-}, welche das zu wandelnde Signal übertragen. Wie bereits erwähnt, wird im ΔΣ-Wandler dann die Differenz zwischen diesen beiden Signalen gebildet, wodurch U_{Mes2} gebildet wird.

Des Weiteren ist eine Filterung, wie sie bereits im Schema der Signalanpassung im vorherigen Kapitel verwendet wurde, vorhanden. Auch zwei Glättungskondensatoren sind an den beiden Speispannungen vorhanden. Es werden zwei Einspeisungen benötigt. Die eine +5VA ist für den analogen Teil und die andere +3.3VD für den digitalen Teil. "_MEZ" bedeutet, dass diese von der Mezzanine-Schnittstelle her kommt. Entsprechend sind auch zwei verschiedene Massen, eine analoge und eine digitale, vorhanden.

Auch ersichtlich ist der Bypass-Kondensator C8 mit einem Wert von 1µF.

Auf der rechten Seite des ΔΣ-Wandlers sind die Kontrolllinien, wie auch die SPI-Schnittstelle ersichtlich. Hierbei ist ersichtlich, dass die Kontrolllinien, wie auch die DIN-Linie für alle A/D-Wandler dieselben sind. Dies wird so gemacht, da alle Kanäle mit derselben Konfiguration und Synchron betrieben werden sollen.

Der Pin PINMODE wird auf Masse gezogen. So wird der Funktionsmodus des ADS1281 festgelegt. Hierbei kann zwischen Pin-Modus und Register-Modus gewählt werden. In diesem Fall wird der A/D-Wandler im Register-Modus betrieben, da dieser mehr Konfigurationsmöglichkeiten als der Pin-Modus bietet. Aus diesem Grund werden die Pins 7, 8 und 9 auch auf Masse gezogen.

Weitere Informationen zur Beschaltung sind dem Datenblatt [1] zu entnehmen.

3.3.3 Der Delta-Sigma-Wandler in der Messkette

3.3.3.1 Wertebereich

Der ΔΣ-Wandler führt den Übergang vom analogen zum digitalen Messsignal durch. Da der durch die analoge Schaltung vorgegebene Eingangswertebereich nicht dem des A/D-Wandlers selber entspricht resultiert auch nicht den vorgeschriebenen Temperaturbereich.

Der Wertebereiche $-1.212V < U_{Mes2} < 1.165V$ der unter *3.2.3.3 Signalanpassung* definiert wurde entspricht nicht dem effektive vorhanden Wertebereich, welcher $-1.25V < U_{Mes2} < 1.25V$ ($-V_{REF}/2 < U_{Mes2} < V_{REF}/2$) entspricht. Dies bedeutet, dass der Wertebereich der durch die Spannung am PT1000-Widerstandsthermometer erzeugt wird nicht $0.823V < U_{Mes1} < 1.646V$ sondern $0.818V < U_{Mes1} < 1.6513V$ beträgt.

Der effektive Temperaturwertebereich beträgt dann $-53.224℃ < T < 157.166℃$.Dies geht über die Vorgabe hinaus. Das Risiko bei der Messung der Extremwerte (-50℃ und 150℃) irgendwo in der Messkette an die Bereichsgrenze zu stossen wird so gemindert.

Der Wertebereich der digitalen Werte wurde unter *3.3.1.2 Eigenschaften* beschrieben.

3.3.3.2 Störabstand

Da nicht mit der nominellen Taktfrequenz von 4.096MHz sondern nur mit einer Taktfrequenz von 1.024MHz gearbeitet wird, wird der ursprüngliche Störabstand nicht erreicht. Dem Datenblatt zu Folge beträgt der Störabstand mit dem tieferen Takt 124dB statt 130dB. Aus diesem Grund sind schlussendlich auch nur 18 statt idealerweise 19 Bits stabil.

Mehr zu diesem Sachverhalt kann im Abschnitt *9.2.3 Analog-Digital-Wandlung* gefunden werden.

3.4 Mezzanine-Verbindung

Die Schnittstelle zwischen der Erweiterungskarte und der FPGA-EBS-Karte umfasst alle Verbindungen die nötigt sind. Dabei handelt es sich um digitale Verbindungen.

Das Schema der Verbindung selber kann im Anhang *14.3.2 Elektrisches Schema der Erweiterungskarte (Version 1.2)* gefunden werden. Das Schema wurde mit Hilfe des Schemas der FPGA-EBS-Basiskarte erstellt werden. Dieses Schema heisst *FPGA_EBS.sch* und kann unter *P:/PCB/Produit/Logical/FPGA-EBS/FPGA_EBS_V1.1* gefunden werden.

Trotzdem soll hier gezeigt werden, wie das System mit mehreren Erweiterungskarten funktioniert. Abbildung 3-15 zeigt das Schema, welches dieses Prinzip verkörpert.

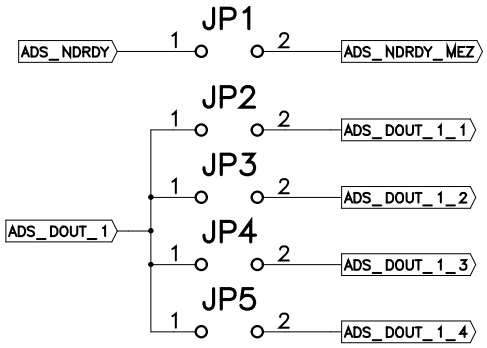


Abbildung 3-15: Schema des Auswahlmechanismus der Erweiterungskarten

Wie zu sehen ist kann die DataReady-Linie verbunden werden oder nicht. Diese Linie gibt einen Impuls sobald neue Konvertierungsdaten vorhanden sind. Diese wird nur bei einem Kanal aller vorhandenen verbunden, da alle A/D-Wandler synchron laufen und diesen Impuls somit zeitgleich aussenden. Die unteren vier Jumper sind da um das Board auszuwählen. Hierbei wird ein Kanal, welcher auf allen Jumper angeschlossen ist, auf eine der vier Linien gebrückt. Beispielsweise wird der Kanal 1 der ersten Erweiterungskarte auf die Verbindung 1_1 gelegt. Der Kanal 1 der zweiten Karte wird dann auf die Verbindung 1_2 gelegt, usw. Dies funktioniert für die Kanäle 2 bis 4 auf dieselbe Art und Weise.

Es kann also festgestellt werden, dass mit der Positionierung der Jumper die ID der Erweiterungskarte festgelegt wird. Die Limitierung des Systems auf 16 Kanäle wird so auch sichtbar.

Die Abbildung 3-16 zeigt die Beschaltung der auf die Erweiterungskarten hinführenden Datenlinien. Dabei werden die digitalen Signale gefiltert um den Einfluss auf die umliegenden Schaltungsteile zu mindern. Im Extremfall könnten diese Signale eine Verschlechterung der Schaltungsqualität verursachen. Dies wird allerdings mit eben diesen Filtern verhindert.

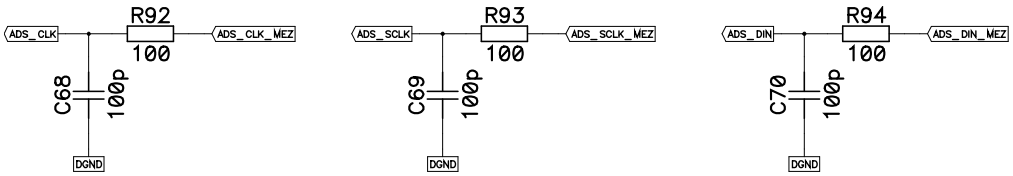


Abbildung 3-16: Eingangsfilterung der digitalen Signale

Es handelt sich dabei um RC-Tiefpass-Filter 1. Ordnung. Die Grenzfrequenz beträgt

$$f_C = \frac{1}{2 \cdot \pi \cdot R \cdot C} = \frac{1}{2 \cdot \pi \cdot 100\Omega \cdot 100pF} = 15.92MHz$$

Mit der relativ hohen Grenzfrequenz bleibt die Grundform, d.h. die wichtigen Frequenzanteile, vorhanden, wobei Frequenzanteile mit sehr hohen Frequenzen abgeschwächt werden.

3.5 FPGA

3.5.1 Allgemeines

Bereits vorweg wurde festgelegt, dass eine FPGA verwendet wird, da dieses für die parallele Verarbeitung mehrere Kanäle geeignet ist. Damit die nötige Hardware für den Betrieb der FPGA nicht hergestellt werden muss wird ein von der Hochschule in Sion entwickelte FPGA-Karte verwendet. Die Abbildung 3-17 zeigt diese Karte.

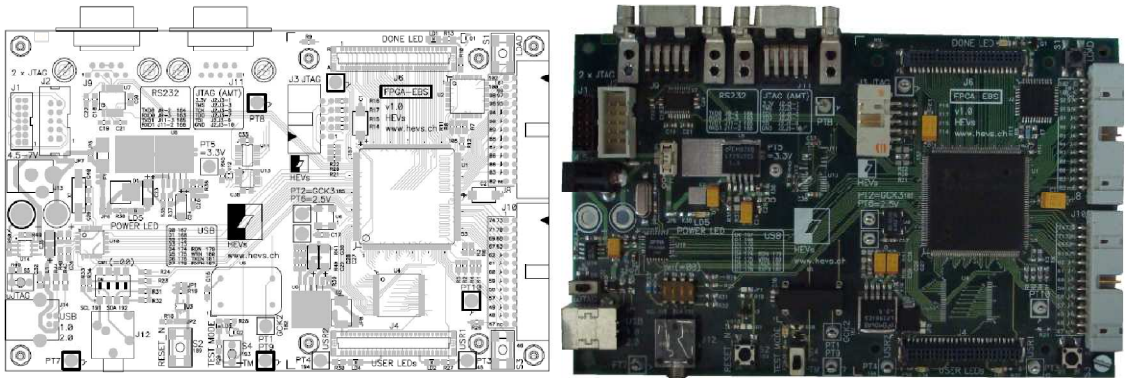


Abbildung 3-17: FPGA-EBS-Karte in der Version 1.0

Diese Karte umfasst alles was benötigt wird. Dazu gehören eine Mezzanine-Verbindung, die FPGA, ein PROM und eine USB-Schnittstelle. Das PROM wird benötigt, damit die Konfiguration[8] der Schaltung auf der Karte gespeichert werden kann und nicht jedes Mal vom PC aus geladen werden muss.

Bei der FPGA handelt es sich um das Model XC2S200 aus der Spartan-II FPGA Familie. Die genaue Bezeichnung der verwendeten FPGA lautet XC2S200-5PQ208. Die FPGA wird mit einem 66MHz-Takt betrieben.

Weitere Informationen zu der verwendeten FPGA-Karte befinden sich unter *P:/PCB/Produit/Logical/FPGA-EBS/FPGA_EBS_V1.1*.

Für die Entwicklung der FPGA-Schaltung wurden die folgenden Hilfsmittel verwendet.

- *HDL-Designer V2007.1a von Mentor Graphics*: Beschreibung der Schaltung
- *ModelSim SE V6.3g von Mentor Graphics*: Simulation und Überprüfung der Schaltung
- *Xilinx ISE Design Suite Software V10.1 von Xilinx*: Synthese und Implementierung der Schaltung

Zur Beschreibung der Hardware wird eine Hardwarebeschreibungssprache für komplizierte digitale Systeme (Very High Speed Integrated Circuit Hardware Description Language, VHDL) verwendet. Der verwendete VHDL-Standard stammt aus dem Jahre 1993 und ist in der Norm IEEE1076 beschrieben.

Das HDL-Projekt und das Xilinx ISE Projekt befinden sich mit dem gesamten Inhalt auf der CD im Verzeichnis *vhdl*.

3.5.2 Beschreibung der Schaltung

Die Schaltung wird hier in ihre einzelnen Blöcke zerlegt und erklärt. Diese Blöcke geben der Schaltung eine Struktur und machen diese so übersichtlich und besser verständlich.

[8] Konfiguration: Der beschriebenen Schaltung entsprechende Konfiguration der FPGA

Im Anhang befinden sich die eigens erstellten VHDL-Dateien. Die von der Software HDL-Designer generierten und aus anderen Projekten übernommene Dateien befinden sich nicht im Anhang, können allerdings auf der CD im Verzeichnis *vhdl* eingesehen werden

3.5.2.1 FPGA_tempArray

Funktion

Dieser Block beinhaltet die gesamte Schaltung. Hier wird die Schnittstelle zwischen der Aussenwelt und dem eigentlichen Hauptblock tempArray erstellt.

Diese Schnittstelle beinhaltet die Definition der Ein- und Ausgänge wie auch deren Synchronisierung und falls nötig auch die Invertierung der Signale.

Dieser Block und die zur Synchronisation und Invertierung benötigten Komponenten befinden sich in der Bibliothek Board. Dies wird so gemacht damit die Teile der Schaltung welche effektiv auf die FPGA geladen werden vom Rest separiert werden. Dies führt zu einer besser Übersicht im HDL-Designer-Projekt.

Struktur

Die Hauptschaltung des gesamten Projektes befindet sich wie bereits erwähnt in diesem Block. Die Abbildung 3-18 zeigt diese Schaltung und ihre Ein- und Ausgänge.

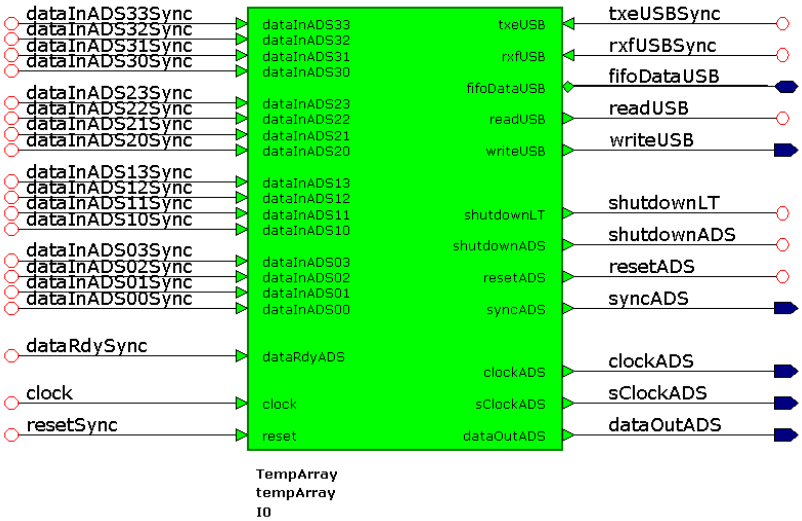


Abbildung 3-18: Der Block tempArray im Hauptblock FPGA_tempArray

Die dataInADSNMSync-Linien sind die synchronisierten, seriellen Datenlinien der A/D-Wandler welche die Daten in die Schaltung übertragen. Das Signal dataRdySync gibt mit einem Impuls an, wann neue Daten gesendet werden. Rechts oben befinden sich die Verbindungen, welche zur Kommunikation über USB benötigt werden. Die einkommenden Linien werden auch synchronisiert. Wie diese Kommunikation genau funktioniert wird unter *3.6 USB-Verbindung* erklärt. Des Weiteren sind die Kontroll- und Takt-Linien, welche benötigt werden um die A/D-Wandler zu betreiben, ersichtlich. Welche Rolle die einzelnen Verbindungen spielen und wie diese generiert werden, wird in den folgenden Kapiteln erklärt.

Der Eingang clock entspricht dem Systemtakt der FPGA und wird in den folgenden Kapiteln als clock bezeichnet. Der Eingang reset ist der Eingang, welche die gesamte Schaltung zurücksetzt und wird nun nur mit reset bezeichnet.

Die Abbildung 3-19 zeigt einen weiteren Teil des Inhalts des Hauptblocks. Eine Synchronisierung und eine Invertierung werden gezeigt. Das blaue Symbol an beschreibt

den Eingang des Signals in die Schaltung. Ein entsprechendes Symbol wird für die Ausgänge verwendet.

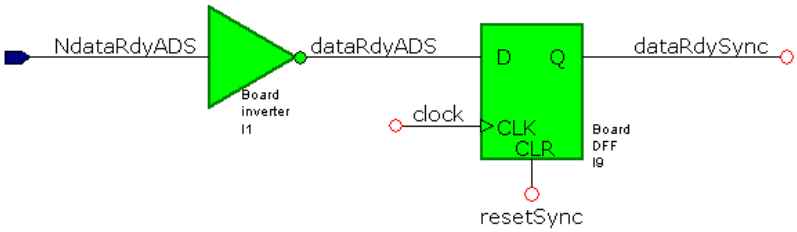


Abbildung 3-19: Invertierung und Synchronisierung am Eingang der FPGA

Die Synchronisierung wird mit Hilfe eines normalen D-Flipflops durchgeführt. Die Invertierung wird mit einem Inverter durchgeführt. Für die Synchronisierung wird nebst dem Reset-Signal auch das Takt-Signal der FPGA (clock) benötigt.

Die erstellten VHDL-Dateien dieses Blockes sind im Anhang unter *14.5.1 VHDL-Dateien des Blockes FPGA_tempArray* ersichtlich.

Die generischen Parameter, welche im folgenden Kapitel erklärt werden, werden auf dieser Stufe der Struktur festgelegt.

Simulation

Da sich die komplexen Funktionen im Hauptblock tempArray befinden und dieser ansonsten nur Flipflops und Inverter enthält ist hier keine Simulation durchgeführt worden.

3.5.2.2 tempArray

Funktion

Dieser Block umfasst alle Blöcke, welche für die Ausführung der Funktion, welche in der FPGA implementiert werden soll, verantwortlich sind. Dieser und die zugehörigen Blöcke befinden sich in der Bibliothek tempArray.

Die Zusammenfassung der verschiedenen seriellen Eingänge zu einem Vektor wird auf dieser Ebene durchgeführt. Es ist auch ersichtlich, dass die in der FPGA implementierte Schaltung für maximal 16 Kanäle ausgelegt ist.

Der Block tempArray ist in der Abbildung 3-20 dargestellt.

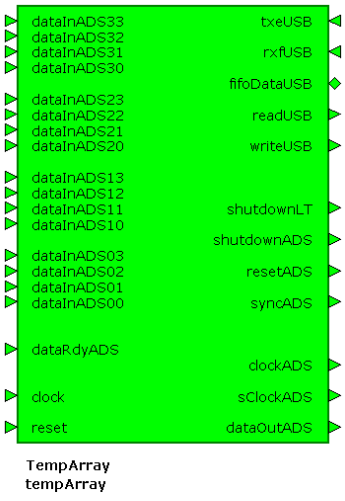


Abbildung 3-20: Block tempArray

Die Eigenschaften der Schaltung können mit Hilfe der generischen Parameter festgelegt werden. Die Tabelle 3-2 zeigt all diese Einstellungsmöglichkeiten, wobei die Funktion mancher Parameter erst in den nächsten Kapiteln erklärt wird.

Parameter	Typ	Standardwert	Bedeutung
nbOfChanPerBoard	positive	4	Anzahl Kanäle pro Erweiterungskarte
nbOfBoards	positive	1	Anzahl Erweiterungskarten
ADSDataBitNb	positive	32	Anzahl Datenbits vom ADS1281 selber
ADSDataHeadBitNb	positive	40	Anzahl Datenbits inkl. Header
USBIntDataBitNb	positive	40	Entspricht ADSDataHeadBitNb
USBExtDataBitNb	positive	8	Anzahl Datenbits der USB-Schnittstelle
nbOfValsToSum	integer	256	Anzahl Werte zur Mittelwertsbildung
fifoDepth	positive	4	Anzahl Speicherstellen im FIFO-Speicher
clkDiv	positive	64	Takt für den ADS1281
sClkDiv	positive	528	Takt für den seriellen Bus des ADS1281

Tabelle 3-2: Generische Parameter der FPGA-Schaltung und deren Bedeutung

Wie zu sehen ist sind die meisten Parameter vom Typ positive, da sie sicherlich nie negativ sind. Nur nbOfValsToSum ist ein integer, da dies für die Weiterverarbeitung nötig ist.

Struktur

Die folgende Abbildung 3-21 zeigt die Struktur des Blocks tempArray.

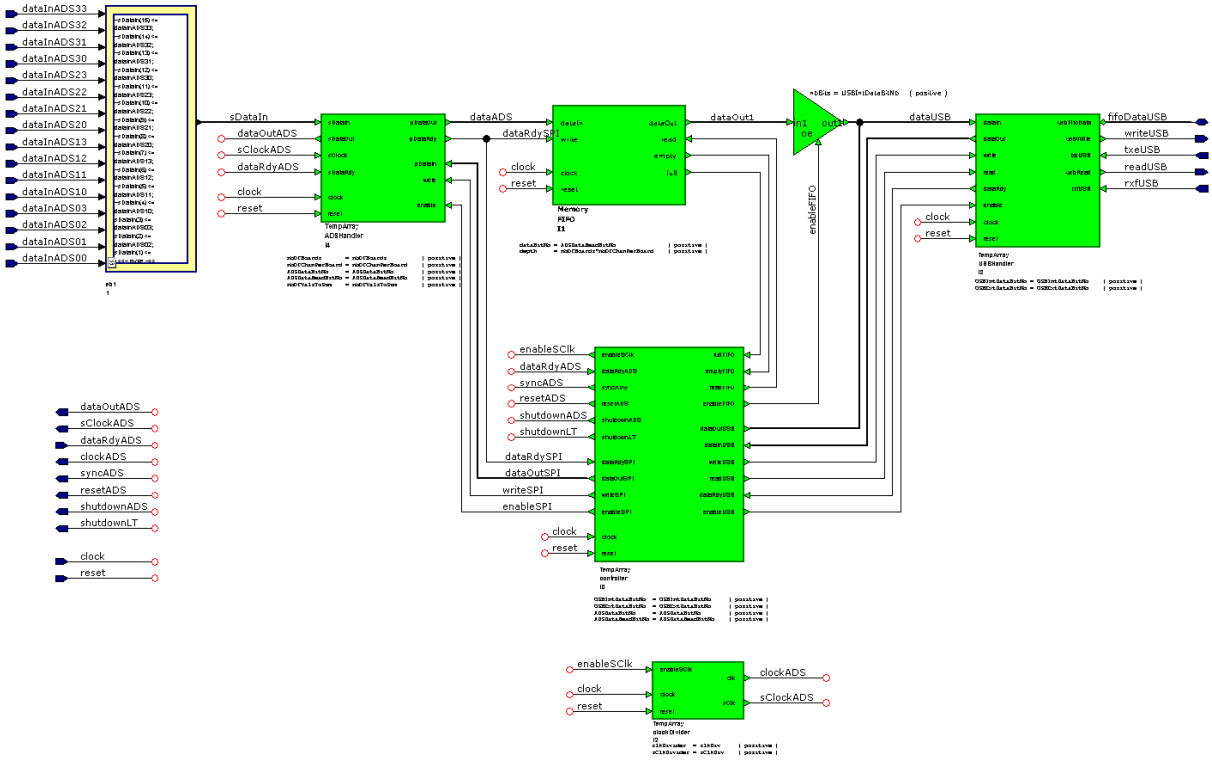


Abbildung 3-21: Struktur des Blockes tempArray

Links oben ist die Bündelung der einzelnen Dateneingänge (dataInADSNM) der A/D-Wandler zu einem Vektor (sDataIn) ersichtlich. Dies wird gemacht, da das System so leichter und übersichtlicher wird. Danach werden diese Daten an den Block ADSHandler übertragen. Dieser deserialisiert und mittelt die Daten. Die gemittelten Werte werden dann über den Bus dataADS im Speicher-Block FIFO abgelegt. Sobald die USB-Verbindung verfügbar ist, führt der Kontroll-Block den Versand der Daten ein. Hierzu werden die Daten welche im Speicher abgelegt wurden auf den Bus dataUSB gelegt. Dies wird mit Hilfe eines Tri-State-Puffers gemacht. Der Block USBHandler nimmt die Daten dann entgegen und versendet diese über den Ausgang fifoDataUSB.

Der Kontroll-Block, namens controller, befindet sich im Zentrum der Struktur und führt auch die Initialisierung und die Steuerung der umliegenden Blöcke durch.

Am unteren Rand der Abbildung 3-21 ist der Block clockDivider zu sehen, welcher die Taktsignale für den Betrieb der A/D-Wandler generiert.

Zudem sind links unten die Ein- und Ausgänge der Kontrollinien, wie auch die Systemlinien clock und reset ersichtlich. Am rechten Bild sind die Verbindungen ersichtlich, welche zur Kommunikation mit der USB-Schnittstelle benötigt werden.

Die einzelnen Blöcke werden in den folgenden Kapiteln erklärt.

Simulation

Um die Funktionalität der gesamten Schaltung überprüfen zu können wurde eine Simulation durchgeführt. In dieser Simulation werden am Eingang Werte gesendet. Nach der Mittelung von 256 Werten sollte dann am Ausgang dieser gemittelte Wert in der Form von USB-Daten erscheinen. Die Abbildung 3-22 zeigt diese Simulation.

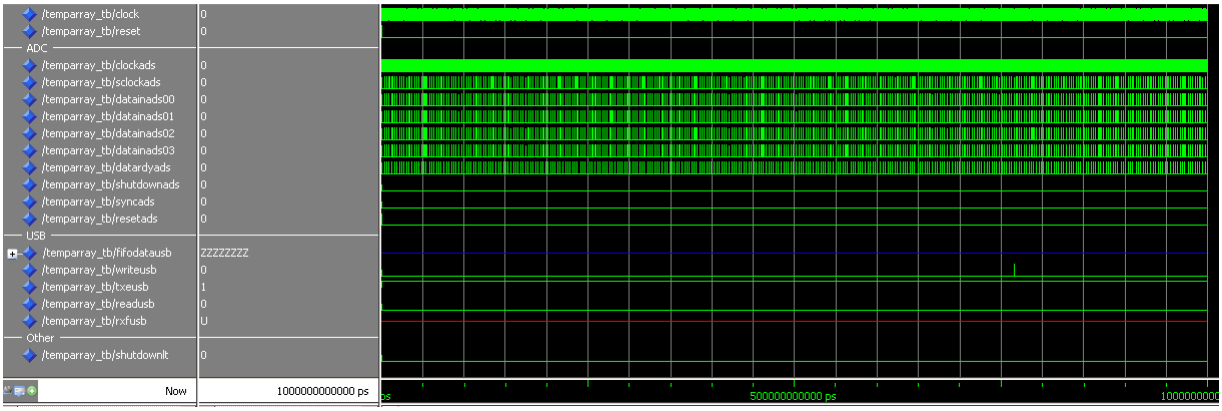


Abbildung 3-22: Simulation des Blockes tempArray (Aufnahme der Daten)

Es ist zu erkennen, dass nach 256 Werten am Ausgang Daten geschrieben werden. Auf dem Ausgangsbus fifoDataUSB sind keine Daten ersichtlich, da die zeitliche Auflösung zu gross ist um diese Darzustellen.

Die Abbildung 3-23 zeigt den vergrösserten Ausschnitt des Versands der Daten über USB.

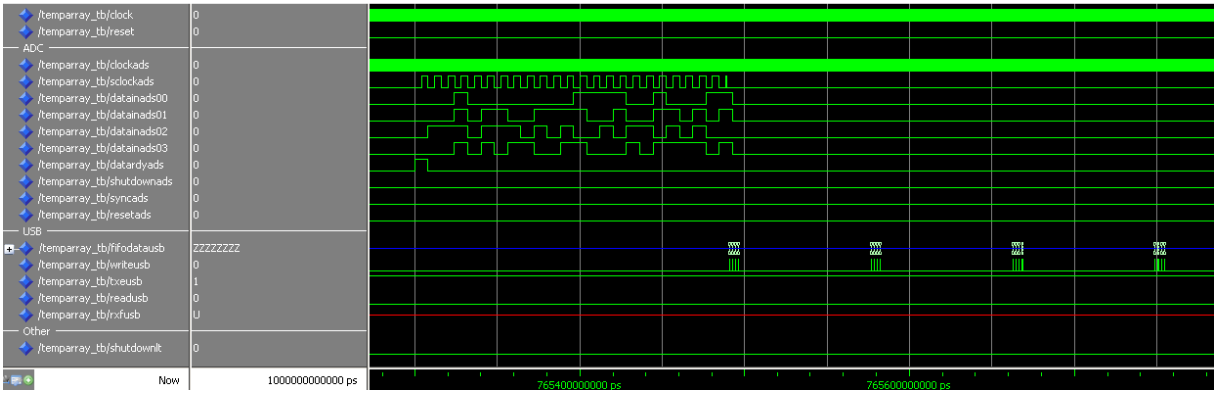


Abbildung 3-23: Simulation des Blockes tempArray (Ausgabe der Daten)

Unmittelbar nachdem die letzten Daten über die dataInADSNM-Linien empfangen wurde werden die Daten über USB versendet. Vier Pakete sind ersichtlich, wobei jedes Paket vier Bytes umfasst. Diese Art des Datenversands ist korrekt. Warum die Daten so versendet werden müssen wird unter *3.5.2.6 USBHandler* erklärt.

Die Korrektheit der versendeten Daten wurde manuell überprüft.

Die zur Simulation verwendete Architektur ist im Anhang *14.5.2 Testbank der Simulation des Blockes tempArray* ersichtlich.

Auch wenn diese Simulation bereits die Funktionalität der gesamten Schaltung bestätigt werden in den folgenden Kapiteln die Blöcke einzeln simuliert.

Der Versand der Daten an die A/D-Wandler wurde nicht überprüft, da dieser noch nicht implementiert wurde.

3.5.2.3 clockDivider

Funktion

Dieser Block (Abbildung 3-24) enthält zwei Zähler, welche die benötigten Taktsignale für den Betrieb der A/D-Wandler erstellen.

Struktur

Der Block clockDivider enthält lediglich zwei Zähler und eine Vorrichtung welche das Zählen erlaubt oder nicht und keine interne Struktur.

Der Systemtakt der A/D-Wandler f_{clkADS} wird generiert. Hierbei wird der Takt der FPGA f_{clk} mit dem Faktor 64 geteilt. So ergibt sich ein Takt von 1.03125MHz. Ein solch untypischer Wert kommt zu Stande, da ein 1.024MHz-Takt erstellt werden soll, dies aber mit dem Takt der FPGA (66MHz) nicht möglich ist. Der Systemtakt des ADS1281 ist lediglich durch einen Bereich, welcher von 1MHz bis 4.096MHz reicht, eingeschränkt. Allerdings erhält ergibt sich mit einem Takt von 1.03125MHz nicht eine Datenrate von 250SPS sondern 251.77SPS. Dies bedeutet, dass die Datenrate der gemittelten Werte nicht 0.977SPS sondern 0.984SPS beträgt und so alle 1.017 Sekunden ein neuer Mittelwert bereitsteht.

Der Takt des seriellen Busses f_{sclk} wird wie der Systemtakt auch vom Haupttakt der FPGA abgeleitet. Hierbei wird mit dem Faktor 528 geteilt, was zu einem Takt von 125kHz führt. Der Takt des seriellen Busses ist im Vergleich zum Takt der FPGA langsam. Dies wird so gemacht, dass genügend Zeit für die Verarbeitung der seriellen Daten vorhanden ist. Der Takt des seriellen Busses ist lediglich durch einen Bereich, welcher von $f_{clkADS}/2$ bis $f_{clkADS}/16$ reicht, eingeschränkt. Mit 125kHz liegt der Wert bei ca. $f_{clkADS}/8$.

Der Block ist in Abbildung 3-24 dargestellt. clock wie auch reset bilden zwei Eingänge. clock wird benötigt da dieser die Grundlage des Blockes darstellt und reset wird zum initialisieren des Blockes benötigt. Der Eingang enableSClk schaltet den Zähler des Ausgangs sClk ein oder aus. sClk entpsricht dem Takt des seriellen Datenaustausches mit den A/D-Wandlern. Der Systemtakt der A/D-Wandler clk wird immer generiert.

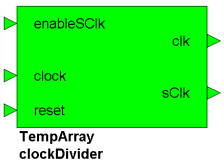


Abbildung 3-24: Block clockDivider

Die erstellten VHDL-Dateien dieses Blockes sind im Anhang unter *14.5.3 VHDL-Dateien des Blockes clockDivider* ersichtlich.

Simulation

Zur Kontrolle der Funktion wurde eine Simulation durchgeführt. Die Abbildung 3-25 zeigt diese. Der Systemtakt der A/D-Wandler clk wird immer generiert und der Takt des seriellen Bussen wird generiert sobald die Linie enableSClk auf ‘1‘ geht.

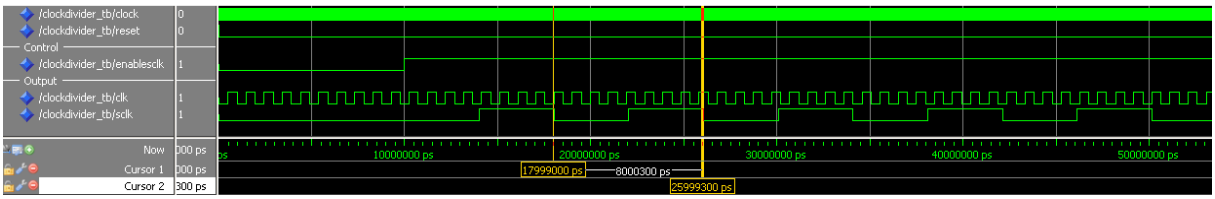


Abbildung 3-25: Simulation des Blockes clockDivider

Die beiden Takte wurden mit Hilfe der Kursoren kontrolliert. Hier ist dies am Beispiel des sClk gezeigt. Mit Hilfe der gemessenen Periode lässt sich die Frequenz bestimmen.

$$f_{sclk} = \frac{1}{T_{sclk}} = \frac{1}{8us} = 125kHz$$

Da die Auflösung des Simulators zu wenig genau ist beträgt die Periode nicht exakt 8us.

Die zur Simulation verwendete Architektur ist im Anhang *14.5.4 Testbank der Simulation des Blockes clockDivider* ersichtlich.

3.5.2.4 ADSHandler

Funktion

Der Block ADSHandler (Abbildung 3-26) hat die Aufgabe die seriellen Daten zu empfangen und diese zu deserialisieren. Danach werden die Daten in diesem Block gemittelt und im FIFO-Speicher abgelegt. Ein Block, welche die Konfiguration der A/D-Wandler ermöglichen soll ist auch in diesem Block enthalten.

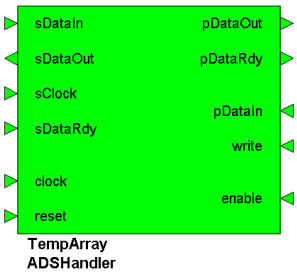


Abbildung 3-26: Block ADSHandler

Struktur

Die Abbildung 3-27 zeigt die Struktur des Blockes ADSHandler.

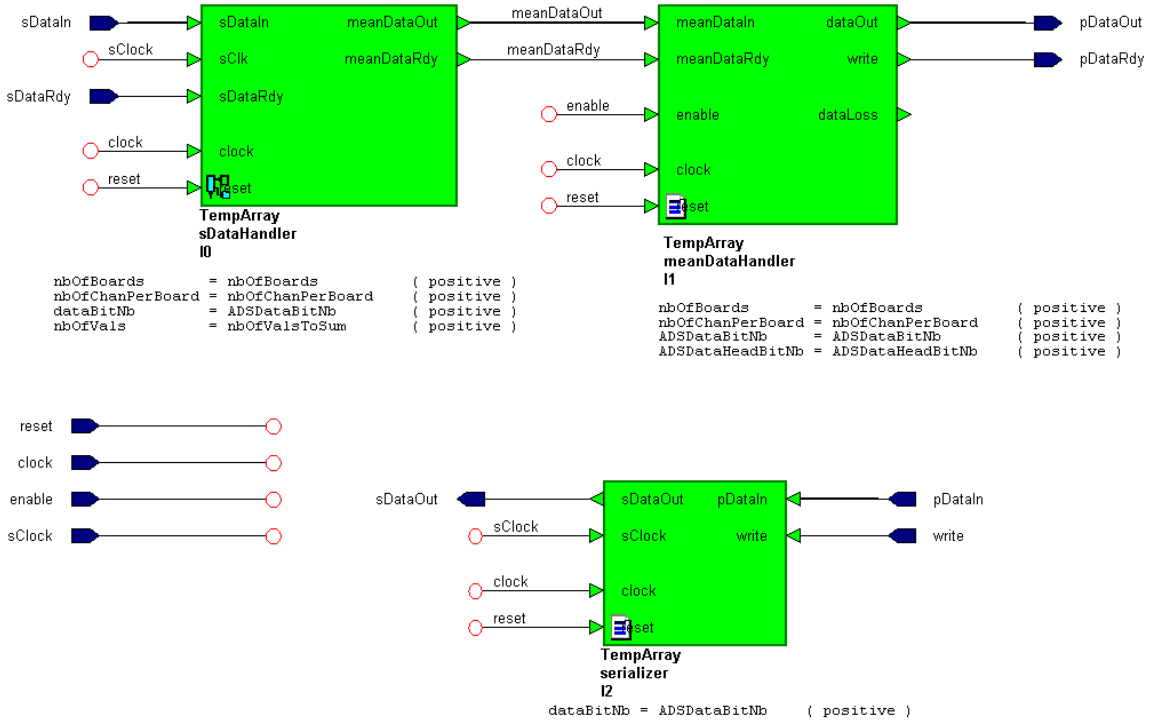


Abbildung 3-27: Struktur des Blockes ADSHandler

Die zum Vektor sDataIn zusammengefügten Datenlinien der A/D-Wandler werden an den Block sDataHandler übergeben. Der Eingang sDataRdy informiert mittels eines Impulses über die Ankunft neuer Daten. In diesem Block sind so viele Schieberegister der Grösse der Daten des ADS1281 vorhanden wie Kanäle vorhanden sind. Wie viele Schieberegister vorhanden sind, ist durch die beiden Parameter nbOfChanPerBoard und nbOfBoards gegeben. Die Schieberegister schieben ein Bit nach dem anderen ein und geben, sobald die Daten komplett sind einen Impuls weiter an den nächsten Block. Die Grösse der Daten des ADS1281 wird mit dem generischen Parameter ADSDataBitNb bestimmt. Dieser ist standardmässig 24 und nicht 32 eingestellt. Warum die wertniedrigsten 8 Bit einfach weggelassen werden können wurde im Kapitel *3.3.1.3 Delta-Sigma-Modulator* erklärt.

Auch im Block sDataHandler werden die Mittelwerte berechnet. Es sind so viele Blöcke zur Berechnung der Mittelwerte wie Kanäle vorhanden. Diese Blöcke nehmen die Daten entgegen und summieren diese auf. Der generische Parameter nbOfValsToSum bestimmt hierbei wie viele Werte summiert werden sollen, bevor durch die Anzahl der summierten Werte geteilt wird. So wird der Mittelwert erstellt. Die Werte, welche dem generischen Parameter nbOfValsToSum gegeben werden können, müssen eine eine Potenz der Zahl 2 sein, da dies die Teilung zum Schluss der Mittelwertsberechnung erfordert. Diese Teilung wird durch eine Verschiebung nach rechts realisiert.

Die Anzahl Werte, welche für die Mittelwertsbildung verwendet werden könnte auch per Register vorgegeben werden. D.h. falls dieser Parameter während dem Betrieb konfigurierbar sein soll müsste ein 16Bit-Register vorgesehen werden, in welchem dieser Parameter abgelegt wird. Bei der Berechnung des Mittelwerts würde dann diesen Wert anstelle des generischen Parameters nbOfValsToSum ausgelesen und verwendet.

Sobald die Mittelwerte bereitstehen wird der nächste Block (meanDataHandler) wiederum mit Hilfe eines Impulses darüber informiert. Dieser nimmt dann diese Mittelwerte der einzelnen Kanäle und fügt diesen einen Daten-Kopf hinzu. Danach werden die Werte im

FIFO-Speicher[9] abgespeichert. Diese Daten-Köpfe werden für das Protokoll der Kommunikation mit dem PC benötigt und werden unter *3.6.3 USB-Protokoll* erklärt.

Beim deserializer handelt es sich um den unter *Funktion* erwähnten Block, welcher für den Versand von Daten an die A/D-Wandler zuständig wäre. Dieser Block wird serializer genannt, da er die Daten welche an die A/D-Wandler gesendet werden sollen serialisiert und überträgt. Dieser Block wurde allerdings noch nicht getestet und wird zurzeit auch nicht verwendet, was bedeutet, dass die A/D-Wandler und auch das System während dem Betrieb nicht konfigurierbar sind. Lediglich mit den generischen Parametern können einige Einstellungen vorgenommen werden.

Die erstellten VHDL-Dateien dieses Blockes sind im Anhang unter *14.5.5 VHDL-Dateien des Blockes ADSHandler* ersichtlich.

Simulation

Um die Funktion des ADSHandlers zu überprüfen wurde eine Simulation durchgeführt. In der Testbank wurden für vier Kanäle Eingangssignale generiert. Die Abbildung 3-28zeigt den Versand von einem simulierten Messwert an den Block.

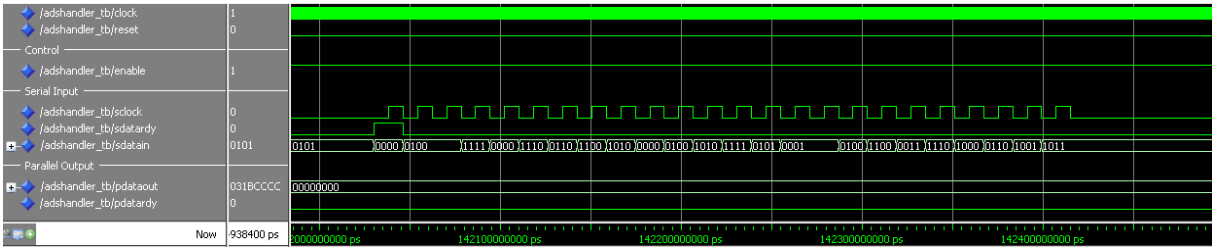


Abbildung 3-28: Simulation des Blockes ADSHandler (Versand eines Messwertes)

Die Generierung der Daten wie auch des Impulses, welcher die Information eines neuen Messwertes transportiert, ist ersichtlich.

Die Abbildung 3-29 zeigt die Ausgabe von 256 gemittelten Werten, welche manuell überprüft wurde.

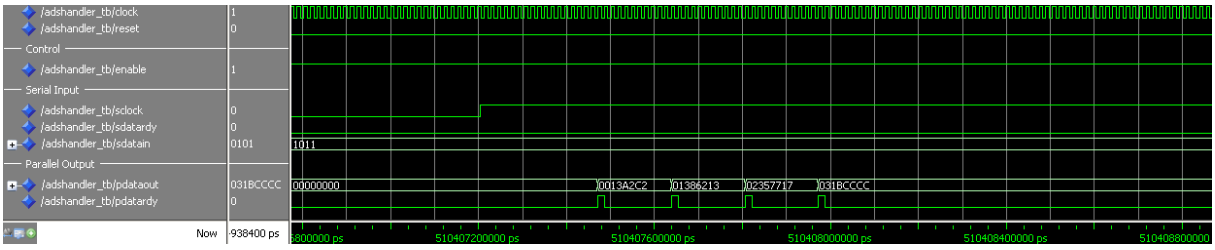


Abbildung 3-29: Simulation des Blockes ADSHandler (Ausgabe der Daten)

So konnte die Funktionalität dieses Blockes kontrolliert werden. Der Versand der Daten wurde nicht kontrolliert, da diese, wie bereits erwähnt, von der restlichen Schaltung noch nicht unterstützt wird.

Die zur Simulation verwendete Architektur ist im Anhang *14.5.6 Testbank der Simulation des Blockes ADSHandler* ersichtlich.

[9] FIFO-Speicher: (First in first out) Die Werte, welche zuerst gespeichert werden, werden auch zuerst wieder ausgelesen (Puffer-Funktion)

3.5.2.5 FIFO-Speicher

Funktion

Der FIFO-Speicher speichert die erstellten Mittelwerte und übernimmt so die Rolle eines Puffers. Dies wird gemacht, da die USB-Verbindung nicht immer Verfügbar sein muss. So kann ein Datenverlust verhindert werden.

Es handelt sich beim FIFO-Speicher um einen simplen Speicher, welcher aus der Bibliothek Memory der Hochschule stammt.

Mit dem generischen Parameter fifoDepth kann die Anzahl der Speicherstellen bestimmt werden. Hierbei wird standardmässig 4 gesetzt, denn während einer Sekunde ist der USB mit grosser Wahrscheinlichkeit irgendwann frei. Falls trotzdem bedenken über Datenverlust vorhanden sein sollten kann dieser Parameter vergrössert werden. Dies ist insofern unbeschränkt möglich als Platz in der FPGA vorhanden ist.

Struktur

Die Abbildung 3-30 zeigt das Symbol des FIFO-Speichers.

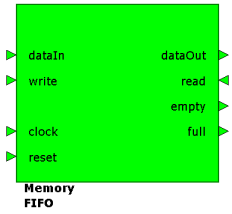


Abbildung 3-30: Block FIFO

Simulation

Da dies ein Element ist, welches aus einer bestehenden Bibliothek übernommen wurde, wurde keine Simulation durchgeführt.

3.5.2.6 USBHandler

Funktion

Der Block USBHandler organisiert die Kommunikation zwischen der FPGA und dem USB-Chip. Im Block USBHandler sind zwei Blöcke enthalten. Der erste Block (TXRXHandler) für die Anpassung des Datenformats in beide Richtungen durch. Der zweite Block (USBDriver) organisiert die Kommunikation mit dem USB-Chip.

Die Abbildung 3-31 zeigt den Block USBHandler.

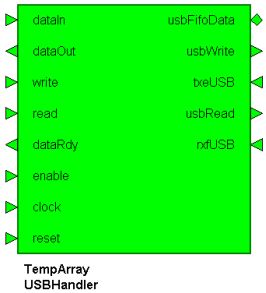


Abbildung 3-31: Block USBHandler

Struktur

Die Abbildung 3-32 zeigt die Struktur des USBHandler.

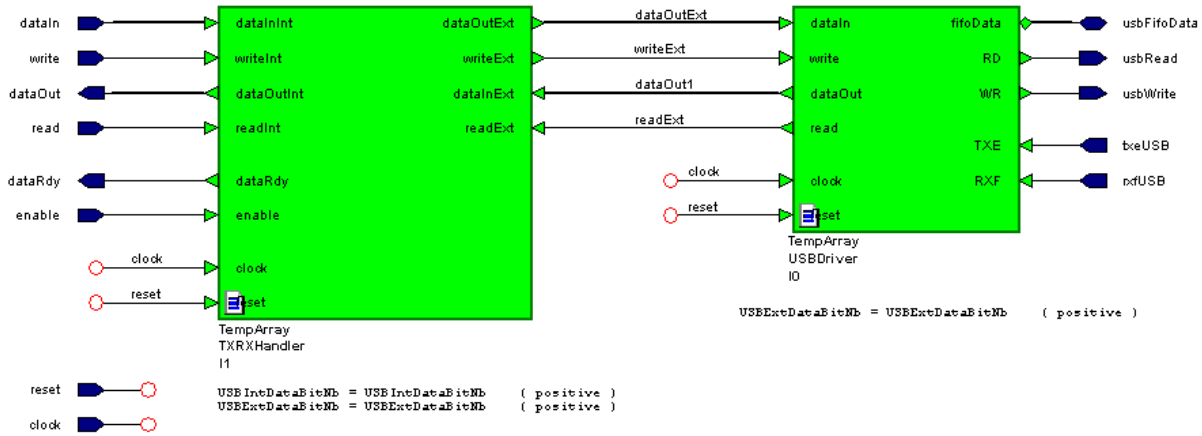


Abbildung 3-32: Struktur des Blockes USBHandler

Der erste Block, namens TXRXHandler, nimmt einerseits die von der USB-Schnittstelle kommenden 8Bit-Daten entgegen und bringt diese in die benötige Form. Diese Form kann je nach Daten variieren. Kontrollbefehle beispielsweise sind auf 8 Bit kodiert und Konfigurationsbefehle auf 40Bit. Mehr zum Kommunikationsprotokoll zwischen dem PC und der FPGA kann unter *3.6.3 USB-Protokoll* gefunden werden. Nebst dieser Aufgabe hat der TXRXHandler noch die Aufgabe die im FIFO-Speicher gespeicherten Daten entgegenzunehmen und an den USBDriver zu übergeben. Dabei müssen die Daten, welche standardmässig 32 Bit lang sind, in 8 Bit Daten aufgeteilt werden. Somit werden vier Schreibzyklen des USB-Interfaces benötigt um einen Mittelwert zu versenden.

Der zweite Block innerhalb des USBHandlers ist der Block USBDriver. Dieser organisiert den Datenaustausch mit der USB-Schnittstelle. Bei der USB-Schnittstelle handelt es sich um einen Chip, welcher als FIFO-Speicher gesehen wird. Dieser Block beinhaltet eine Zustandsmaschine und wurde aus dem Projekt [3] übernommen.

Da der Driver von dem Projekt [3] Übernomen wurde, wurde dieser nicht im Detail studiert. Die Funktionalität wurde aber trotzdem mit der Simulation überprüft. Dies bedeutet unter anderem, dass für diese Projekt der FTDI-Chip als transparent erscheint und nicht studiert wurde.

Die Zustandsmaschine ist fähig die Datenübertragung in beide Richtungen durchzuführen, allerdings nicht zur selben Zeit durchzurühren.

Es soll hier noch erwähnt werden, dass der Empfang der Daten soweit nicht implementiert ist. Das bedeutet, dass der TXRXHandler zwar für den Empfang der Daten von der USB-Schnittstelle her vorgesehen ist, dies allerdings zurzeit nicht unterstützt.

Die erstellten VHDL-Dateien dieses Blockes sind im Anhang unter *14.5.7 VHDL-Dateien des Blockes USBHandler* ersichtlich.

Simulation

Die Simulation dieses Blockes ist in dargestellt.

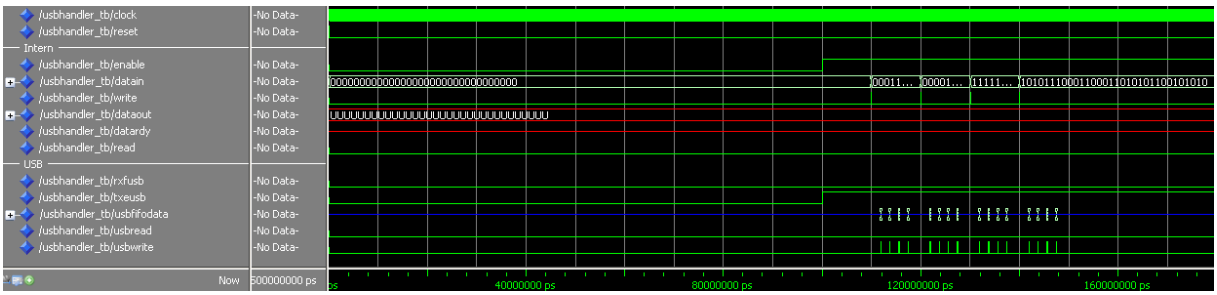


Abbildung 3-33: Simulation des Blockes USBHandler

Die Daten welche am Eingang dataIn gesendet werden sobald der Schreib-Impuls (write) folgen am Ausgang in Gruppen von 8 Bit gesendet. Dies entspricht der korrekten Funktionalität.

Die Korrektheit der Daten wurde manuell überprüft.

Die zur Simulation verwendete Architektur ist im Anhang *14.5.8 Testbank der Simulation des Blockes USBHandler* ersichtlich.

3.5.2.7 Controller

Funktion

Der Controller kontrolliert die Verarbeitung der Daten in der gesamten FPGA. Bei der aktuell vorhandenen Schaltung umfasst dies die Steuerung des USBHandlers, sobald Daten im FIFO-Speicher vorhanden sind und die Durchführung des Resets der gesamten Schaltung.

Falls der Empfang der Daten über die USB-Schnittstelle implementiert wird müsste der Controller für diese Aufgabe erweitert werden, da dies noch nicht implementiert wurde.

Der Kontroller (Abbildung 3-34) ist mit Hilfe einer Zustandsmaschine erstellt und wird folgend erklärt.

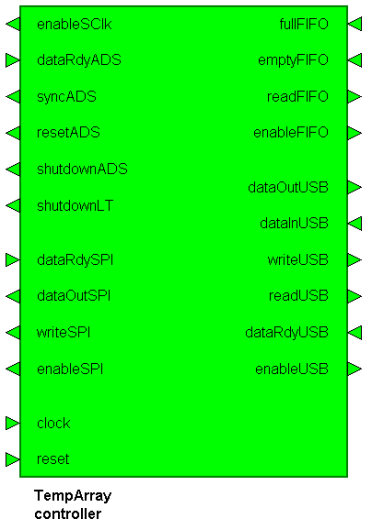


Abbildung 3-34: Block Controller

Struktur

Die Abbildung 3-35 zeigt die Zustandsmaschine welche im Controller vorhanden ist.

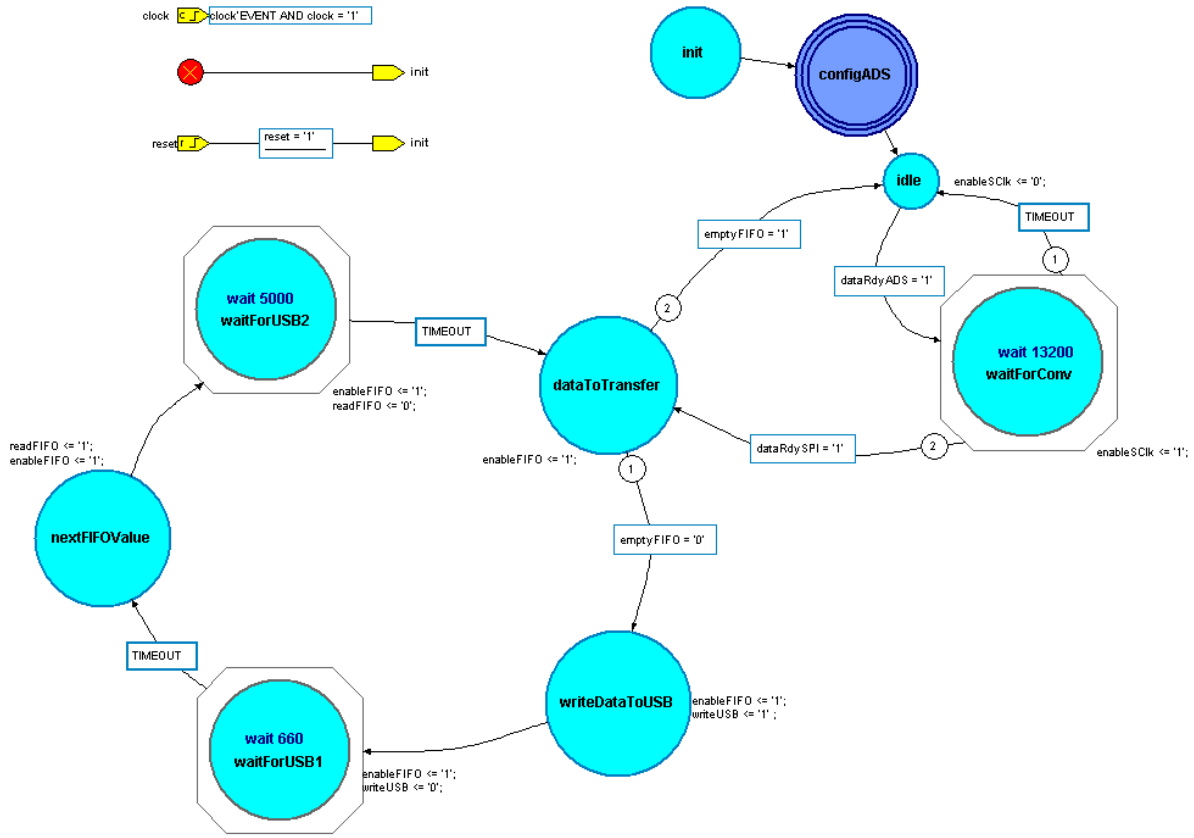


Abbildung 3-35: Zustandsmaschine des Blockes controller

Die Zustandsmaschine enthält neun Zustände (init, configADS, idle, waitForConv, dataToTransfer, writeDataToUSB, waitForUSB1, nextFIFOValue, waitForUSB2). Beim Zurücksetzten der Schaltung (reset-Signal auf ‘1‘) der Schaltung geht die Zustandsmaschine in den Zustand init über. Dieser führt weiter in den Zustand configADS, welcher die A/D-Wandler zurücksetzt und dann einschaltet.

Danach geht die Zustandsmaschine über in den Zustand idle in welchem sie wartet, bis ein Impuls auf dem Signal dataRdyADS ankommt, welcher besagt, dass neue Daten ankommen. Nach diesem Impuls wird im Zustand waitForConv gewartet bis die Daten der A/D-Wandler komplett übertragen, gemittelt und im FIFO-Speicher abgespeichert wurden.

Falls in der maximalen Zeit, welche diesen Aktionen und einer gewissen Marge entspricht, kein Impuls auf dem Signal dataRdySPI ankommt, geht die Zustandsmaschine zurück in den Zustand idle. Ansonsten kommt sie in den Kreis, welcher die Daten über USB versendet.

Hierzu wird im Zustand dataToTransfer erst kontrolliert ob überhaupt Daten vorhanden sind. Falls der FIFO-Speicher nicht leer ist wird auf die USB-Schnittstelle geschrieben (writeDataToUSB). Danach wird gewartet bis diese Aktion beendet ist (waitForUSB1). Da diese Daten nun versendet wurden wird der nächste Eintrag des Speichers angewählt (nextFIFOValue) und eine gewisse Zeit abgewartet (waitForUSB2) um das System nicht unnötig zu belasten. Solche Wartezeiten können eingebaut werden, da mit einer Datenrate von ca. 1SPS genügend Zeit vorhanden ist.

Nach dieser Wartezeit wird wiederum kontrolliert ob Daten im FIFO-Speicher vorhanden sind (dataToTransfer). Falls Daten vorhanden sind wird der Kreis wiederholt und ansonsten geht die Zustandsmaschine zurück in den Wartezustand idle.

Die erstellten VHDL-Dateien dieses Blockes sind im Anhang unter *14.5.9 VHDL-Dateien des Blockes controller* ersichtlich.

Simulation

Eine Simulation wurde durchgeführt. Hierbei wurden die Eingangssignale, welche die Zustandsmaschine fortfahren lässt der Realität entsprechend generiert. Die Abbildung 3-36 zeigt diese Simulation.

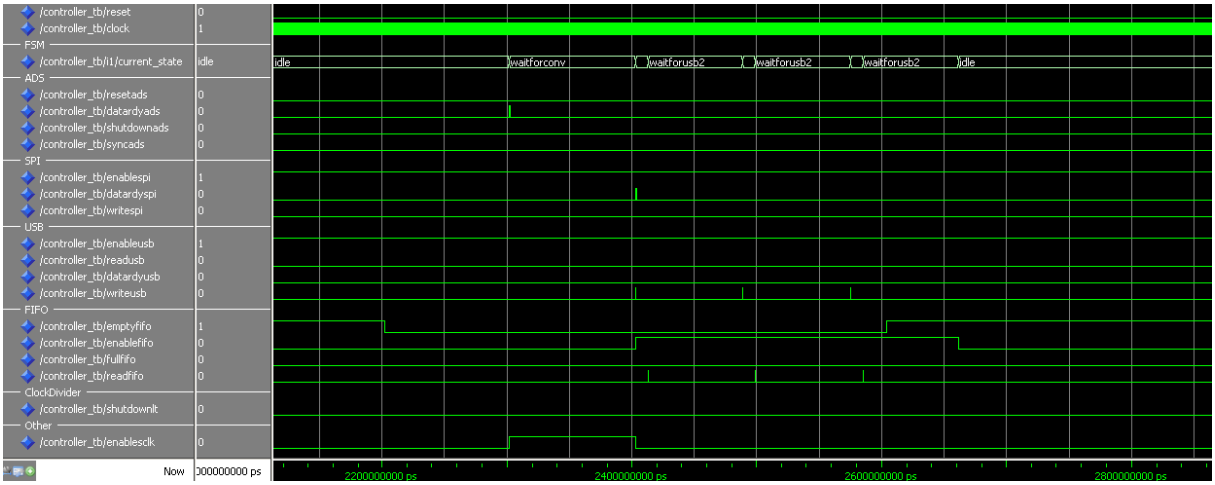


Abbildung 3-36: Simulation des Blockes controller (Übersicht)

Wie zu sehen ist werden sobald neue Daten ankommen gewartet bis diese deserialisiert und gemittelt wurden. Danach werden die Daten über USB Versendet, was durch die dreimalige Aktivierung des Zustanden waitForUSB2 ersichtlich ist. So konnte der korrekte Ablaufeder der Zustandsmaschine kontrolliert werden.

In der Abbildung 3-36 sind nicht alle Zustände ersichtlich, da die zeitliche Auflösung zu gross ist.

Die zur Simulation verwendete Architektur ist im Anhang *14.5.10 Testbank der Simulation des Blockes controller* ersichtlich.

3.5.2.8 Bibliothek TempArrayPkg

Damit die Signale innerhalb der Schaltung effizient und übersichtlich dargestellt werden können wurde eine eigene Bibliothek erstellt, welche die Definition eines Datentypen für die seriellen Daten der A/D-Wandler beinhaltet.

Da die Anzahl des benötigten Bits des Mittelwertes auch generisch errechnet werden soll wurde eine Funktion definiert welche bereits von der Hes-so Valais / Wallis Sion erstellt wurde. Die Funktion lautet `requiredBitNb()` und gibt die benötigten Anzahl Bits für die Aufnahme eines maximalen Zahlenwerts dar. Dieser maximale Wert stellt in diesem Fall den maximalen Wert des Zählers dar, welcher durch die Anzahl Werte der Berechnung des Mittelwertes definiert ist.

Die Bibliothek befindet sich im Anhang *14.5.11 Bibliothek TempArrayPkg*.

3.5.3 Anpassung der Schaltung

Falls die Anzahl der Kanäle geändert werden soll. Muss der generische Parameter der Anzahl Erweiterungskarten (nbrOfBoards) angepasst werden.

Des Weiteren muss die Übersetzung der Datenlinien im Block tempArray angepasst werden. Diese Übersetzung ist mittels eines Textblockes realisiert.

Der Datentyp, welcher in der Bibliothek TempArrayPkg definiert ist, muss auch auf die neue Anzahl Kanäle angepasst werden. Hierbei müssen die beiden Werte, welche die Anzahl Datenlinien angeben angepasst werden. Dabei handelt es sich um den Parameter spiParArrayLength und spiSerArrayLength.

3.5.4 Implementierung der Schaltung

Die beschriebene Schaltung wird dann mit Hilfe der Software Xilinx ISE Design Suite implementiert. Implementieren bedeutet in diesem Fall in die FPGA geladen.

Hierzu sind mehrere Schritte nötig. Nachdem die verschiedenen VHDL-Dateien generiert und kompiliert sind wird während der Synthese eine Netzliste erstellt. Diese wird dann für die entsprechende FPGA implementiert. Danach wird eine Programmierdate (.bit) generiert und letztendlich wird die Schaltung in der FPGA konfiguriert.

3.5.4.1 Synthese

Für die Synthese wird die Anwendung Xilinx Synthesis Technology (XST) verwendet der erwähnten Software. Während der Synthese wird aus dem kompilierten VHDL-Code die Netzliste (NGD-Datei) erstellt. Die Netzliste enthält logische Daten, wie auch die Design-Daten. Dies bedeutet, dass in der Netzliste die gesamte Beschreibung der Schaltung der FPGA vorhanden ist.

Für die Synthese werden die verschiedenen VHDL-Dateien und eine Datei mit Bedingungen (User Constraint File, UCF) benötigt. Die UCF-Datei enthält in diesem Projekt lediglich die Beschreibung der Ein- und Ausgänge der FPGA.

Die Datei ist im Anhang *14.5.12 UCF-Datei* ersichtlich.

Während der Synthese wird der Nutzer auf Fehler oder Unstimmigkeiten in den verwendeten Dateien aufmerksam gemacht. Bei der Synthese dieser Schaltung erscheinen Warnungen, welche besagen dass einzelne Teile der Schaltung weggelassen werden. Dies ist auch korrekt so, da, je nach Wert der generischen Parameter nbOfBoards und nbOfChanPerBoard, gewisse Teile der Schaltung wegfallen.

3.5.4.2 Implementierung

Die durch die Synthese erstellte Netzliste wird dann für die eingestellte FPGA implementiert. Während der Implementierung wird die Schaltung auf die vorhandenen Blöcke in der FPGA angepasst. Dieser Vorgang wird Mapping genannt. Danach wird die Platzierung der Schaltung auf die Blöcke in der FPGA durchgeführt. Dies nennt sich Placing. Zuletzt werden, während dem Routing, die nötigen Verbindungen zwischen den Blöcken in der FPGA erstellt.

Nach der Implementierung kann die Schaltung welche schlussendlich in die FPGA konfiguriert wird betrachtet werden. Zudem kann eine Zusammenfassung über die Verwendung der FPGA betrachtet werden.

Die Abbildung 3-37 zeigt wie viel Kapazität der FPGA für eine Synthese für 16 Kanäle, also für die maximale Anzahl an Kanälen, verwendet wird.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	2,324	4,704	49%	
Number of 4 input LUTs	2,390	4,704	50%	
Logic Distribution				
Number of occupied Slices	1,854	2,352	78%	
Number of Slices containing only related logic	1,854	1,854	100%	
Number of Slices containing unrelated logic	0	1,854	0%	
Total Number of 4 input LUTs	2,687	4,704	57%	
Number used as logic	2,390			
Number used as a route-thru	233			
Number used for Dual Port RAMs	64			
Number of bonded IOBs	37	140	26%	
Number of GCLKs	1	4	25%	
Number of GCLKIOBs	1	4	25%	

Abbildung 3-37: Zusammenfassung der Implementierung der FPGA-Schaltung für 16 Kanäle

Aus der Zusammenfassung geht hervor, dass 49% der Flip Flops in den Blöcken genutzt werden. Von den Wahrheitstabellen der Blöcke (LUTs) werden 50% verwendet. Zu sehen ist auch, dass 78% der verfügbaren Blöcke (Slices) besetzt sind. Die Ein- bzw. Ausgänge (IOBs) werden zu 26% verwendet.

Daraus lässt sich schliessen, dass in der FPGA noch Platz vorhanden ist, falls weiter Funktionen erstellt werden sollen.

Während der Implementierung erscheinen wiederum Warnungen, welche besagen, dass manche Signale nicht verbunden werden. Auch dies ist normal, da nicht immer alle definierten Signale verwendet werden.

3.5.4.3 Konfiguration der FPGA

In den letzten beiden Abschnitten des Implementierungsprozesses werden die Daten vom vorherigen Abschnitt übernommen. Aus den Daten wird dann eine Bitstrom-Datei generiert. Hierbei handelt es sich um die Datei, welche effektiv auf die FPGA geladen wird.

Zum Schluss wird die Bitstrom-Datei auf die FPGA geladen und die Schaltung so konfiguriert.

3.5.5 PROM

Da die FPGA keinen Speicher enthält, welcher die Konfiguration behalten kann sobald keine Spannungsversorgung mehr vorhanden ist, wird ein zusätzlicher Speicher auf der FPGA-EBS-Karte verwendet. Dieser Speicher beinhaltet die Konfiguration der FPGA. Sobald die FPGA-EBS-Karte eingeschaltet wird, wird die FPGA von diesem Speicher aus konfiguriert.

Die Speicherung der Daten in dem PROM-Speicher erfordert jedoch eine spezielle Datei. Die gesamte Implementierung der Schaltung wird auf dieselbe Art und Weise durchgeführt. Zum Schluss wird allerdings nicht direkt auf die FPGA geschrieben sondern eine PROM-Datei erstellt, welche den originalen Bitstrom enthält. Diese Datei wird dann auch mit der ISE Software von Xilinx auf den PROM-Speicher geladen.

3.6 USB-Verbindung

Der FTDI-Chip, welcher die Schnittstelle zwischen der FPGA und der USB-Verbindung darstellt ist bereits auf der FPGA-EBS-Karte vorhanden.

3.6.1 USB-Schnittstelle

Bei dieser Schnittstelle handelt es sich um einen IC, der einerseits einen parallelen 8Bit-Bus hat und auf der anderen Seite einen USB. Der integrierte Schaltkreis FT245BM wird von FTDI Chip hergestellt.

Von der FPGA her gesehen handelt es sich um einen FIFO-Speicher welcher über einen Datenbus geschrieben und gelesen wird.

Die USB-Schnittstelle erscheint auf dem PC als virtueller COM-Port[10]. So kann sie sehr einfach ausgelesen und gehandhabt werden. Es handelt sich dabei um eine serielle Schnittstelle welche die folgende Konfiguration aufweist. Die Konfiguration der seriellen Schnittstelle ist in der Tabelle 3-3 zusammengefast.

Parameter	**Wert**
Datenbits	8
BaudRate	115.2kBd/s
Sartbit	1
Stopbit	1
Parity	-
Handshake	-

Tabelle 3-3: Konfiguration der seriellen Schnittstelle (COM-Port)

3.6.2 USB-Treiber

Damit die USB-Verbindung seitens des PCs funktioniert muss erst ein Treiber installiert werden. Dabei handelt es sich um einen Treiber, welcher die USB-Verbindung in eben diesen virtuellen COM-Port umwandelt.

Der Treiber trägt den Namen Virtual COM Port Driver. Bei diesem Projekt wurde die Version 2.06.00 verwendet. Der Treiber kann unter der folgenden Internetadresse gefunden werden.

http://www.ftdichip.com/Drivers/VCP.htm

Es kann vorkommen, dass der FTDI Chip trotz des installierten Treibers falsch erkannt wird. Im Geräte Manager erscheint dann eine Maus anstelle des Virtual Com Ports. Hier hilft es das USB-Kabel zu entfernen und wider anzuschliessen.

[10] COM-Port: Eine standardisierte, asynchrone, serielle Schnittstelle welche in Windows COM-Port genannt wird (entspricht der RS-232 bzw. EIA-232 Norm)

3.6.3 USB-Protokoll

Damit die Kommunikation zwischen dem PC und der FPGA geregelt ist wird ein Protokoll definiert.

Allgemein wird ein sehr einfaches Protokoll verwendet. Für beide Datenrichtungen wird die Paketform der Abbildung 3-38 verwendet.

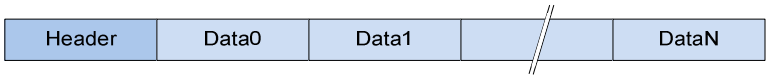


Abbildung 3-38: Paketform des Protokolls zwischen FPGA und PC

Bei allen Daten wird zuerst ein Header versendet. Dieser bestimmt über den Zweck der Daten. Danach folgen die eigentlichen Datenbytes.

Für die Konfiguration der Messkette werden die Typen der Tabelle 3-4 definiert.

ID (Header-Wert)	Datengrösse	Richtung	Parameters
$0X_h$	4 Bytes	FPGA PC	Messdaten des Kanals X
10_h	2 Bytes	PC FPGA	Anzahl der Werte welche für die Mittelung verwendet werden sollen (nbOfValsToSum)
20_h	1 Byte	PC FPGA	Konfiguration der Datenrate der A/D-Wandler
21_h	1 Byte	PC FPGA	Konfiguration der Filter der A/D-Wandler
22_h	2 Bytes	PC FPGA	Konfiguration der Grenzfrequenz des Hochpassfilter der A/D-Wandler
23_h	3 Bytes	PC FPGA	Offset-Kalibrierung der A/D-Wandler
24_h	3 Bytes	PC FPGA	Vollausschlags-Kalibrierung der A/D-Wandler

Tabelle 3-4: Pakettypen zur Konfiguration der Messkette

Wie bereits erwähnt ist die Messkette zurzeit nicht konfigurierbar. D.h. weder die A/D-Wandler noch die Schaltung in der FPGA kann während dem Betrieb konfiguriert werden. Lediglich eine Konfiguration mit Hilfe der generischen Parameter der FPGA-Konfiguration ist möglich.

4. Herstellung der Hardware

Die FPGA-Karte und der PC waren schon vorhanden und mussten nicht mehr hergestellt werden. Damit aber die gesamte Messkette montiert war mussten diese Komponenten auch verbaut werden. Von Grund auf hergestellt wurde lediglich die Erweiterungskarte.

In den folgenden Kapiteln soll erklärt werden wie die Erweiterungskarte hergestellt wird, wie der Aufbau genau aussieht und welche Form das fertig montierte System hat.

4.1 Mezzanine-Erweiterungskarte

Diese Kapitel sollen das Mezzanine-Prinzip und die Herstellung einer Erweiterungskarte zeigen. Jede weitere Erweiterungskarte wird auf dieselbe Art und Weise hergestellt.

4.1.1 Mezzanine-Prinzip

Das Mezzanine-Prinzip erlaubt die Stapelung der Erweiterungskarte. Wie bereits vorher erwähnt wird so nur eine FPGA-EBS-Karte zum Betrieb von bis zu vier Erweiterungskarten benötigt. Dieses Stapelprinzip wird bei den FPGA-Basiskarten wie auch bei den ARM-Basiskarten angewendet.

Die Abbildung 4-1 Zeigt wie die einzelnen Elemente gestapelt werden.

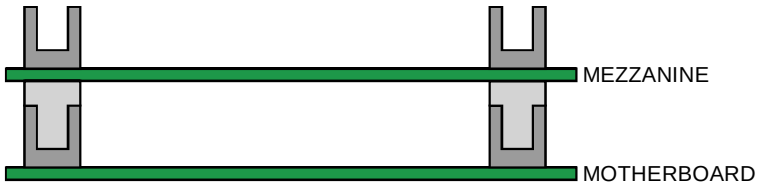


Abbildung 4-1: Mezzanine Stapel ([4] figure 11: mezzanine stacking)

Weiter Informationen zu diesem Aufbau können auf den Servern der Hes-so Valais / Wallis Sion unter *P:/PCB/Produit/Logical/FPGA-EBS/doc* gefunden werden. Mit Hilfe der Datei fpga-ebs_board.doc [4] und dem Schema der Basiskarte konnte dieses Prinzip in das Projekt eingebunden werden. Das Schema der Basiskarte befindet sich unter *P:/PCB/Produit/Logical/FPGA-EBS/FPGA_EBS_V1.1* und heisst *FPGA_EBS.sch*.

4.1.2 Layout

Da eine möglichst störungsunempfindliche Messung durchgeführt werden soll wird die Erweiterungskarte auch unter diesem Gesichtspunkt erstellt. Damit die Signale sauber von der Masse (GNDs) und von der Spannungsversorgung (V_{CC}) getrennt werden können wird urde ein PCB[11] mit vier Schichten (Layer) erstellt.

Folgend werden die einzelnen Layouts[12] gezeigt und erklärt, warum diese so gestaltet sind.

[11] PCB (Printed Circuit Board): Leiterplatte

[12] Layout: (auch Design) die Anordnung der Bahnen, Kupferflächen und Komponenten auf der Leiterplatte

Der Querschnitt des PCBs ist in Abbildung 4-2 dargestellt.

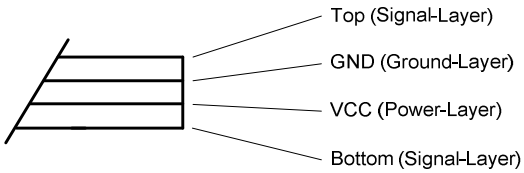


Abbildung 4-2: Querschnitt der Leiterplatte

Die Signal-Schichten befinden sich oben und unten. Da werden bei der Montage auch die Komponenten angebracht. So ist es möglich eine Masse-Schicht (Ground-Layer) und eine V_{CC}-Schicht ohne Unterbrüche zu erstellen wodurch Probleme vermieden werden können.

Die Abbildung 4-3 zeigt die Signal-Schicht. Dieses Layout wird auf der oberen wie auch auf der unteren Signal-Schicht (Top-Layer und Bottom-Layer) verwendet.

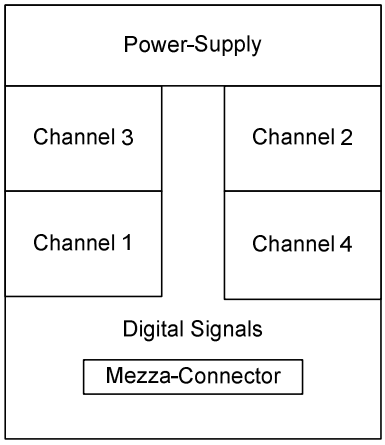


Abbildung 4-3: Layout der Signal-Schichten

Diese klare Untertrennung in Signalbereiche wird vorgenommen, damit keine gegenseitige Beeinflussung zu Stande kommt. So enthalten die Bereiche Power-Supply, welches die Einspeisung ist und die Kanäle 1, 2, 3 und 4 analoge Signale wobei der Bereich welcher mit Digital Signals gekennzeichnet ist naheliegender Weise die digitalen Signale enthält und auf die Mezzanine-Verbindung führt.

Abbildung 4-4 zeigt den VCC-Layer. Bei dieser Schicht wird aus bereits erwähnten Gründen auf eine Trennung des analogen und digitalen Bereichs geachtet.

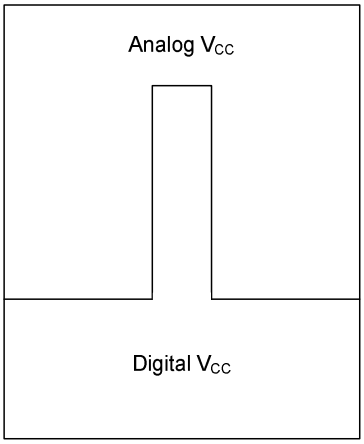


Abbildung 4-4: Layout der V_{CC}-Schicht

Beim analogen V_{CC} handelt es sich um die erstellten 5V und beim digitalen um die von der Mezzanine-Verbindung erhaltenen 3.3V.

Die in der Abbildung 4-5 dargestellte Masse-Schicht (Ground-Layer) wird nach demselben Prinzip wie die Signal-Schicht gestaltet.

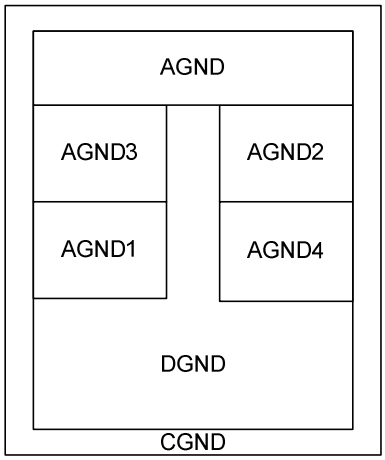


Abbildung 4-5: Layout der Masse-Schicht

Die einzelnen Kanäle wie die Einspeisung und der digitale Teil erhalten einen eigenen Bereich, welcher natürlich jeweils unter dem entsprechenden Signal- und V_{CC}-Bereich liegt. Dies wären der AGND für die Einspeisung, der DGND für den digitalen Bereich und AGNDn für den Kanal n, wobei A jeweils für analog und D für digital steht. Des Weiteren kann der CGND gesehen werden. Dies ist der Chassis-Ground und wird mit dem Gehäuse in welchem sich die Leiterplatte befindet verbunden.

Da die einzelnen Massen alle mit der Einspeisung verbunden sein müssen werde diese durch Brücken kurzgeschlossen. Diese Brücken sind in der Form von einem 0Ω-Widerstand auf dem Signal-Layer vorhanden. Die Abbildung 4-6 zeigt am Beispiel der Brücke zwischen dem DGND und dem AGND wie dies umgesetzt wird.

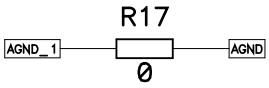


Abbildung 4-6: Massen-Brücke zwischen der Masse des Kanals 1 und der Masse der Einspeisung

Solche Massen-Brücken werden gemacht damit die einzelnen Massen alle an einem Punkt zusammengeführt werden. Dieser Punkt liegt unmittelbar neben der der Einspeisung, da dort die wirkliche Erdung am nächsten ist.

Nebst der Aufteilung in verschieden Bereiche gibt es noch andere Bedingungen, welche beim Erstellen des Layouts beachtet werden mussten. Die folgende Aufzählung fasst die wichtigsten zusammen.

- Die einzelnen Komponenten welche eine gemeinsame Funktion ausführen sollen gruppiert werden, damit die Signalleitungen so kurz wie möglich gehalten werden können und Einflüsse von aussen minimiert werden können.
- Entkopplungskondensatoren und Filter sollten immer so nahe wie möglich an dem entsprechenden Eingang der Schaltung bzw. des IC positioniert werden.
- Um die Referenzspannung und um die Operationsverstärker soll ein Massen-Ring gelegt werden. Dieser Ring soll das Rauschverhalten optimieren. Allerdings kann hierbei auch ein einfacher Massenplan verwendet werden.

Die Thematik mit dem Massen-Ring wird noch etwas ausgeführt. Um dessen Einfluss testen zu können sind zwei Kanäle mit einem solchen ausgestattet. Ein anderer Kanal ist mit einem

zweiten Massenplan anstelle eines Rings ausgestattet. Und beim letzten Kanal wird in diesem Bezug gar keine Vorkehrung getroffen.

Die Abbildung 4-7 zeigt einen solchen Massen-Ring.

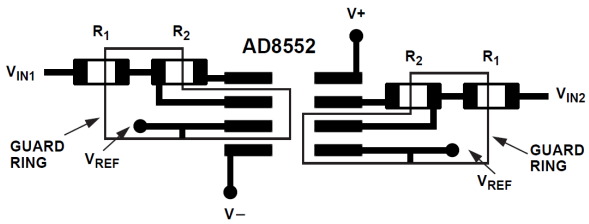


Abbildung 4-7: Massen-Ring um den Operationsverstärker AD8552

Welche Variante die beste ist und wie stark der jeweilige Einfluss ist wird unter *6.3.1.2 Präzision* evaluiert.

Das endgültige Layout der PCBs wurde von den Mitarbeitern der Elektronik-Werkstatt AE01 der Hes-so Wallis Sion erstellt.

Die Abbildung 4-8 zeigt das hergestellte PCB. Die PCBs wurden von einer spezialisierten Firma namens Euro Circuits hergestellt, da PCBs mit mehr als zwei Schichten in der Hochschule nicht hergestellt werden können.

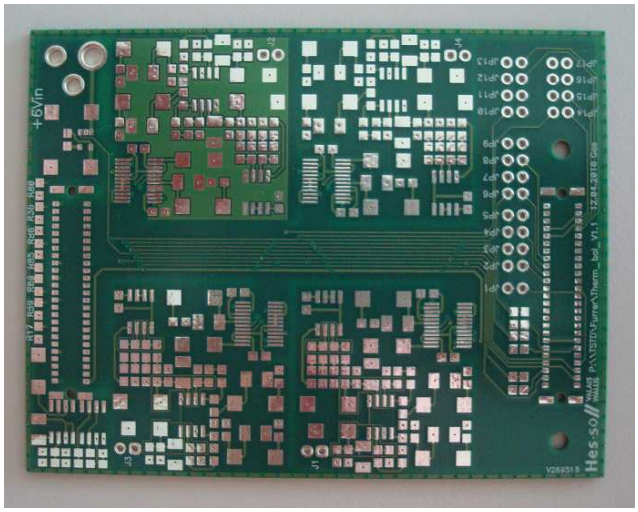


Abbildung 4-8: nicht bestückte Erweiterungskarte

Bei den vier Kanälen ist ein Unterschied zu sehen. Dabei handelt es sich um die soeben beschriebenen Massnahmen zur Störungsminderung. Im oberen linken Kanal ist ein Masse-Plan auf der Signalschicht vorhanden. Bei den beiden unteren Kanälen wurde ein Massen-Ring angebracht und beim Kanal oben rechts ist diesbezüglich, wie bereits erwähnt, nichts unternommen worden.

Die Layouts der verschiedenen Schichten in der endgültigen Form sind im Anhang unter *14.6.1 Layout* zu finden. Das Layout kann auch auf der beigelegten CD unter *pcad* und auf dem Server der Hes-so Valais / Wallis Sion unter *P:\PCB\Students\TSTD\Furrer D\Manufactured* gefunden werden.

Die hergestellten PCBs entsprechen der Version 1.1_2 des Schemas.

4.1.3 Montage

Die Komponenten wurden bei Farnell und Distrelec bestellt. Eine Komponentenliste ist im Anhang *14.6.2 Komponentenliste* ersichtlich.

Die Montage wurde im Atelier AE08 PCB-SMD der Hes-so Valais / Wallis Sion durchgeführt. Wie das PCB nach der Montage ausgesehen hat zeigt die Abbildung 4-9.

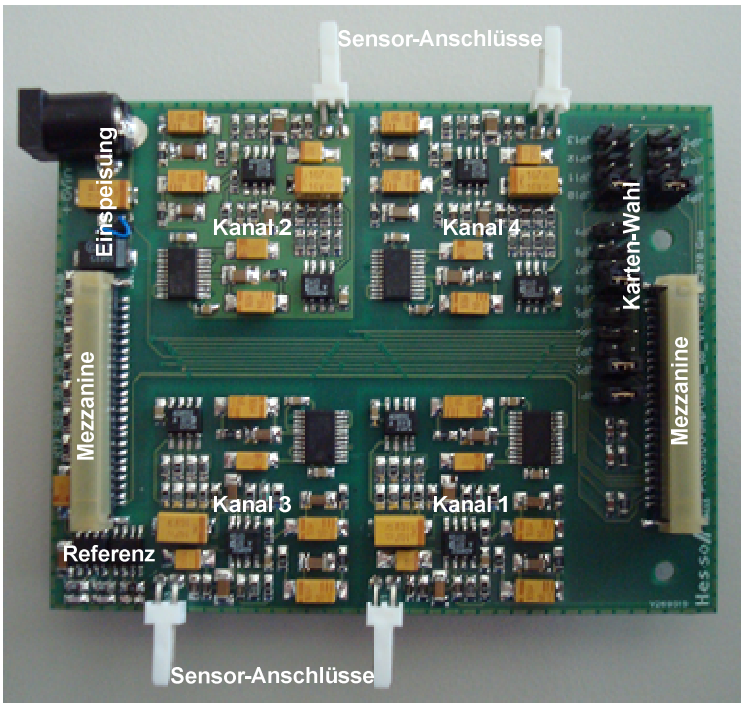


Abbildung 4-9: bestückte Erweiterungskarte

Die einzelnen Teile der Erweiterungskarte sind angeschrieben.

4.2 Gesamtes System

Die fertig bestückte Erweiterungskarte wird auf die FPGA-Basis-Karte gesteckt. Das Resultat ist in Abbildung 4-10 dargestellt.

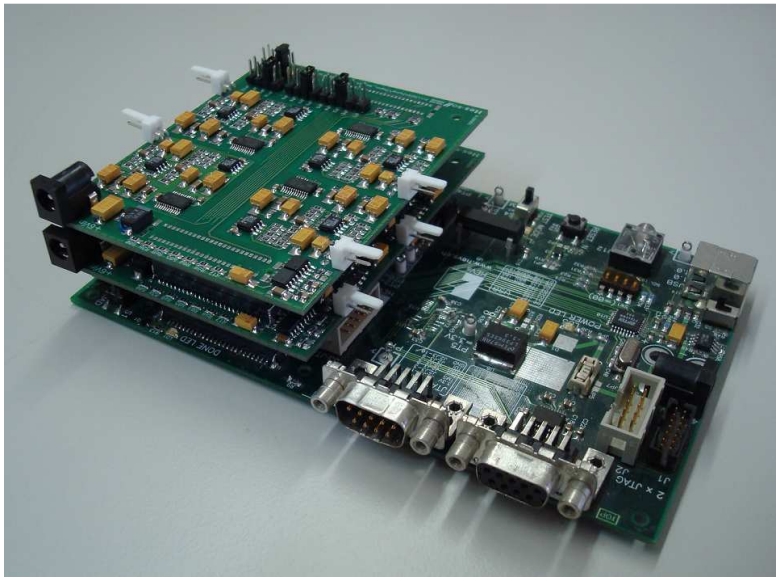


Abbildung 4-10: Die zusammengesteckte Hardware

Ersichtlich sind die FPGA-EBS-Basiskarte und zwei Erweiterungskarten.

Die zeigt die Hardware in einem Gehäuse. Es handelt sich dabei um ein Kunststoffgehäuse. Dieses soll die Hardware vor Wasserspritzer oder ähnlichem schützen. Ein Metallgehäuse würde die Hardware zusätzlich vor elektromagnetischen Einflüssen schützen.

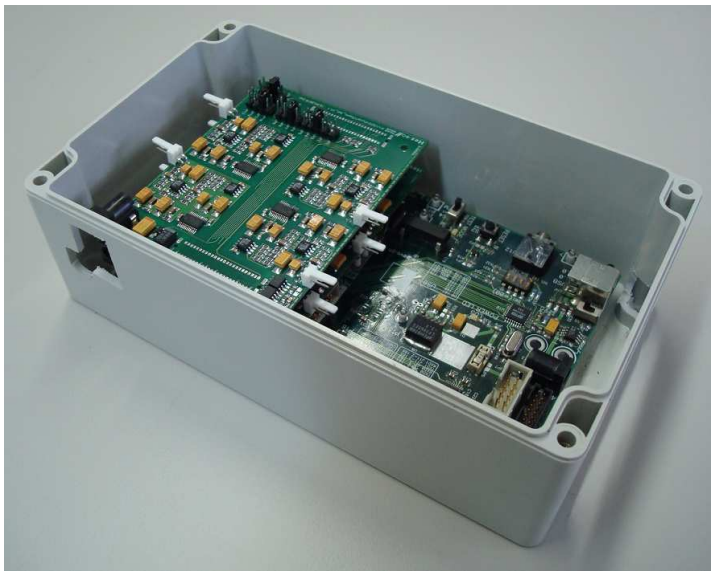


Abbildung 4-11: Hardware im Gehäuse

4.3 Stabsonde

Damit die Sensoren nicht nur ausserhalb des Reaktors, an welchem die Messung installiert wird, angebracht werden können wurde eine Stabsonde hergestellt. Die Stabsonde besteht aus 8 Teilen und wird von oben in den Bioreaktor eingeführt.

Die Stabsonde besteht aus zwei Materialien. Einerseits aus Stahl, welcher die Temperatur an die Sonden leitet und andererseits Kunststoff, welcher die Stahlteile im Bezug auf die Temperatur untertrennen soll, damit die Temperatur in Schichten gemessen werden kann.

Fixiert wird die Stabsonde wie die üblichen Elemente, welche von oben in die Fermentationsanlage eingeführt werden.

Die Abbildung 4-12 zeigt die Sonde.

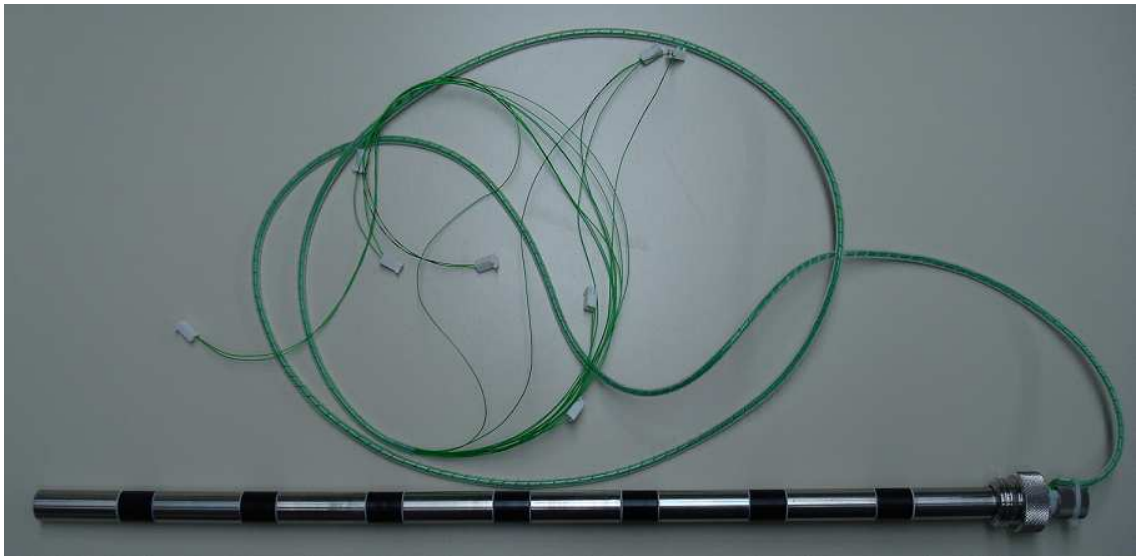


Abbildung 4-12: Stabsonde mit Verbindungskabel

Auf der Abbildung sind die einzelnen Elemente der Sonde ersichtlich. Auch das Verbindungskabel zur Erweiterungskarte mit den Steckern ist ersichtlich. Zwischen den einzelnen Elementen der Sonde befinden sich weisse Ringe, welche die Verschlüsse dichten sollen, da sich die Sonde in der Flüssigkeit befinden wird.

Die Stabsonde besteht aus 8 Metallelementen, was bedeutet dass sich 8 PT1000-Widerstandsthermometer in der Sonde befinden. So kann der Innenraum des Reaktors in 8 Bereiche unterteilt werden. Die Stecker sind entsprechend der Position des Sensors nummeriert. Die Sonde 1 befindet sich ganz unten und die Sonde 8 zuoberst.

Hergestellt wurde die Stabsonde von der Mechanik-Abteilung und vom Elektronik-Atelier AE01 der Hes-so Valais / Wallis Sion.

5. Entwicklung der Software

Die Software, welche die Daten über USB entgegen nimmt, wurde mit Matlab erstellt. Dabei handelt es sich um eine grafische Benutzeroberfläche (Graphical User Interface, GUI).

Die Software nimmt die Daten über USB entgegen und speichert diese ab. Danach können die Daten im zeitlichen Verlauf angezeigt werden. Die Anzeige des Leistungsdichtespektrums ist auch möglich. Zudem können die Daten gebrauchsfertig als Matlab-Dateien abgespeichert werden. Ein Medianfilter ermöglicht es Spitzen, welche auf externen Einflüssen beruhen, herauszufiltern.

Die Software ist derzeit für 8 Kanäle konzipiert. Dies kann zu einem späteren Zeitpunkt allerdings problemlos erweitert werden.

Folgend werden die einzelnen Teile der Software erklärt. Der kommentierte Code befindet sich im Anhang und wird jeweils erwähnt.

Für die Entwicklung der Software wurden einerseits alte Projekte der Hes-so Valais / Wallis Sion und andererseits die Webseite des Herstellers von Matlab [5] genutzt.

In Matlab wird in einer proprietären Sprache programmiert. Das Programm besteht aus einzelnen Teilen welche Funktionen oder Skripts genannt werden. Die Skriptsprache von Matlab ermöglicht ein effizientes Entwickeln der Software.

Die gesamte entwickelte Software, wie auch die zugezogenen Codes, befinden sich auf der beigelegten CD im Verzeichnis *matlab*.

5.1 Serielle Schnittstelle

Von Matlab her gesehen handelt es sich bei der USB Verbindung um einen virtuellen seriellen Port. Wie bereits erwähnt wird dies durch den Treiber von FTDI-Chip bewerkstelligt.

Mit der von Matlab zur Verfügung gestellten Funktion `serial()`, kann ein serieller Port erstellt werden.

Danach werden die Parameter der Verbindung wie unter *3.6.1 USB-Schnittstelle* beschrieben gesetzt. Nebst diesen Parametern werden noch Programmspezifische Parameter eingestellt. Hierzu gehört die Festlegung der Rückruf-Funktion (Callback-Funktion) welche aufgerufen wird, sobald gewisse Bedingungen erfüllt sind oder gewisse Programmteile aktiviert werden. Zu den Bedingungen gehört der Empfang von 16 Bytes oder dem abwarten von einer Sekunde. Die Callback-Funktion die aufgerufen wird heisst `tempArrayLogData()`.

Diese Funktion nimmt die Daten entgegen und speichert diese direkt in eine vorher festgelegte Datei ab. Der Code dieser Funktion befindet sich im Anhang *14.7.2 Funktion tempArrayLogData*.

Die Handhabung der Verbindung (Öffnen, Schliessen usw.) wird im GUI durchgeführt.

5.2 Grafische Benutzerobefläche

Die GUI wurde mit einer Entwicklungsumgebung für grafische Benutzeroberflächen von (GUI Development Environment, GUIDE) von Matlab erstellt. GUIDE ermöglicht die grafische Platzierung der Elemente und generiert automatisch den Code, welcher die GUI betreibt.

Der Code der GUI musste mit den erwünschten Funktionen ergänzt werden, damit auch die anwendungsspezifischen Operationen ausgeführt werden.

Die Abbildung 5-1 zeigt die erstellte GUI mit einem Beispiel einer Messung.

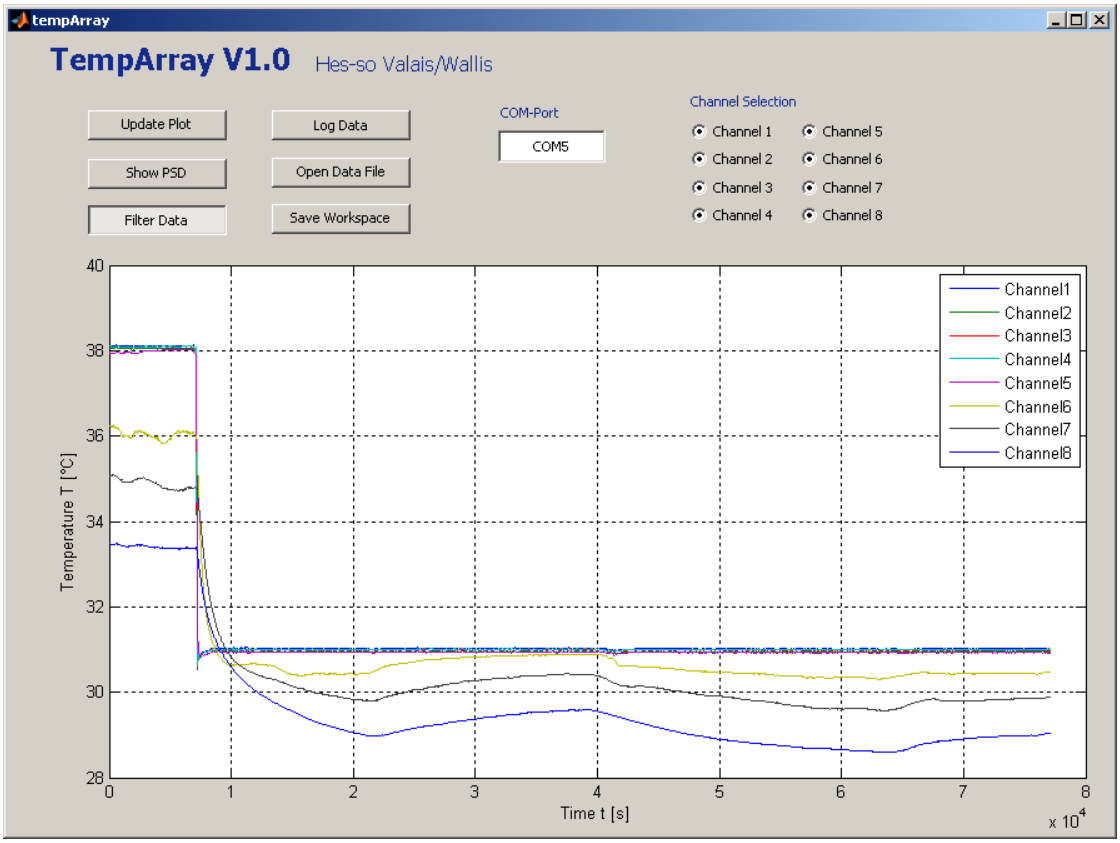


Abbildung 5-1: GUI des Matlab-Programms

Die möglichen Operationen sind das Aufzeichnen einer Messung (Log Data), das Zeichnen des Graphen (Update Plot), das Zeichnen der Leistungsdichtespektren (Show PSD), die Filterung der Daten (Filter Data), das Öffnen einer früheren Messung (Open Data File), und das speichern der gemessenen Temperatur und Zeit (Save Workspace). Des Weiteren können die Kanäle, welche auf dem Graphen erscheinen sollen ausgewählt, werden. Die Wahl des COM-Ports kann auch getroffen werden.

Der generierte Code der GUI befindet sich im Anhang *14.7.1 Code der grafischen Benutzeroberfläche*. In den folgenden Kapiteln werden die einzelnen Operationen erklärt.

5.3 Operationen

Der Code der Operationen ist in den Code der GUI integriert und kann im selben Anhang gefunden werden. Welche Funktion welcher Operation entspricht wird jeweils erwähnt.

5.3.1 Daten Aufzeichnen (Log Data)

Die Daten können mit der Betätigung dieser Taste aufgezeichnet werden. Die Aktivierung der Taste führt zum Aufruf der Callback-Funktion `togglebutton_log_data_Callback()`. Diese Funktion initialisiert die serielle Verbindung wie weiter oben beschrieben. Danach wird der Benutzer mit Hilfe der Matlab-Funktion `uiputfile()` aufgefordert eine Datei und einen Pfad festzulegen, in welcher die Daten gespeichert werden sollen. Bei der festzulegenden Datei handelt es sich um eine Datei, welche die empfangenden Daten roh enthält. Das bedeutet, dass es sich um eine Bit-Datei mit der Endung .bit handelt.

Nachdem die Datei festgelegt wurde wird ein Flag[13] gesetzt, welches der Funktion `tempArrayLogData()` erlaubt die Daten zu schreiben.

Bei der Taste handelt es sich hierbei um eine Taste nach dem Prinzip eines Umschalters. Das bedeutet, solange die Taste aktiviert bleibt werden die Daten aufgezeichnet. Sobald die Taste deaktiviert wird, wird die Aufzeichnung der Daten beendet.

Achtung
Bei der Wahl der Datei in welche die Daten geschrieben werden sollen muss aufgepasst werden. Es wird stets eine neue Datei erstellt, was bedeutet dass bei einer neuen Messung in dieselbe Datei die alten Daten überschrieben werden.

5.3.2 Daten Anzeigen (Update Plot)

Bei der Betätigung dieser Taste wird der Graph der GUI gezeichnet, falls Daten vorhanden sind. Die Callback-Funktion `pushbutton_update_plot_Callback()` wird aufgerufen. Diese lässt die Daten der binären Datei von der Funktion `tempArrayCalcData()` auswerten und zeigt diese dann an. Zu den Aufgaben dieser Funktion gehört das Zeichnen der einzelnen Kurven und das erstellen der Legende.

Achtung
Diese Operation sollte beim der Daten-Acquisition nicht, oder nur zu Beginn durchgeführt werden. Ein PC mit limitierter Rechenleistung kann, sobald die beiden Aufgaben gleichzeitig ausgeführt werden, überfordert sein.

5.3.3 Daten Laden (Open Data File)

Die Operation ermöglicht es dem Benutzer eine früher aufgezeichnete Messung zu öffnen und erneut auszuwerten. Die binäre Datei mit den aufgezeichneten Daten wird dazu geöffnet. Die entsprechende Callback-Funktion `pushbutton_open_Callback()` hat lediglich die Aufgabe eine Datei zu erhalten. Hierzu wird die Matlab-Funktion `uigetfile()` verwendet. Diese öffnet die Abfrage einer Datei.

Achtung
Diese Funktion unterstützt derzeit nur da Öffnen von binären Dateien. Aus dem Grund sollten diese auch gespeichert werden. Die gespeicherten Matlab-Dateien können in Matlab selber importiert und verwendet werden.

5.3.4 Leistungsdichtespektrum Anzeigen (Show PSD)

Das Leistungsdichtespektrum kann mit einem Klick auf diese Taste gezeichnet werden. Für jeden der ausgewählten Kanäle wird der Graph in einem neuen Fenster gezeichnet.

Die aufgerufene Callback-Funktion heisst `pushbutton_show_psd_Callback()` und erstellt lediglich die Graphen und deren Beschriftungen. Hierbei werden auch die Daten, welche zuerst von der Funktion `tempArrayCalcData()` berechnet werden müssen verwendet. Danach wird die Hilfs-Funktion `PlotPSD()` für alle gewählten Kanäle aufgerufen.

Achtung
Auch diese Operation sollte, aus demselben Grund wie oben, während der Daten-Acquisition nicht durchgeführt werden.

[13] Flag: Binäre Variable die als Statusindikator dient

5.3.5 Daten Filtern (Filter Data)

Falls in den Messdaten störende Signal-Spitzen vorhanden sind, welche nur einem Messwerte entsprechen, können diese Herausgefiltert werden. Die Taste Funktioniert wie ein Umschalter. Der Filter kann also aktiviert und deaktiviert werden. Die Filterung der Daten hat einen Einfluss auf die Berechnung der Messwerte mit der Funktion `tempArrayCalcData()`. Das bedeutet, dass bei der Erstellung der Graphen, abhängig von der Tastenposition, die gefilterten oder ungefilterten Daten verwendet werden.

Die Callback-Funktion `togglebutton_filter_Callback()` setzt ein Flag welches die erwähnte Berechnungs-Funktion über die Wahl des Benutzers informiert.

5.3.6 Daten speichern (Save Workspace)

Falls die gemessenen Daten auf andere Art und Weise verwendet werden sollen bietet diese Operation die Zeit- und Temperatur-Daten als Matlab-Datei mit der Endung .mat abzuspeichern.

Die Callback-Funktion `pushbutton_saveWorkspace_Callback()` öffnet die Matlab-Funktion `uiputfile()`. Diese Funktion erfordert vom Benutzer einen Dateinamen und einen Speicherort zum fortfahren.

Die so abgespeicherten Daten können in dieser Software allerdings nicht mehr geöffnet werden. Sie können jedoch in Matlab in einen bestehenden Workspace[14] geladen werden.

5.4 Hilfs-Funktionen

Damit die einzelnen Operationen entlastet werden können und damit bereits bestehende Funktionen bzw. Skripte verwendet werden können wurden einige Hilfs-Funktionen erstellt oder übernommen.

5.4.1 Berechnung der Werte (tempArrayCalcData)

Die Umrechnung der erhaltenen Daten in Temperaturdaten wird mit dieser Funktion durchgeführt. Dies wird benötigt um die erwünschten Graphen erstellen zu können.

Die Funktion `tempArrayCalcData()` öffnet eine festgelegte Datei und liest dann eine Messung nach der anderen aus. Sobald alle Daten ausgelesen sind werden die erhaltenen Werte in Temperaturen umgerechnet.

Die berechneten Temperaturwerte können, wie bereits erwähnt, wahlweise auch gefiltert werden. Diese Filterung wird in dieser Funktion, mit Hilfe einer weiteren Funktion, durchgeführt.

Der Ausschnitt Code 5-1 zeigt die Berechnung der Temperaturwerte ausgehend der Werte des A/D-Wandlers.

```
uAd = adData * coeff;           % Calculate the ADC-Input_voltage
uMes = (uAd / 3) + vRefAcq;     % Calculate the PT1000-voltage
rMes = uMes / iMes;             % Calculate PT1000-resistance
rMes = rMes - rCable;           % Correct cable error
tMes = ((rMes / r0) - 1) * (1 / alpha); % Finaly, the temperature
```

Code 5-1: Berechnung der Temperaturwerte

[14] Workspace: Arbeitsbereich der Matlab-Sitzung, welche u.a. die Daten enthält.

Wie zu sehen ist wird der Weg der Messwerte durch die Hardware rückgängig gemacht. Das heisst, zuerst wird die Spannung am Eingang des A/D-Wandlers berechnet. Danach wird die am PT1000-Widerstandsthermometer gemessene Spannung berechnet. Mit der Konstante des konstanten Messstroms wird dann der PT1000-Widerstand berechnet, womit die Temperatur berechnet werden kann. Der durch das Kabel zur Sonde eingefügte Fehler wird auch noch korrigiert.

Da während der Übertragung der Daten über USB auch Fehler auftreten können wurde Code-Sequenz Code 5-2 erstellt, welche dieses Problem behandelt.

```
% If the data is incorrect, read until the next correct data
% arrives
if id > 7
    while id > 7
        id = fread(file,1,'uint8');
    end;
    val = fread(file,1,'bit24','b');
end;
```

Code 5-2: Korrektur von fehlerhaften Messdaten

Sobald falsche Daten gelesen werden, wird die Datei weitergelesen bis Daten gefunden werden, welche mit einer gewissen Wahrscheinlichkeit korrekt sind. Daten werden als falsch erkannt, falls die Identifikationsnummer (ID), welche dem Kanal von dem die Daten stammen entspricht, die ID des höchsten Kanals übersteigt.

Da aber auch Daten die Werte der ID annehmen können, ist es möglich, dass erneut falsche Daten gelesen werden. Die Wahrscheinlichkeit eines wiederholten Fehlers soll bestimmt werden.

Falls eine falsche ID vorhanden ist verbleiben maximal vier Datenbytes bis zur nächsten ID. Somit hat ist die Möglichkeit nicht die ID sondern einen Teil der Daten auszulesen 4-mal vorhanden.

Daraus resultiert die Wahrscheinlichkeit eines Fehlers während der Korrektur.

$$P_{\mathrm{CorrError}} = \frac{N_{Channels}}{N_{TotalValues}} = \frac{8}{256} = 0.03125 = 3.125\%$$

Bei einer Wiederholung des Fehlers beim nächsten einzulesenden Wert multipliziert sich die Wahrscheinlichkeit wodurch erkannt werden kann, dass sie gegen 0 strebt.

Falls die Wahrscheinlichkeit eines Fehlers während der Übertragung der Daten per USB auch noch in Betracht gezogen werden würde, wäre die Fehlerwahrscheinlichkeit sehr klein.

Der Code der Funktion ist im Anhang *14.7.3 Funktion tempArrayCalcData* ersichtlich.

5.4.2 Zeichnen der Leistungsdichtespektren (PlotPSD)

Die Berechnung und Zeichnung der Leistungsdichtespektren wird mit Hilfe eines Skripts durchgeführt welches bereits bestand und von einem früheren Projekt [3] übernommen wurde. Der Aufruf der Funktion `PlotPSD()` erfordert einen Daten-Vektor und einen Zeit-Vektor.

Die Funktion kann parametriert werden. Hierbei kann vieles eingestellt werden. Unter anderem können die Fensterfunktion, die Anzahl der Segmente und das Überlappen bestimmt werden. Die Fensterfunktion (window) bestimmt die Gewichtung der Frequenzen des Eingangssignals und auch die spektrale Verarbeitung. Zudem ist der spektrale Fehler von der Fensterfunktion abhängig. Die Anzahl der Segmente (segments) gibt die Fenstergrösse an. Die Fenstergrösse besagt, wie viele Frequenzen in die Berechnung eines Energiewertes einfliessen sollen. Das Überlappen (overlay) bestimmt grafisch gesehen, wie

weit ein Energiewert den anderen nächsten überlappen soll. Des Weiteren können die Beschriftungen der Achsen und der Titel gewählt werden.

Die Funktion wird stets mit den Standardwerten verwendet, da diese ein zufriedenstellendes Resultat liefern. Die Standardwerte sind Fensterfunktion Blackman-Harris, 9 Segmente und ein Überlappen von 50%.

Die Funktion wird in der entsprechenden Callback-Funktion `pushbutton_show_psd_Callback()` aufgerufen, nachdem die Werte berechnet wurden.

Die Quelldateien dieser Funktion befinden sich auf der beiliegenden CD im Verzeichnis *matlab*.

5.4.3 Medianfilterung der Daten (fastmedfilt1d)

Falls während einer Messung starke Störungen auftreten, welche nur einer Messung betreffen kann diese, trotz der Mittelwertbildung in der FPGA, zu Verzerrungen des schlussendlich dargestellten Signals führen.

Aus diesem Grund kann wahlweise ein Medianfilter angewendet werden. Dieser Filter filtert die von der Funktion `tempArrayCalcData()` berechneten Temperaturwerte bevor sie angezeigt oder für die Darstellung des Spektrums benutzt werden.

Das folgende Beispiel zeigt die Funktion des Filters.

Das Eingangssignal lautet x = [2 3 76 8]. Der Filter hat eine gewisse Fenstergrösse, die besagt aus wie vielen Werten der Median entnommen werden soll. Standardmässig beträgt diese Grösse 3. Somit ergeben sich die folgenden Ausgangssignale.

y[1] = median(x[1:2]) = median([2 2 3]) = 2 (Die 2 wird wiederholt da der erste Wert)
y[2] = median(x[1:3]) = median([2 3 76]) = 3
y[3] = median(x[2:4]) = median([3 76 8]) = 8
y[4] = median(x[3:4]) = median([76 8 8]) = 8 (Die 8 wird wiederholt da der letzte Wert)

Das Ausgangssingal ist dann y = [2 3 8 8]. Wie zu sehen ist wurde der Extremwert herausgefiltert.

Der Filter wird angewendet indem die Funktion `fastmedfilt1d()` aufgerufen wird. Die Daten welche gefiltert werden sollen werden als Parameter übergeben.

Die Funktion wurde von einer Webseite [6] übernommen. Die Quelldateien können auf der CD im Verzeichnis matlab und auf der entsprechenden Webseite [6] gefunden werden.

6. Tests und Labormessungen

Das in Betrieb genommene System wurde getestet und gemessen.

Zuerst wurden Funktionstests durchgeführt. Diese dienten dazu, die Funktion der erstellten Hardware zu kontrollieren.

Danach wurde Schaltung gemessen. Das bedeutet, dass die einzelnen Signale kontrolliert wurden. Die Einflüsse der einzelnen Komponenten wurden auch untersucht.

Alle hier verwendeten Messdaten befinden sich auf der beigelegten CD im Verzeichnis *measure*.

6.1 Verwendete Geräte

Die zur Durchführung der Tests und Messungen verwendeten Geräte werden folgende aufgelistet.

- Laboreinspeisung: *TTi LTD EL302T S/N: 248460 (Hes-so Inv. Nr. A304/8399.07)*
- Oszilloskop: *Agilent 54622D S/N: MY40005503 (Hes-so Inv. Nr. A309/5025.6)*
- Multimeter: *Multimeter Agilent U1252A S/N: TW46410015 (Hes-so Inv. Nr. A203/8400.16)*

6.2 Funktionstests

6.2.1 Spannungsversorgung und Spannungsreferenz

Bereits während der Montage des Systems wurden die Einspeisung und die Spannungsreferenz kontrolliert. Das bedeutet, zuerst wurden diese beiden Teile der Schaltung bestückt und überprüft.

Spannungsversorgung

Die Spannungsversorgung, welche den unter *3.2.1 Spannungsversorgung* beschrieben 5V-Spannungsregler und dessen Beschaltung umfasst wurde getestet. Hierbei wurde kontrolliert, ob die erzeugte Spannung auch den vorgegeben 5V entspricht.

Die erste Messung ergab, dass dies nicht der Fall ist. Die erzeugte Spannung lag bei ca. 3V und war instabil. Durch eine Kontrolle des Datenblatts wurde klar, dass der verbaute Spannungsregler (LP2985) nicht dem entspricht, welcher im Schema verwendet wurde (LT1761). Durch eine Korrektur der Beschaltung konnte dieser Fehler jedoch behoben werden. Der Bypass-Kondensator muss nicht mit an dem Ausgang des Spannungsreglers, sondern an der Masse angeschlossen werden.

Da dieser Fehler früh entdeckt wurde handelt es sich beim Schema, welches unter *3 Entwicklung der Hardware* erklärt wurde bereits um die korrigierte Version. Im Anhang befinden sich die fehlerhafte Version des Schemas (V1.1_2) und korrigierte Version des Schemas (V1.2). Das PCB entspricht allerdings nicht dem korrigierten Schema. Dies stellt allerdings kein Problem dar, da das PCB durch eine kleine Korrektur angepasst werden kann. Eine andere Möglichkeit zur Korrektur dieses Fehlers wäre die Verwendung des im Schema verwendeten Spannungsreglers.

Nach der Korrektur entsprach die erzeugte Spannung den Vorgaben und betrug 4.9856V.

Spannungsreferenz

Die Referenzspannung wurde mit dem Multimeter auf Genauigkeit und mit dem Oszilloskop auf Stabilität überprüft.

Die Messung der Referenzspannung mit dem Multimeter ergab eine Spannung von 2.4995V, welches der Vorgabe von 2.5V sehr nahe kommt. Eine Betrachtung des Signals mit dem Oszilloskop ergab allerdings, dass die Referenzspannung instabil ist. Instabil bedeutet, dass sich der Gleichspannung eine Wechselspannung mit einer Amplitude von 325mV und einer Frequenz von ca. 107kHz überlagert. Die Abbildung 6-1 zeigt diese Messung.

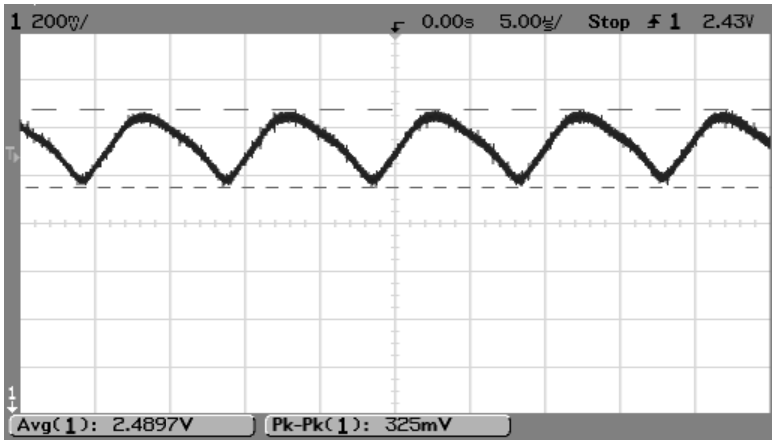


Abbildung 6-1: Messung der Referenzspannung mit dem Oszilloskop

Da dieser Fall bereits bei der Erstellung des Schemas in Betracht gezogen wurde, konnte die Konfiguration der Spannungsreferenz und dem dazugehörigen Impedanzwandler geändert werden. Der Grund für die Schwingung liegt in der kapazitiven last am Ausgang des Impedanzwandlers, welche zu gross ist. Die Konfiguration konnte mit Hilfe der Widerstände R90 und R91, welche unter *3.2.2 Spannungsreferenz* beschrieben wurden, angepasst werden.

Als erstes wurde versucht die Spannungsreferenz ohne Impedanzwandler zu betreiben (R90 entfernt und R91 = 0Ω). Dies führte allerdings auch zu einer instabilen Referenzspannung, wodurch das Problem bestehen blieb.

Durch das Einsetzen eines 10Ω-Widerstandes für R90 und der Entfernung des Widerstandes R90 wurde die Referenzspannung stabil. Der Widerstand R90 dämpft die Schwingungen, welche vorhanden sind. Der Wert der Referenzspannung beträgt durch das Hinzufügen des Widerstands allerdings nicht mehr 2.4995V, sondern 2.4809V. Dies entspricht zwar nicht der Vorgabe, kann im Matlab-Skript allerdings problemlos korrigiert werden.

6.2.2 Erweiterungskarte

Da die Spannungsversorgung und die Spannungsreferenz nun funktionierten konnte ein Kanal bestückt und getestet werden.

Hierbei wurden aller nötigen Komponenten bestückt.

Die Funktion des A/D-Wandlers wurde dann mit Hilfe des Oszilloskops und einem eingefügten Signal kontrolliert. Hier wurde der A/D-Wandler mit Hilfe eines 3V-Signals am Reset-Eingang betrieben. Mit dem Oszilloskop wurden dann die vom A/D-Wandler generierten Signale kontrolliert. Die beiden Takte (Systemtakt und serieller Takt), welche benötigt werden um die A/D-Wandler zu betreiben wurden mit Hilfe von Funktionsgeneratoren erzeugt.

Die Abbildung 6-2 zeigt die in die Schaltung eingefügten Taktsignale, den generierten Impuls und die vom A/D-Wandler gesendeten Daten. Durch Abgleich mit dem Protokoll des

ADS1281, welches im Anhang *14.4.2 ADS1281* eingesehen werden kann, konnte das korrekte Funktionieren bestätigt werden. Danach konnten alle Kanäle bestückt werden.

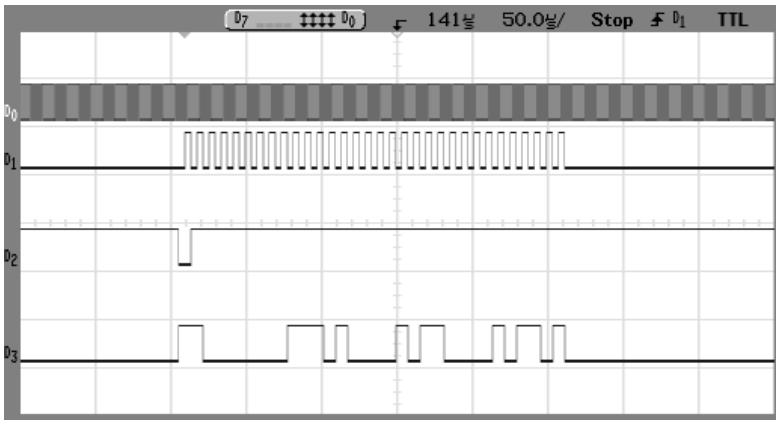


Abbildung 6-2: Takt- und Datensignale eines A/D-Wandlers

D_0 ist der Systemtakt, D_1 der Takt der seriellen Schnittstelle, D_2 der Impuls, welcher neue Daten ankündigt (dataRdy) und D_3 die rohen Messdaten.

6.2.3 FPGA

Die Funktionalität der FPGA wurde bereits anhand der Simulationen überprüft. Um die gesamte Hardware zu überprüfen wurde das System zusammengestellt und gestartet.

Mit Hilfe eines Terminals konnten die über USB empfangen Daten analysiert werden. Die Daten konnten den Vorgaben entsprechend empfangen werden. Die Vorgaben zu den übermittelten Daten wurden unter *3.6 USB-Verbindung* definiert.

Die Abbildung 6-3 zeigt das Terminal mit den empfangenen Daten von einer Erweiterungskarte.

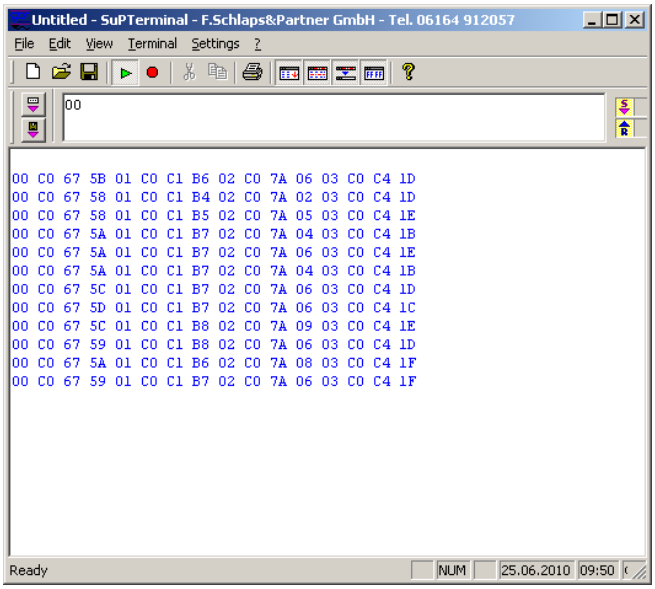


Abbildung 6-3: Empfang der A/D-Wandler-Daten mit Hilfe eines Terminals

Beim verwendeten Terminal handelt es sich um SupTerminal in der Version 1.1.

6.2.4 Software

Nachdem die Hardwarefunktion überprüft worden war, konnte das erstellte Matlab-Programm getestet werden. Da die Erstellung des Programms nach dem Testen der Hardware durchgeführt wurde konnte das Programm bei der Entwicklung stets überprüft und korrigiert werden. Aus dem Grund waren keine finalen Tests nötig.

6.3 Labormessungen

Da die Funktionalität der Hardware kontrolliert und als korrekt empfunden wurde konnten Messungen durchgeführt werden.

Hierzu wurden zuerst Messungen mit Präzisionswiderständen anstelle von PT1000-Widerstandsthermometern durchgeführt. So war der Sollwert stets bekannt und die gemessenen Werte konnten überprüft werden. Zudem sind bei Präzisionswiderständen keine äusseren Einflüsse, welche über die Sensoren in die Messkette eingebracht werden können, vorhanden. Danach wurden auch Messungen mit PT1000-Widerstandsthermometern durchgeführt.

Bei den Messungen musste auch stets beachtet werden, dass die Schaltung schon eine gewisse Zeit eingeschaltet ist, da sich die Elektronik erst aufwärmen muss. Hierbei sollte eine Aufwärmzeit von ca. 45 Minuten genügen. Warum eine solch lange Aufwärmzeit benötigt wird, wird während den Messungen erklärt.

Zur Analyse der Messdaten wurde die erstellte Software verwendet.

Bei der für die Messung verwendeten Hardware handelt es sich um die erste erstellte Erweiterungskarte und die beschriebene FPGA-EBS Karte. Für die Stromversorgung wurde die weiter oben beschriebene Laboreinspeisung verwendet. Alle hier durchgeführten Messungen wurden mit der ersten hergestellten Erweiterungskarte durchgeführt, da die zweite Erweiterungskarte erst in der letzten Woche der Projektarbeit hergestellt wurde.

Die Labormessungen wurden im Raum A304 der Hes-so Valais / Wallis in Sion durchgeführt

6.3.1 Messung mit Präzisionswiderständen

Bei den verwendeten Präzisionswiderständen handelt es sich um 1kΩ Widerstände. Der Temperaturkoeffizient beträgt 10ppm/℃ und die Tole ranz beträgt 0.1%.

Dies bedeutet, dass pro Grad Abweichung von 0℃ der Widerstand um 10ppm, also um 1mΩ abweicht. Die Toleranz von 0.1% besagt, dass der Widerstandswert von 1kΩ um ±1Ω abweichen kann.

Mit den folgenden Messungen wird die Qualität und Präzision der Messkette bestimmt. Der Einfluss einzelner Teile der analogen Schaltung wird auch untersucht.

Bei der Auswertung der Messkette mit Präzisionswiderständen wurden ungefilterte Daten verwendet, da sonst Teile des Rauschens versteckt worden wären.

6.3.1.1 Störabstand

Bedingungen und Methode

Damit der Störabstand der Messkette bestimmt werden konnte wurde eine Messung über einen grösseren Zeitraum durchgeführt.

Die Messung wurde während einer Nacht ausgeführt, damit sie möglichst frei von äusseren Einflüssen, wie Person welches sich im Labor befinden oder Geräten die laufen, ist. Die Messung dauerte 15h20min29s und ergab somit 55‘229 Messwerte.

Eine solch lange Messung wurde durchgeführt, damit für eine korrekte Auswertung genügend Messwerte vorhanden sind.

Die Messwerte werden ungefiltert verwendet, da so der reale Störabstand gefunden werden kann.

Messung

Die Abbildung 6-4, Abbildung 6-5, Abbildung 6-6 und Abbildung 6-7 zeigen das Leistungsdichtespektrum des Spannungsrauschens für die vier Kanäle. Diese Graphen wurden mit Hilfe des unter *5.4.2 Zeichnen* der Leistungsdichtespektren (PlotPSD beschriebenen Matlab-Skripts erstellt.

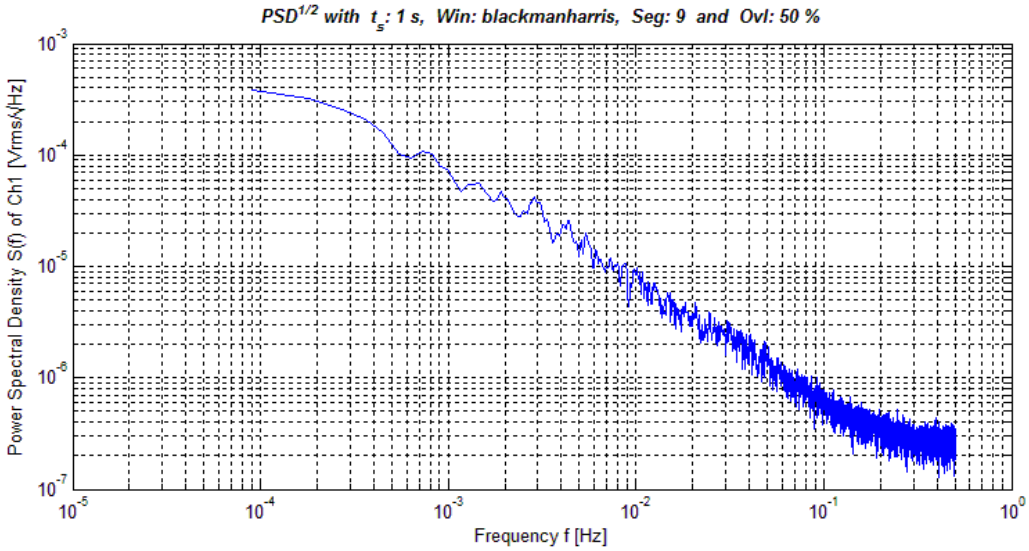


Abbildung 6-4: Leistungsdichtespektrum des Kanals 1 mit Präzisionswiderstand (S(f) [Vrms/√Hz])

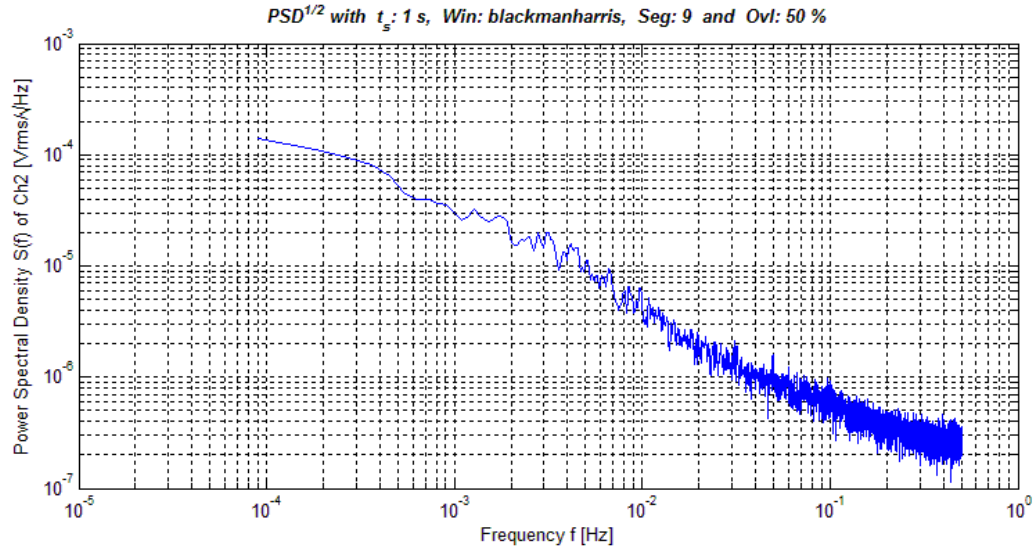


Abbildung 6-5: Leistungsdichtespektrum des Kanals 2 mit Präzisionswiderstand (S(f) [Vrms/√Hz])

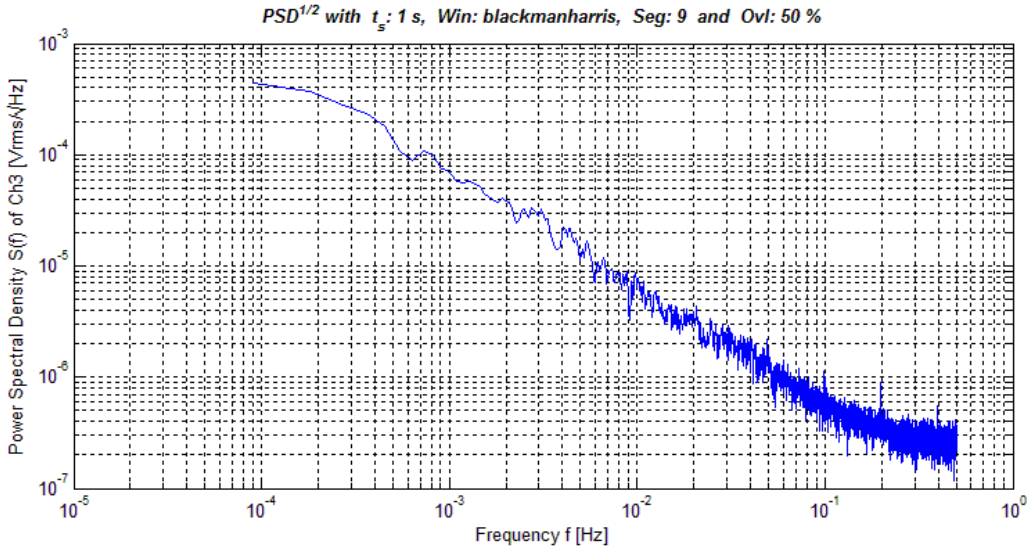


Abbildung 6-6: Leistungsdichtespektrum des Kanals 3 mit Präzisionswiderstand (S(f) [Vrms/√Hz])

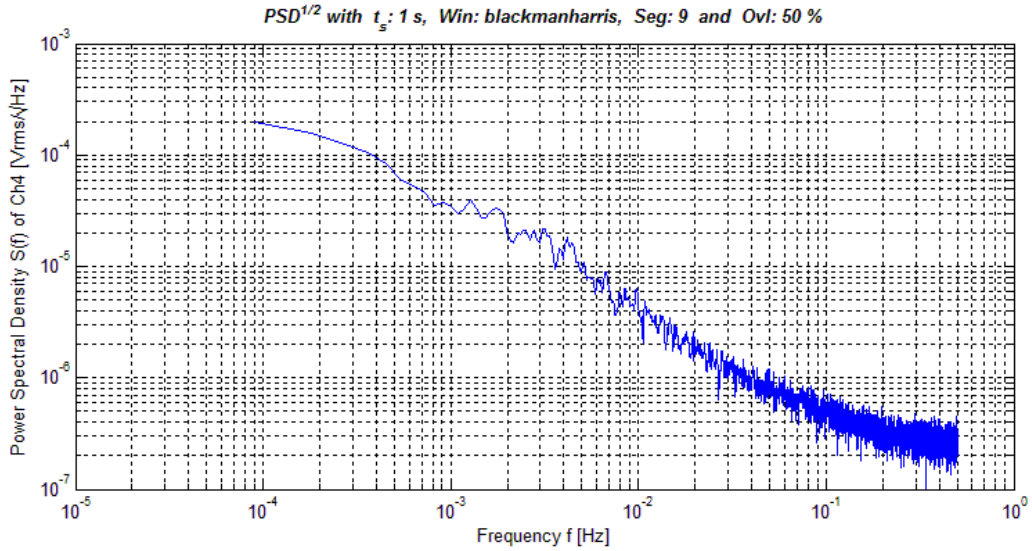


Abbildung 6-7: Leistungsdichtespektrum des Kanals 4 mit Präzisionswiderstand (S(f) [Vrms/√Hz])

Die Signale reichen nur bis zu 0.5Hz da dies der Hälfte der maximalen Frequenz entspricht die durch die Messung mit einer Datenrate von 1SPS erhalten werden kann.

Werte, deren Frequenz kleiner als 1mHz ist, können als Gleichstromsignal betrachtet werden und müssen in die Berechnung der Präzision nicht einbezogen werden.

Für den Kanal 1 kann der Wert von $S_1(f)$ bei $4 \cdot 10^{-6}$Vrms/√Hz festgelegt werden, da dies in etwa dem mittleren Wert des Rauschens entspricht und so einen durchschnittlichen Fall beschreibt.

Für den erwähnten Bereich muss also zuerst der Wert von √Hz berechnet werden.

$$\sqrt{Hz} = \sqrt{f_{max} - f_{min}} = \sqrt{5 \cdot 10^{-1} Hz - 1 \cdot 10^{-3} Hz} = 0.7064\sqrt{Hz}$$

Mit dem Wert von √Hz und dem Wert für $S_1(f)$ kann nun das Eingangsrauschen bestimmt werden.

$$N_{U_{in}1} = \sqrt{Hz} \cdot S_1(f) = 0.7064\sqrt{Hz} \cdot 4 \cdot 10^{-6} \frac{V_{rms}}{\sqrt{Hz}} = 2.826 \cdot 10^{-6} V_{rms}$$

Da die Spannung, welche dem Präzisionswiderstand mit dem Wert 1kΩ entspricht, bekannt ist kann nun der Störabstand berechnet werden.

$$SNR_{dB} = 20 \cdot log\left(\frac{U_{in1}}{N_{U_{in}1}}\right) = 20 \cdot log\left(\frac{0.886V_{rms}}{2.826 \cdot 10^{-6}V_{rms}}\right) = 109.91dB$$

Nach demselben Prinzip ergibt sich für den Kanal 2 ein Störabstand von 115.93dB, wobei für $S_2(f)$ der Wert $2 \cdot 10^{-6}$Vrms/√Hz. Für den Kanal 3 ergibt sich für ein $S_3(f)$ von $4 \cdot 10^{-6}$Vrms/√Hz ein Störabstand von 109.91dB. Der Störabstand des Kanals 4 beträgt 112.41dB bei einem $S_4(f)$ von $3 \cdot 10^{-6}$Vrms/√Hz.

Zu bemerken ist erneut, dass die Messung mit einem festgelegten Wert in einer idealen Laborumgebung durchgeführt wurde. Im Feld wird dieser Wert durch die Sonden und deren Kabel sicher schlechter ausfallen.

Auswertung der Resultate

Die Messungen zeigen, dass der unter *3.3.3.2 Störabstand* definierte Wert für den Störabstand der A/D-Wandler von 124dB beinahe erreicht wird.

Dies bestätigt das Konzept der Schaltung und wird als genügend empfunden. Die Abweichung des Störabstands vom idealen Wert kommt durch die nicht idealen Komponenten auf der analogen Seite des A/D-Wandlers zustande.

6.3.1.2 Präzision

Bedingungen und Methode

Bei dieser Messung wurden dieselben Daten wie bei der Messung des Störabstandes verwendet. Demzufolge gelten auch dieselben Bedingungen für die Messung.

Messung

Die Abbildung 6-8 zeigt die Signale der drei Kanäle im Zeitbereich.

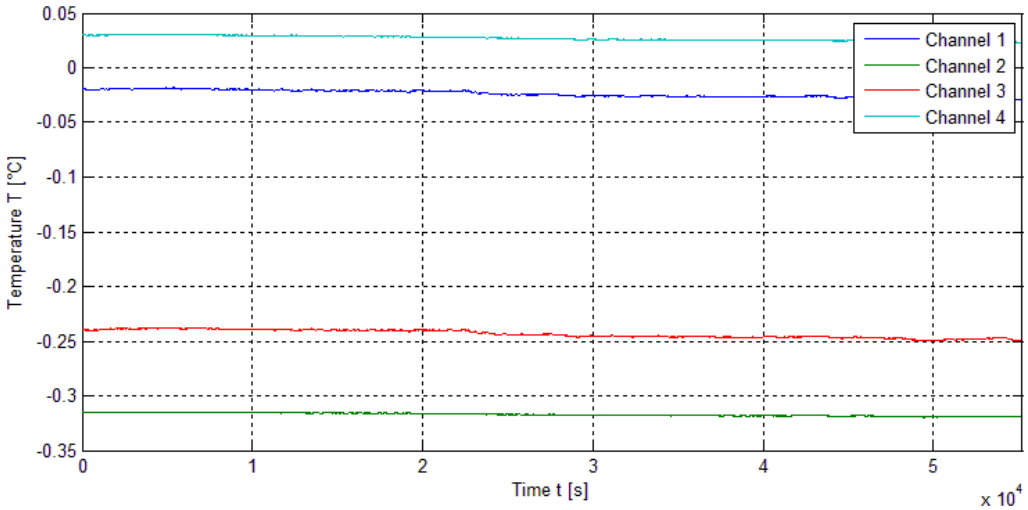


Abbildung 6-8: Messung der Präzision im Zeitbereich (T=f(t) [℃])

Aus der Messung geht hervor, dass die Signale in etwa dem Sollwert 0℃ entsprechen. Jeder Kanal verfügt über ein eigenes Offset.

Die Variation der Signale in der Zeit, welche theoretisch nicht vorhanden sein sollte, ist auf das nicht ideale Verhalten der verbauten Elemente und der Präzisionswiderstände zurückzuführen. Wie weiter oben erwähnt wurde, haben diese auch eine Toleranz und einen Temperaturkoeffizienten. Daher bewirkt die Abkühlung in der Nacht, dass das Offset animmt. Dies lässt darauf schliessen, dass das Offset abhängig von der Temperatur ist.

Da diese Messung keine grosse Auskunft über die Präzision der Messkette gibt wurde das Leistungsdichtespektrum gemessen. Mit dem fixen Widerstandswert am Eingang kann mit Hilfe dieses Spektrums die Genauigkeit bestimmt werden.

Die Abbildung 6-9, Abbildung 6-10, Abbildung 6-11 und Abbildung 6-12 zeigen die Leistungsdichtespektren des Temperaturrauschens der vier Kanäle.

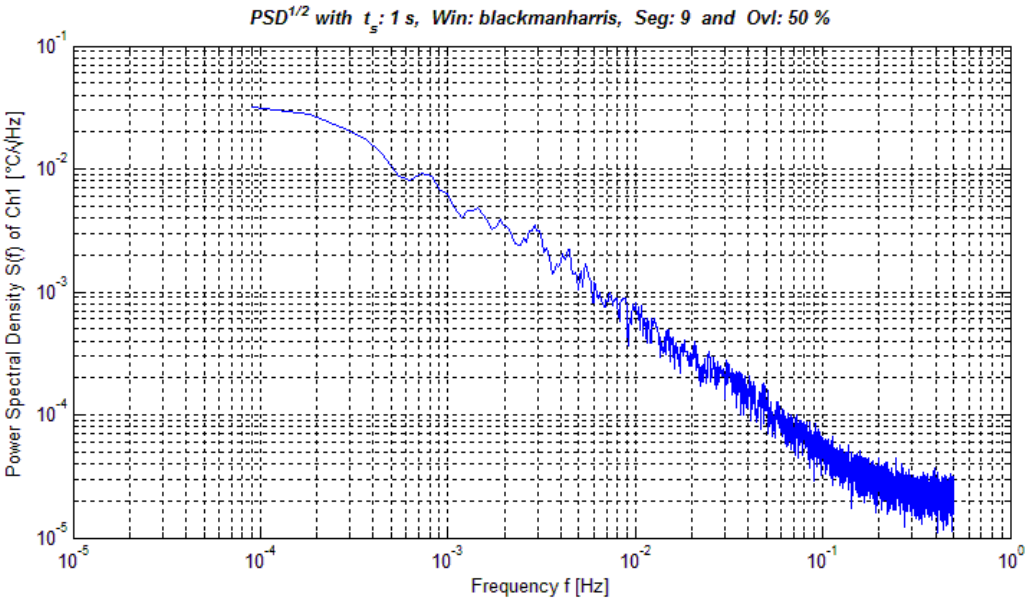


Abbildung 6-9: Leistungsdichtespektrum des Kanals 1 mit Präzisionswiderstand (S(f) [℃/ √Hz])

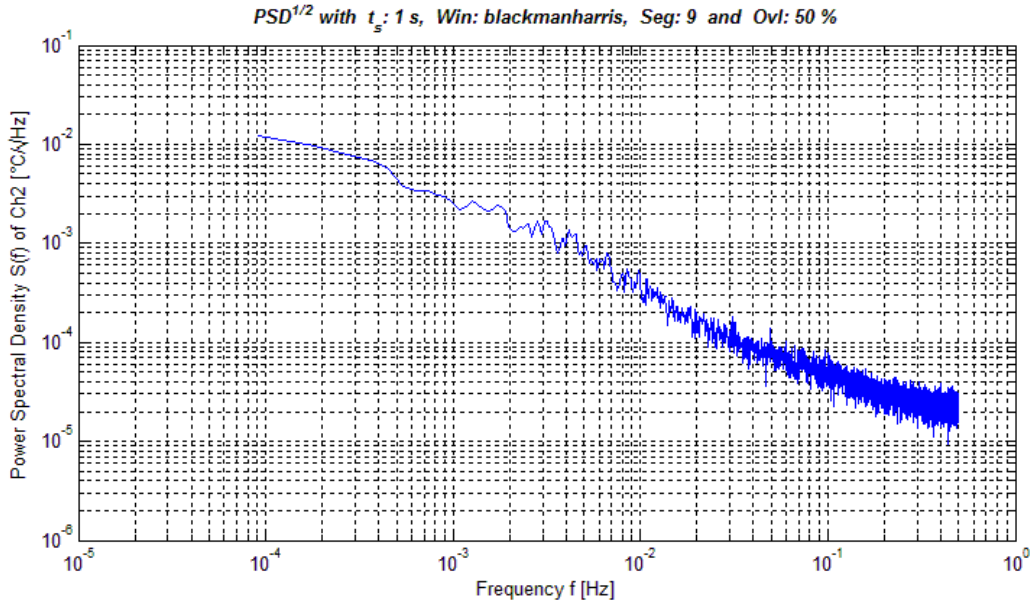


Abbildung 6-10: Leistungsdichtespektrum des Kanals 2 mit Präzisionswiderstand (S(f) [℃/ √Hz])

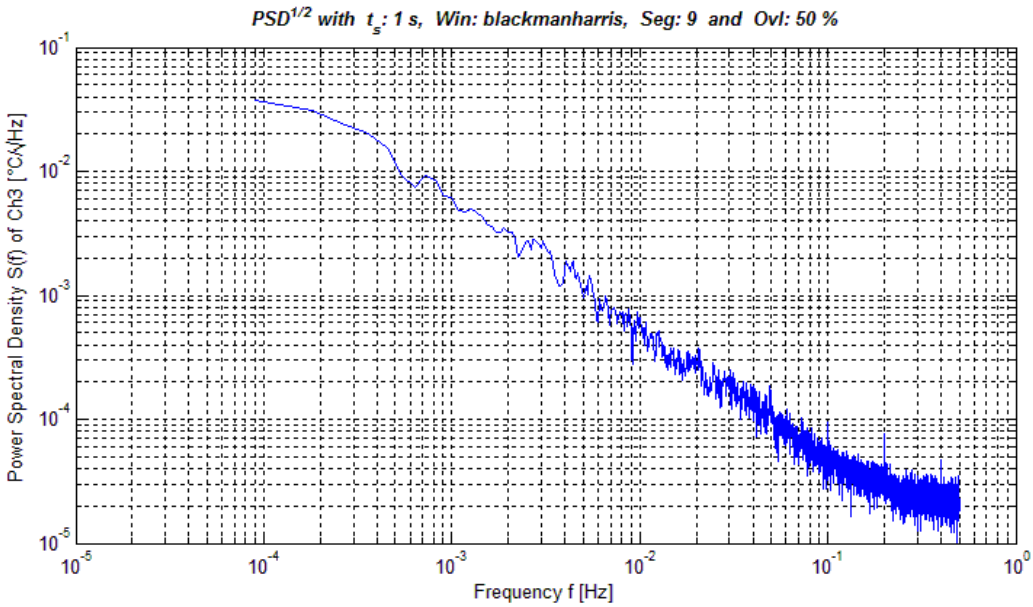


Abbildung 6-11: Leistungsdichtespektrum des Kanals 3 mit Präzisionswiderstand (S(f) [℃/ √Hz])

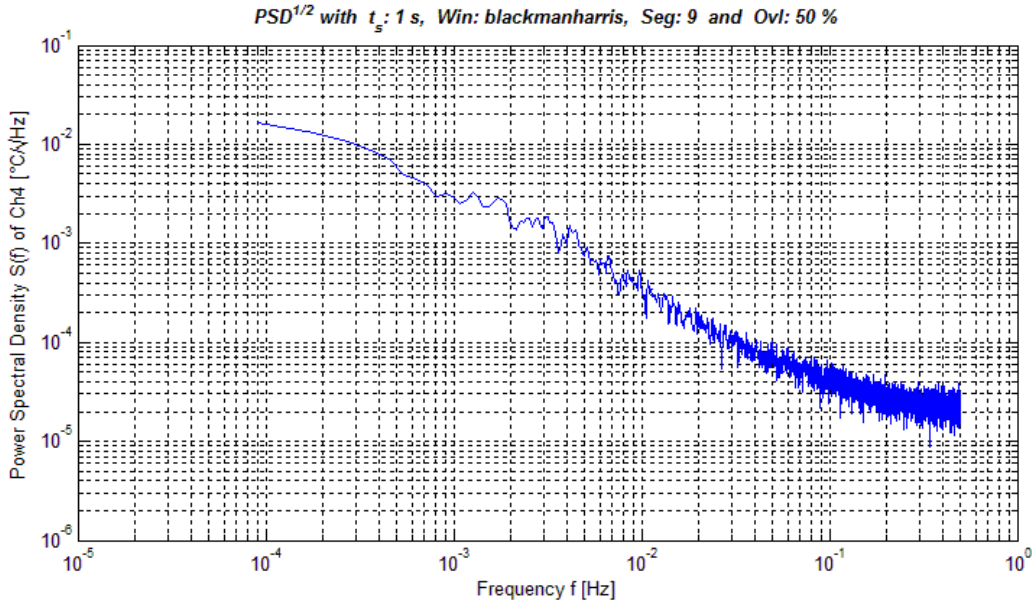


Abbildung 6-12: Leistungsdichtespektrum des Kanals 4 mit Präzisionswiderstand (S(f) [℃/ √Hz])

Nach demselben Prinzip, welches zur Berechnung des Störabstandes verwendet wurde wird nun für jeden Kanal die Präzision berechnet. Für $S_1(f)$ wird $3 \cdot 10^{-4}$℃/ √Hz angenommen.

Als Frequenzbereich wird derselbe wie oben definiert. Daraus folgt für √Hz

$$\sqrt{Hz} = \sqrt{f_{max} - f_{min}} = \sqrt{5 \cdot 10^{-1} Hz - 1 \cdot 10^{-3} Hz} = 0.7064\sqrt{Hz}$$

Für das Temperaturrauschen ergibt sich dann

$$N_{T_1} = \sqrt{Hz} \cdot S_1(f) = 0.7064\sqrt{Hz} \cdot 3 \cdot 10^{-4} \frac{°C}{\sqrt{Hz}} = 0.2119 \cdot 10^{-3} °C$$

Auf dieselbe Weise wird das Temperaturrauschen der anderen Kanäle berechnet.

Für den Kanal 2 ergibt sich für ein $S_2(f)$ von $2 \cdot 10^{-4}$℃/ √Hz ein Temperaturrauschen von $0.1413 \cdot 10^{-3}$℃. Für den Kanal 3 ergibt sich für ein $S_3(f)$ von $3 \cdot 10^{-4}$℃/ √Hz ein

Temperaturrauschen von $0.2119 \cdot 10^{-3}$℃. Für den Kanal 4 ergibt sich für ein $S_4(f)$ von $2.5 \cdot 10^{-4}$℃/ √Hz ein Temperaturrauschen von $0.1766 \cdot 10^{-3}$℃.

Auswertung der Resultate

Anhand der Messung können mehrere Beobachtungen gemacht werden. Die geforderte Auflösung von $1 \cdot 10^{-4}$℃ wird um eine maximale Abweichung von $1.119 \cdot 10^{-4}$℃ nicht erreicht. Zu bemerken gilt wiederum, dass es sich hierbei um Labormessungen handelt, was bedeutet, dass die Werte optimal sind und bei Messungen im Feld sicherlich schlechter ausfallen werden.

Aus der Messung geht auch hervor, dass jeder Kanal über ein eigenes Offset verfügt. Dieses Offset hängt von mehreren Faktoren ab und soll in der nächsten Messung genauer untersucht werden.

Durch die gefundenen Resultate kann nun auch auf den Einfluss des Layouts der Masse auf der Signal-Schicht geschlossen werden. Wie unter *4.1.2 Layout* erklärt wurde sind drei verschiedene Layouts verwendet worden. Die Kanäle 1 und 3 sind mit einem Masse-Ring ausgestattet. Der Kanal 2 wurde mit einem Masseplan ausgestattet und der Kanal hat keine Masse auf der Signal-Schicht. Wie zu sehen war hat der Kanal 2 mit dem Masse-Plan die höchste Präzision. Der Kanal 4 hat die zweithöchste Präzision und die Kanäle 1 und 3 sind nur halb so präzise wie der Kanal 1. Dies lässt auf einen positiven Einfluss durch einen Masseplan schliessen, wobei sogar angenommen werden muss, dass der Massen-Ring im Vergleich zu einem Layout ohne Masse einen negativen Einfluss aufweist.

6.3.1.3 Offset

Bedingungen und Methode

Da das Offset bei einer hochauflösenden Messkette eine Rolle spielt soll mit Hilfe von Messungen herausgefunden werden, was die Ursache für das Offset ist und wie es sich verhält.

Hierzu wurden zwei Messungen durchgeführt. Zuerst wurde die kalte Hardware gestartet, wobei sofort mit der Messung begonnen wurde. So konnte herausgefunden werden wie sich das Offset verhält. Um herauszufinden woher das Offset stammt wurde das PT1000-Widerstandsthermometer eines Kanals mit der Stromquelle eines anderen Kanals versorgt.

Beide Messungen wurden im erwähnten Labor mit der hergestellten Hardware durchgeführt und dauerten je 5 Stunden. Es sind also 18‘000 Messwerte pro Messung vorhanden.

Messung

Die zeitlichen Verläufe der gemessenen Signale sind im Anhang unter *14.8.1 Messungen des Offsets* ersichtlich. Die Abbildung 6-13 zeigt den Kanal 1.

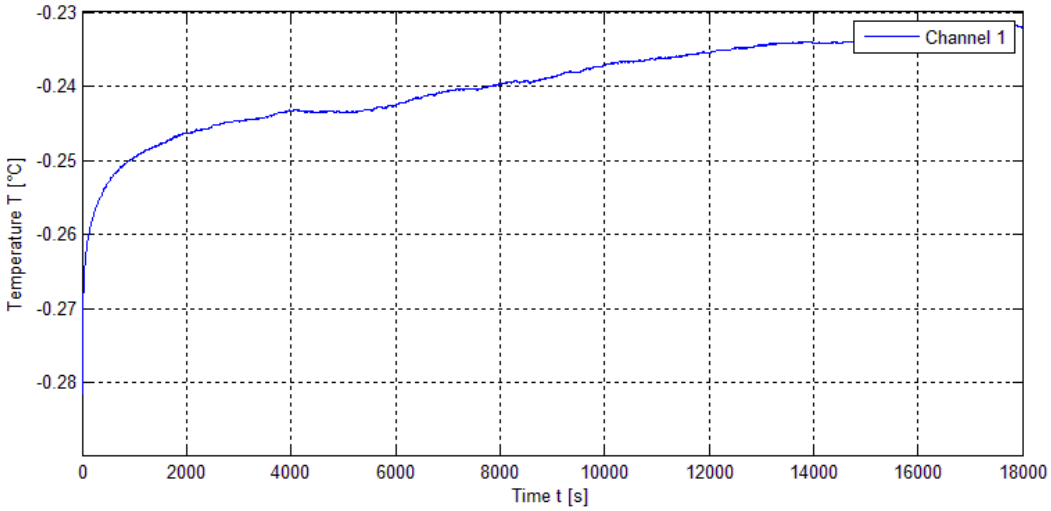


Abbildung 6-13: Offset-Messung des Kanals 1 (T=*f*(t) [℃])

Es ist gut ersichtlich wie sich das Offset im Bezug auf die Zeit verhält.

Die zweite durchgeführte Messung befindet sich auch im Anhang unter *14.8.1 Messungen des Offsets*. Wie bereits erwähnt wurden bei der zweiten Messung die Stromquellen der Kanäle vertauscht um die Herkunft des Offsets feststellen zu können.

Auswertung der Resultate

Der Zeitliche Verlauf des Offsets weist daraufhin, dass das Offset stetig variiert. Es ist anzunehmen, dass die Temperatur bei der Offset-Variation die grösste Rolle spielt. Das Offset bei t_0 stammt von den nicht idealen Komponenten. Wie auch ersichtlich ist, ist das Offset nach 5 Stunden noch nicht stabil. Dies womöglich, da sich dann noch immer nicht die ganze Hardware auf der Betriebstemperatur befindet.

Die zeigt die in den beiden Messungen gefundenen Offset-Werte.

Kanal i	**Offset T_{offset} [℃] (normal)**	**Offset T_{offset} [℃] (Stromquellen gewechselt)**	
		Stromquelle von Kanal i	**T_{offset} [℃]**
1	-0.26	3	-0.01
2	-0.48	4	-0.2
3	-0.28	2	0.2
4	0.04	1	-0.4

Tabelle 6-1: Vergleich der Offsetwerte

Anhand der gefundenen Werte kann kein Zusammenhang zwischen den Stromquellen und dem Offset gezeigt werden. Dies kann mehrere Gründe haben. Sehr wahrscheinlich ist, dass das Offset in der Verstärkerschaltung oder im A/D-Wandler hinzukommt. Das Offset könnte aber trotz des Nichtvorhandenseins eines Zusammenhangs auf den ersten Blick von den Stromquellen kommen.

6.3.1.4 Einfluss der Qualität der Spannungsreferenz

Bedingungen und Methode

Wie während den Funktionstests erklärt wurde war die Spannungsreferenz zuerst instabil. Nun soll untersucht werden welchen Einfluss dies auf das Temperaturrauschen der Messkette.

Um den Einfluss der Spannungsreferenz messen zu können wurde eine Messung mit einer instabilen (mit ungedämpften Impedanzwandler) Spannungsreferenz durchgeführt. Diese Messung dauerte 20 Minuten. Es sind also 1‘200 Messwerte vorhanden.

Diese Messung wurde mit dem Kanal 1 durchgeführt.

Messung

Das mit Hilfe der Messdaten erstellte Leistungsdichtespektrum kann im Anhang unter 14.8.2 Messungen des Einfluss der Qualität der Spannungsreferenz gefunden werden.

Der Wert der Leistungsdichte $S_{Instabil}(f)$ im vorher definierten Bereich beträgt für eine Messung mit einer instabilen Referenzspannung ca. $7 \cdot 10^{-2}$℃/ √Hz.

Die führt mit, einem √Hz von 0.7064√Hz, zu einem Temperaturrauschen von $49.45 \cdot 10^{-3}$℃.

Auswertung der Resultate

Wie gut erkennbar ist, ist die Amplitude des Rauschens bei einer instabilen Referenzspannung um einiges grösser. Hier beträgt sie $49.45 \cdot 10^{-3}$℃ wobei sie bei einer stabilen Referenzspannung $0.2119 \cdot 10^{-3}$℃ beträgt.

Dies zeigt, dass es wichtig ist eine fehlerfreie Referenzspannung zur Verfügung zu stellen.

6.3.1.5 Einfluss des Bypass-Kondensators der A/D-Wandler

Bedingungen und Methode

Da der Störabstand bei der A/D-Wandlung wichtig ist soll hier untersucht werden, welchen Einfluss die Qualität des Bypass-Kondensator auf das Rauschen hat.

Hierzu wurden mehrere Kondensatoren am A/D-Wandler des Kanals 1 nacheinander angebracht. Durch die Messung mit dem Präzisionswiderstand kann so der Einfluss des Kondensators bestimmt werden, insofern alle anderen Bedingungen konstant bleiben. Dies wurde durch eine rasche Ausführung der Messungen mit den verschiedenen Dielektrika versucht.

Die Messungen dauerten jeweils 15 Minuten wodurch 900 Messwerte vorhanden sind. Getestet wurde mit Kondensatoren mit Dielektrika Polykarbonat (MKC4) und Metallpolyester (MKS4) und auch mit einem Tantal-Kondensator.

Messung

Die einzelnen Messungen mit den verschiedenen Bypass-Kondensatoren wurden durchgeführt. Die resultierenden Spektren des Temperaturrauschens befinden sich im Anhang unter *14.8.3 Messungen des Einfluss des Bypass-Kondensators der A/D-Wandler*.

Ein Vergleich des Temperaturrauschens kann hier gemacht werden, da die Temperatur linear von dem Wert des A/D-Wandlers abhängt und so ein allfälliger Unterschied auch ersichtlich wäre.

Auswertung der Resultate

Anhand der erstellten Leistungsdichtespektren lässt sich kein Unterschied zwischen den verschiedenen Kondensatoren ausmachen. Trotzdem ist es möglich, dass ein kleiner Unterschied vorhanden ist. Um auch einen kleiner Unterschied im Rauschen der Messung finden zu können müsste über eine längere Zeit gemessen werden.

Aus dieser Messung lässt sich schliessen, dass die Bypass-Kondensatoren keinen grossen Einfluss auf die Qualität der Schaltung haben.

6.3.1.6 Einfluss der Signalpfad-Kondensatoren

Bedingungen und Methode

Um herauszufinden wie stark die Signalpfad-Filterungs-Kondensatoren das Rauschen herausfiltern wurde eine Messung ohne solche Kondensatoren im Signalpfad durchgeführt. Dieses Resultat soll dann mit dem unter *6.3.1.2 Präzision* gefundenen Resultat verglichen werden.

Die Messung wurde mit dem Kanal 1 durchgeführt. Die Kondensatoren C6, C10, C7 und C9 wurden entfernt. Die Dauer der durchgeführten Messung beträgt 1 Stunde. Das bedeutet es sind 3‘600 Messwerte vorhanden.

Wie auch in den anderen Messungen wurde die weiter oben erwähnte Hardware verwendet.

Messung

Das Resultat der Messung befindet sich im Anhang *14.8.4 Messungen des Einfluss der Signalpfad-Kondensatoren*. Das Leistungsdichtespektrum ist ersichtlich.

Für die Leistungsdichte $S_{NoCSig}(f)$ wird ein Wert von 1℃/ √Hz gewählt. Dies führt nach demselben Vorgehen und mit demselben Frequenzbereich wie unter *6.3.1.2 Präzision* zu einem Temperaturrauschen von 0.706℃.

Auswertung der Resultate

Bereits anhand der Leistungsdichte, welche hier ca. um einen Faktor 10^4 grösser ist als bei der Messung mit Kondensatoren kann erkannt werden, das das Rauschen mit dieser Konfiguration um einiges stärker ist. Der Vergleich des Temperaturrauschens bestätigt dies. Das Temperaturrauschen mit Kondensatoren beträgt $0.2119 \cdot 10^{-3}$℃. Ohne Kondensatoren beträgt dies 0.706℃.

Daraus lässt sich schliessen, dass die Filterkondensatoren auf dem Signalpfad unbedingt benötigt werden. Möglicherweise könnte mit Kondensatoren mit besserer Qualität das Rauschen noch etwas gesenkt werden.

6.3.1.7 Einfluss der Referenzspannungs-Kondensatoren

Bedingungen und Methode

Um den Einfluss der Filterungskondensatoren am Referenzspannungs-Eingang der A/D-Wandler messen zu können, wurden diese entfernt. Diese Messung wurde unter denselben

Bedingungen wie die vorherige Messung durchgeführt. Das bedeutet, dass der Kanal 1 und die vorhanden Hardware verwendet wurden.

Die Messung dauerte 1 Stunde und enthält somit 3‘600 Messwerte.

Messung

Das Resultat der Messung, in Form des Leistungsdichtespektrums befindet sich im Anhang *14.8.5 Messungen des Einfluss der Referenzspannungs-Kondensatoren*.

Der Wert der Leistungsdichte $S_{NoCVref}(f)$ beträgt ca. $3 \cdot 10^{-2}$℃/ √Hz. Daraus ergibt sich ein Temperaturrauschen von $21.19 \cdot 10^{-3}$℃. Die Berechnung wurde nach demselben Vorgehen und mit demselben Frequenzbereich wie unter *6.3.1.2 Präzision* durchgeführt.

Auswertung der Resultate

Aus dem Wert des Temperaturrauschens geht hervor, dass der Filterungskondensator am Eingang des A/D-Wandler seine Aufgabe erfüllt und benötigt wird.

Das Temperaturrauschen beträgt mit diesem Kondensator $0.2119 \cdot 10^{-3}$℃. Ohne beträgt es $21.19 \cdot 10^{-3}$℃. Das Rauschen wird also um den Faktor 100 stärke r.

Mit einem Kondensator höherer Qualität könnten hier evtl. sogar noch besser Werte erzielt werden.

6.3.2 Messung mit PT1000-Widerstandsthermometern

Bedingungen und Methode

Nachdem die Schaltung mit Hilfe von Präzisionswiderständen ausgewertet wurde, wurde noch eine Labormessung mit PT1000-Widerstandsthermometern durchgeführt. Dies wurde gemacht um zu sehen wie sich die Schaltung beim Anschliessen der Sensoren verhält.

Vier PT1000-Widerstandsthermometer wurden angeschlossen. Die Sensoren selber wurden an einem Aluminium-Block befestigt, damit eine gewisse Trägheit im Bezug auf die Temperatur erreicht wird.

Die Messung wurde während einer Nacht durchgeführt und dauerte 13 Stunden und ergab somit 46‘800 Messwerte.

Messung

Die Abbildung 6-14 zeigt die Messung im Zeitbereich.

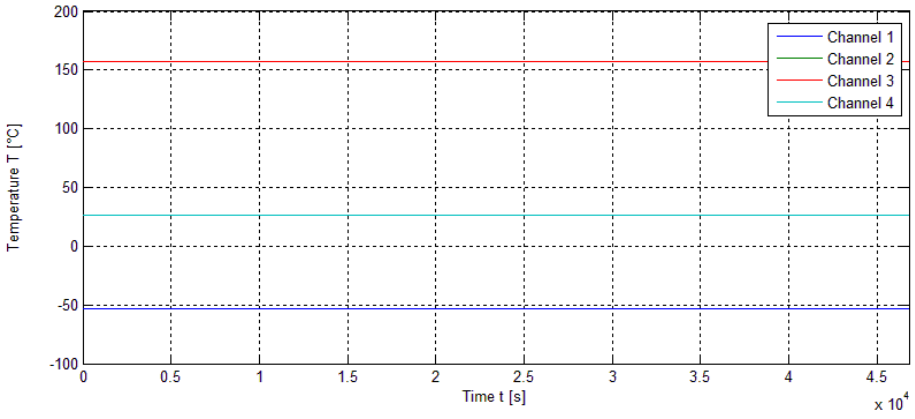


Abbildung 6-14: Messung mit PT1000-Widerstandsthermometern im Zeitbereich

Der Kanal 2 und 4 liegen bei etwa 27℃. Dies entspr icht der Realität. Der Kanal 1 und 3 sind allerdings bei den Extremwerten. Kanal 1 liegt bei -53.224℃ was dem tiefsten Messbaren Widerstand entspricht. Das bedeutet, der Kanal 1 war im Kurzschluss. Der Wert des Kanals 3 liegt bei 157.166℃. Dies entspricht dem maximale n Widerstand wodurch erkannt werden kann, dass die Verbindung unterbrochen wurde.

Das Rauschen der Kanäle, die funktionierten, wurden auch gemessen. Die Abbildung 6-15 und Abbildung 6-16 zeigen das Leistungsdichtespektrum der Kanäle 2 und 4.

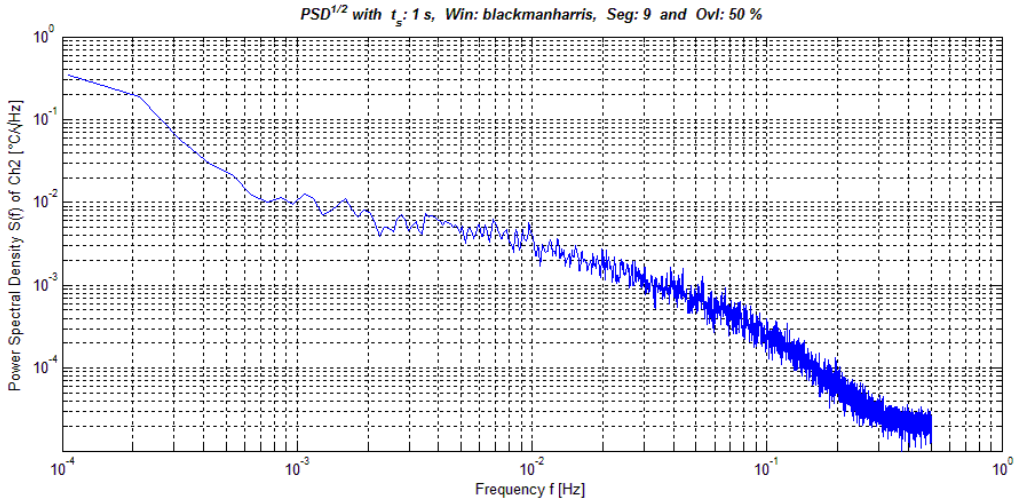


Abbildung 6-15: Leistungsdichtespektrum des Kanals 2 mit PT1000-Widerstandsthermometer (S(f) [℃/ √Hz])

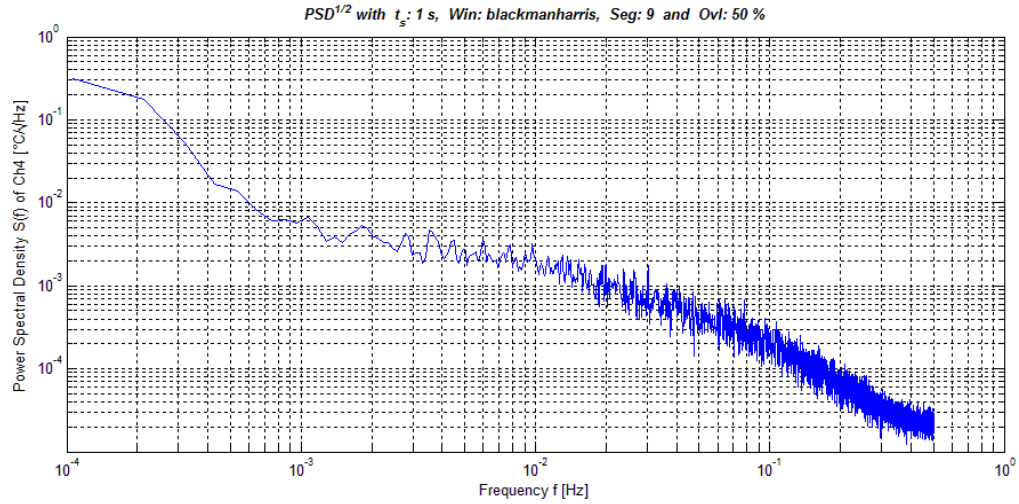


Abbildung 6-16: Leistungsdichtespektrum des Kanals 4 mit PT1000-Widerstandsthermometer (S(f) [℃/ √Hz])

Nach demselben Prinzip, welches zur Berechnung des Störabstandes unter *6.3.1.2 Präzision* verwendet wurde wird für die Kanäle 2 und 4 das Rauschen berechnet. Für $S_2(f)$ wird $3 \cdot 10^{-3}$℃/ √Hz angenommen.

Als Frequenzbereich wird derselbe wie oben definiert. Daraus folgt für √Hz

$$\sqrt{Hz} = \sqrt{f_{max} - f_{min}} = \sqrt{5 \cdot 10^{-1} Hz - 1 \cdot 10^{-3} Hz} = 0.7064\sqrt{Hz}$$

Für das Temperaturrauschen ergibt sich dann

$$N_{T_2} = \sqrt{Hz} \cdot S_1(f) = 0.7064\sqrt{Hz} \cdot 3 \cdot 10^{-3} \frac{°C}{\sqrt{Hz}} = 2.119 \cdot 10^{-3} °C$$

Auf dieselbe Weise wird das Temperaturrauschen des Kanals 4 berechnet. $S_4(f)$ beträgt ca. $5 \cdot 10^{-3}$℃/ √Hz. Daraus folgt ein Temperaturrauschen von $3.5 \cdot 10^{-3}$℃.

Auswertung der Resultate

Im Gegensatz zu den bisherigen Erkenntnissen erzielt der Kanal 4 hier ein grösseres kleineres Rauschen als der Kanal 2. Dies wird an den Verbindungen zu dem PT1000-Widerstandsthermometern liegen.

Wenn die Werte mit denen der Messungen mit Präzisionswiderständen verglichen werden ist ersichtlich, dass bei der Messung mit PT1000-Widerstandsthermometern mehr Rauschen vorhanden ist. Die Rauschwerte mit den Sensoren sind 10 bis 20 Mal grösser. Dementsprechend kleiner ist die Präzision.

6.4 Feldmessungen

Nachdem die Messkette im Labor ausgewertet wurde, wurden Feldmessungen durchgeführt. Während dem von den Mitarbeitern der Bioabteilung der Hes-so Valais / Wallis Sion Fermentationen durchgeführt wurden, wurden die Messungen durchgeführt.

Da diese Messungen allerdings im Zusammenhang mit der Validierung des thermischen Models des Bioreaktors stehen werden die Resultate im Bezug auf die Fermentation dort aufgezeigt. In diesem Teil wird das Rauschen im Feld untersucht.

6.4.1 Präzision

Bedingungen und Methode

Damit das Rauschen analysiert werden konnte wurde aus der gesamten Fermentation, welche ca. zwei Tage dauerte, Werte eines Teils entnommen, in dem die Temperatur im Reaktor konstant gehalten wurde. Diese Daten wurden dann verwendet um die Leistungsdichtespektren zu erstellen.

Die Daten stammen aus der zweiten Fermentation welche im Gebäude F der Hes-so Valais / Wallis Sion vom 23.06.2010 bis zum 25.06.2010 durchgeführt wurde. Diese wurde nur mit 4 gemessenen Kanälen durchgeführt, da die zweite Erweiterungskarte noch nicht vorhanden war. Dabei handelt es sich um die Messung der Luft (Kanal 1), des Zulaufs des Wassers des Wärmetauschers (Kanal 2) des Ablaufs des Wassers des Wärmetauschers (Kanal 3) und des Mantels (Kanal 4).

Die Fermentation wurde mit dem Laborfermenter Typ NLF22 19 Liter durchgeführt. Dabei wurden bei der ersten Fermentation vier PT1000-Widerstandsthermometer verwendet. Der Kanal 1 misst die Lufttemperatur, der Kanal 2 die Temperatur der Zuleitung, der Kanal 3 die der Ableitung und der Kanal 4 die Temperatur des äusseren Gehäuses des Fermenters.

Messung

Die erstellten Leistungsdichtespektren befinden sich im Anhang unter *14.8.6 Feldmessungen*.

Die Tabelle 6-2 fast die Messung und deren Berechnung zusammen.

Kanal i	Leistungsdichte $S_i(f)$ [℃/ √Hz]	Temperaturrauschen N_{Ti} [℃]
1	$3 \cdot 10^{-1}$	0.2119
2	$2 \cdot 10^{-2}$	0.0141
3	$2 \cdot 10^{-2}$	0.0141
4	$6 \cdot 10^{-2}$	0.0424

Tabelle 6-2: Resultat der Feldmessung im Bezug auf die Genauigkeit

Die Temperaturrauschen wurden wie bisher für einen Bereich von $1 \cdot 10^{-3}$Hz bis $5 \cdot 10^{-1}$Hz berechnet.

Auswertung der Resultate

Aus den Resultaten geht hervor, dass die Rausch-Werte deutlich ansteigen. Dies konnte auch so erwartet werden.

Der Kanal 1 verfügt über das grösste Temperaturrauschen, da die in der Luft hängende Sonde wie eine Antenne für Störungen wirkt.

Die Zu- und Ab-Leitung des Fermenters bietet eher stabile Temperaturwerte, da es sich bei den Röhren auf denen die PT1000-Widerstandsthermometer platziert wurden um geerdetes Metall handelt. Der Kanal 4, welcher die Gehäuse-Temperatur misst, weist ein grösseres Temperaturrauschen als die Kanäle 2 und 3 auf. Der grössere Rauschwert ist durch die grosse Fläche des Gehäuses zu erklären. Die grosse Fläche führt zu einer erleichterten Aufnahme von Störungen.

Die Rauschwerte liegen allgemein im Bereich zwischen $15 \cdot 10^{-3}$℃ bis zu 0.21℃. Es ist gut ersichtlich, dass die Feldmessung durch die unzähligen externen Einflüsse stark verschlechtert wird. Wie diese Einflüsse gemindert werden könnten wird unter *9.2.2 Schaltung der Erweiterungskarte* erklärt.

7. Installation an einem Bioreaktor

7.1 Fermentationsanlage

Die Messkette wurde an einer Fermentationsanlage in der Pilothalle der Bioabteilung Hes-so Valais / Wallis Sion installiert. Fermentationsanlagen sind eine Untergruppe der Bioreaktoren.

Bei der Fermentationsanlage handelt es sich um einen Laborfermenter Typ NLF22 19 Liter mit der Referenznummer DET09025 und der Werknummer W25666.

Dies ist eine Fermentationsanlage mit einem eher kleinen Volumen und wird in erster Linie für Forschungszwecke verwendet.

7.2 Platzierung der Sonden

Damit die Temperaturen des Bioreaktors gemessen werden konnten wurden mehrere Sonden (PT1000-Widerstandsthermometer) platziert. Es wurden total 16 Sonden montiert.

7.2.1 Sonden ausserhalb des Reaktors

Um den Wärmeabgabe der Anlage charakterisieren zu können wurden am Bioreaktor 8 Sonden platziert. Die Abbildung 7-1 zeigt den Laborfermenter und die platzierten Sonden.

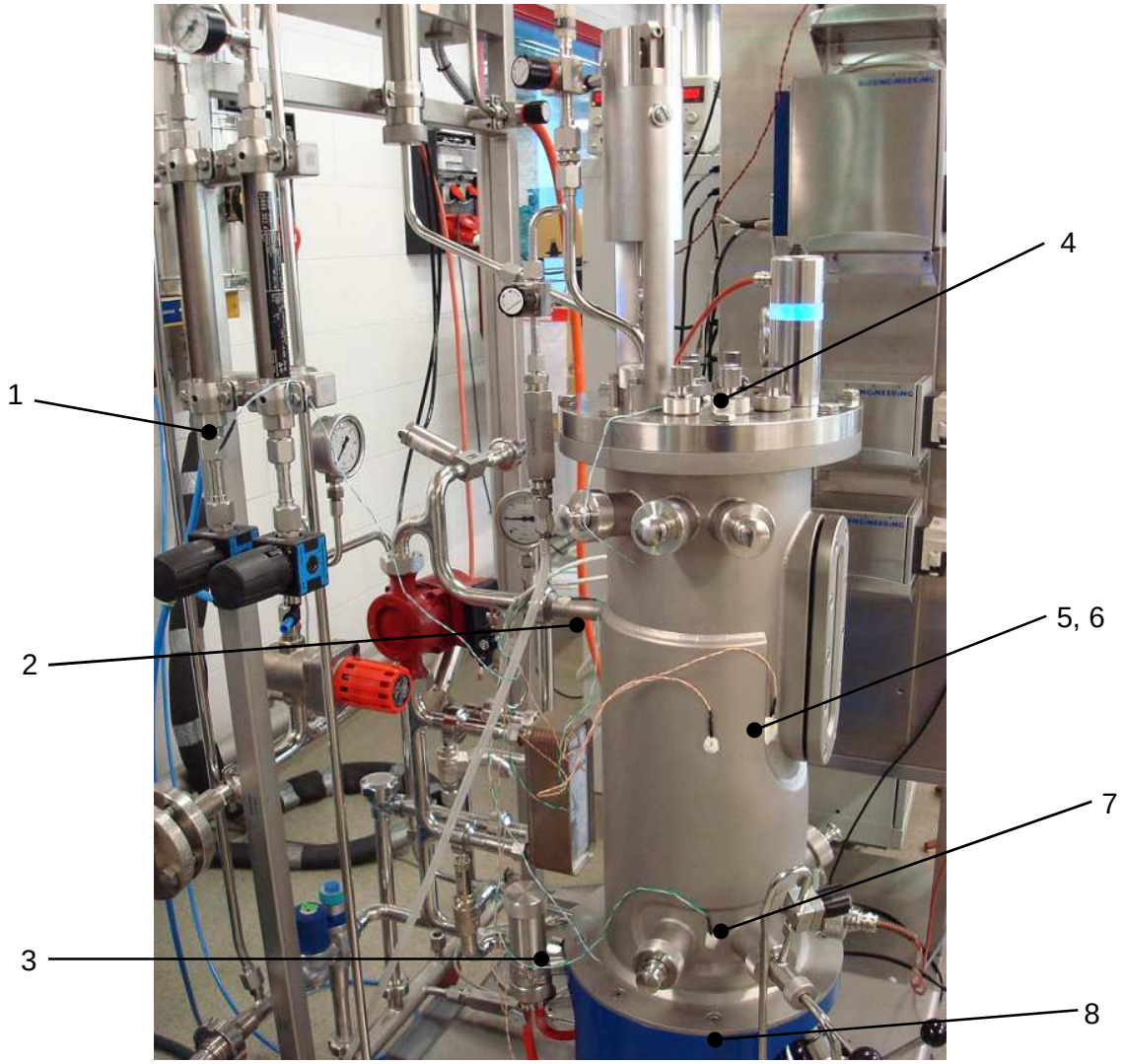


Abbildung 7-1: Laborfermenter mit den angebrachten Sonden

Die Tabelle 7-1 zeigt die verschiedenen Sonden, deren Platzierung und was gemessen wird.

Sensor	Ort	Messaufgabe
1	Luft	Lufttemperatur
2	Abfluss des Wassers für den Wärmeaustausch	abgeführte Wärme
3	Zufluss des Wassers für den Wärmeaustausch (nicht sichtbar)	zugeführte Wärme
4	Deckel	Temperatur des Mantels oben
5	Mantel (aussen Mitte)	Temperatur des äusseren Mantels in der Mitte
6	Mantel (innen Mitte)	Temperatur des inneren Mantels in der Mitte
7	Mantel (innen unten)	Temperatur des inneren Mantels unten
8	Metallboden der Anlage	Referenz

Tabelle 7-1: Sensoren und deren Platzierung und Messaufgabe

Die PT1000-Widerstandsthermometer wurden mit einem Temperaturleitenden Klebstoff angebracht und mit Silikon abgedichtet, da sich während der Fermentation Kondenswasser auf den erhitzten Aussenteile bilden kann.

7.2.2 Stabsonde

Damit die Temperatur auch innerhalb des Reaktors gemessen werden kann wurde die unter *4.3 Stabsonde* erklärte Stabsonde im Reaktor platziert.

Die Abbildung 7-2 zeigt die im Fermenter platzierte Stabsonde.

Abbildung 7-2: Die Stabsonde im Reaktor

7.3 Installation der Messkette

Nachdem alle Sonden angebracht waren wurde die gesamte Messkette Installiert. Das heisst, die erstellte Hardware mit der FPGA-EBS-Karte im Gehäuse und ein Laptop zum Betrieb der erstellten Matlab-Software wurden bereitgestellt.

Die zeigt wie die Hardware in der Umgebung der Fermentationsanlage platziert wurde.

Abbildung 7-3: Installation der Messkette

Die Teile der Messkette sind Beschriftet. Die restliche Vorrichtung war bereits vorhanden und wird zur Messung und Steuerung des Reaktors verwendet.

8. Validierung eines thermischen Modells

Da die Messkette nun installiert war und auch funktionierte sollte nun das thermische Modell des Reaktors, mit Hilfe der durchgeführten Messungen der Fermentationen, validiert werden.

8.1 Idee

Während der Fermentation von Hefe und durch die Zugabe von Nährstoffen für die Hefe-Zellen entsteht ein Protein welches fluoreszierende Eigenschaften aufweist. Die Herstellung dieses Proteins befindet sich derzeit in der Entwicklungsphase und wird erforscht. Das fluoreszierende Protein soll als Indikator in einem anderen biologischen Prozess in der Industrie verwendet werden.

Um das Wachstum der Zellen und die Erzeugung des Proteins besser überwachen zu können soll die verteilte Temperaturmessung verwendet werden.

Die einzelnen Zellen der Biomasse vermehren sich und nehmen die Nährstoffe auf und verwerten diese zu dem gewünschten Protein. Bei dieser Umwandlung entsteht Kohlendioxid, wobei von den Zellen auch Wärme abgegeben wird. Durch eine präzise Temperaturmessung sollte es nun möglich sein herauszufinden in welchem Stadium des Wachstums sich die Biomasse befindet und wie die Produktion des Proteinses vorangeht.

Durch die Wärmeproduktion der Zellen müsste der Temperaturregelkreis den Reaktor kühlen. Dies ist allerdings bei dem eher kleinen Laborfermenter (Model NLF22 19 Liter), welcher verwendet wird, nicht der Fall, da die Wärmeverluste über die Hülle zu gross sind.

Durch die Erstellung einer thermischer Bilanz zwischen der zugeführten Wärme, der abgeführten Wärme und des Wärmeverlustes sollte es möglich sein die Wärmeproduktion der Zellen zu charakterisieren und zu quantisieren, was dem eigentlichen Ziel der Messkette entspricht. Dies dürfte bei dem kleinen Reaktor allerdings aus erwähnten Gründen schwierig sein. Bei Reaktoren mit einem grösseren Volumen-Oberfläche-Verhältnis wären die Verluste kleiner und die Wärmeabgabe der Zellen besser ersichtlich. Bei sehr grossen Reaktoren (50‘000Liter Inhalt z.B.) ist sogar eine Kühlung des Reaktors nötig, da die Zellen merklich mehr Wärme Produzieren.

8.2 Thermisches Modell

Das thermische Modell konnte in der zur Verfügung stehenden Zeit nicht erstellt werden. Ein intensives Studium des vorhanden Reaktors und der Thermodynamik wäre nötig um das Modell zu erstellen.

In dem Modell würde neben allen thermischen Eigenschaften des Reaktors auch andere physikalische Gegebenheiten wie der Durchfluss des Wärmekreislaufs, welcher die Wärme in den Reaktor transportiert, usw. eine Rolle spielen.

Die gemessenen Fermentationen sollen allerdings gezeigt und erklärt werden. Bei allen drei Fermentationen wurde derselbe biologische Prozess durchgeführt.

Alle hier verwendeten Messdaten befinden sich auf der beigelegten CD im Verzeichnis *measure*.

8.3 Fermentation 1

8.3.1 Bedingungen und Methode

Die Messungen wurden mit der erklärten Hardware, Installation und dem erwähnten Laborfermenter durchgeführt.

Diese Fermentation dauerte vom 15.06.2010 um 10:00Uhr bis zum 17.06.2010 um 12:15Uhr. Die Messung dauerte 26h16min18s. Es sind somit 94‘578 Messwerte vorhanden.

Vier Kanäle wurden aufgezeichnet, da die Hardware zur Aufzeichnung von mehr Kanälen noch nicht vorhanden war.

Kanal 1 ist die Messung der Lufttemperatur. Kanal 2 misst die äussere Gehäuse-Temperatur, Kanal 3 den Zufluss des Wärmekreislaufs und Kanal 4 den Abfluss des Wärmekreislaufs.

8.3.2 Messung

Die Abbildung 8-1 zeigt den zeitlichen Verlauf der gefilterten Messignale.

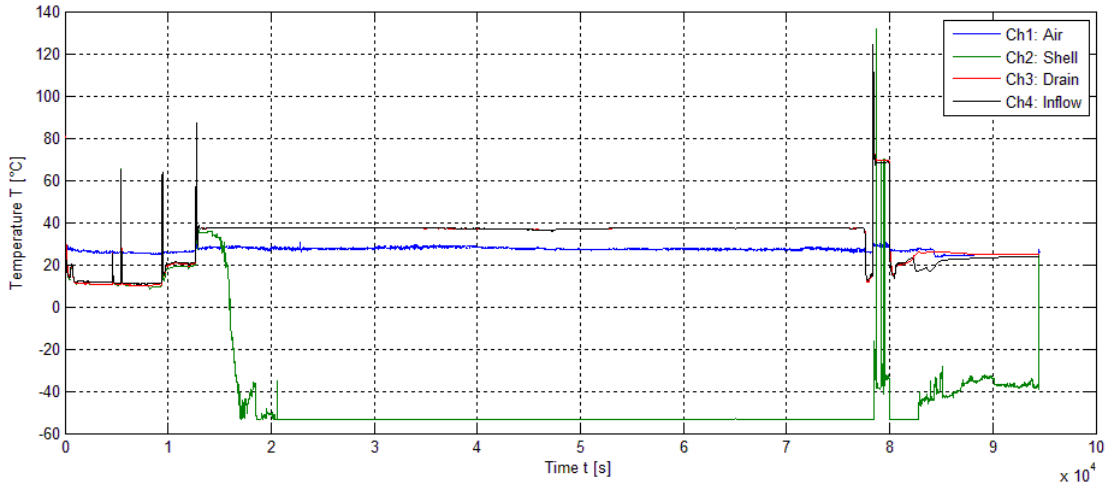


Abbildung 8-1: Zeitlicher Verlauf der Temperaturen der ersten Fermentation

Die Messung beginnt beim Ende der Sterilisation. In der ersten Phase ist die Kühlung des Reaktors auf 10℃ ersichtlich. Dies wird gemacht, d ass der Innenraum des Reaktors nach der Sterilisation steril bleibt. Als nächstes ist ersichtlich wie die Temperatur der Zu- und Ableitung auf ca. 37℃ gebracht wird. Während der Z eit in der die Temperatur auf diesem Wert ist wachsen die Zellen im Reaktor und erstellen dann gegen Ende dieser Phase das erwähnte Protein. Die Ausschläge am Schluss der Messung entsprechen bereits der Reinigung des Reaktors, sind also nicht mehr ausschlaggebend.

Ersichtlich ist zudem wie die Sonde am Mantel (Kanal 2) den Minimalwert von -53.224℃ annimmt. Dies ist ein Kurzschluss, welcher durch die Nachwirkungen der Kondensation kontinuierlich zu Stande kam.

Die Abbildung 8-2 zeigt das Problem der Ausschläge während einer Messung, welches durch den Median-Filter behoben wurde.

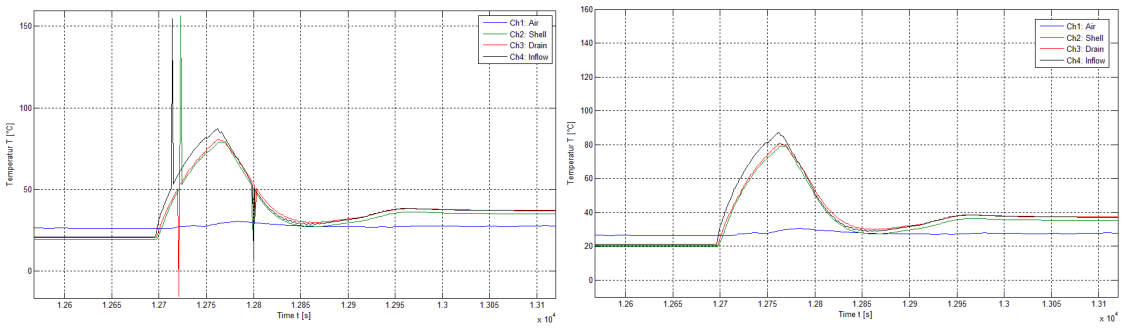


Abbildung 8-2: Filterung der Ausschläge während der Messung

Die vorhandene Kurve zeigt die Erwärmung des Reaktors von ca. 20℃ auf ca. 80℃. Links sind die ungefilterten Daten und rechts die gefilterten Daten vorhanden. Wie zu sehen ist verschwinden die vorhandenen Ausschläge vollkommen und die Kurve wird ergänzt.

8.3.3 Auswertung und Resultate

Die erste Fermentation hat in erster Linie die Funktion der Messkette im Feld bestätigt, auch wenn der Einsatz des Median-Filters, wie zu sehen war, benötigt wird.

Des Weiteren wurden die Mängel der Installation aufgezeigt. Dies kann allerdings auch daher kommen, dass die Sensoren nur ein paar Stunden vor der Fermentation angebracht worden sind und so die Dichtung mit dem Silikon noch nicht vollständig trocknen konnte.

Auf den erstellen Graphen ist gut zu sehen, wie sich die einzelnen Partien des Reaktors im Bezug auf eine Erwärmung verhalten. Sobald die Temperatur der Zuleitung ansteigt steigt, mit einer gewissen Verzögerung auch die Temperatur der Ableitung und des Mantels. Auch die Messung der Luft zeigt, mit einer grösseren Verzögerung, die Aufnahme eines Teils der vom Reaktor abgestrahlten Wärme.

Falls nun ein thermisches Modell bereits vorhanden wäre könnten die Daten mit Hilfe von Matlab exakt ausgelesen und ausgewertet werden. So könnte das thermische Modell validiert werden.

8.4 Fermentation 2

8.4.1 Bedingungen und Methode

Die zweite Fermentation wurde mit derselben Hardware und derselben Installation wie die erste durchgeführt.

Die Aufzeichnung der Messdaten dauerte vom 22.06.2010 um 12:55Uhr bis zum 24.06.2010 um 13:32Uhr. Die Messung dauerte 48h37min3s. Somit ergeben sich 175‘023 Messwerte. Die Fermentation selber begann nach der Sterilisation, welche am 23.06.2010 ca. um 07:00Uhr durchgeführt wurde.

Vier Kanäle wurden aufgezeichnet, da die Hardware zur Aufzeichnung von mehr Kanälen noch nicht vorhanden war.

Kanal 1 ist die Messung der Lufttemperatur. Kanal 4 misst die äussere Gehäuse-Temperatur, Kanal 3 den Zufluss des Wärmekreislaufs und Kanal 2 den Abfluss des Wärmekreislaufs.

8.4.2 Messung

Die Abbildung 8-3 zeigt das Resultat der Messung.

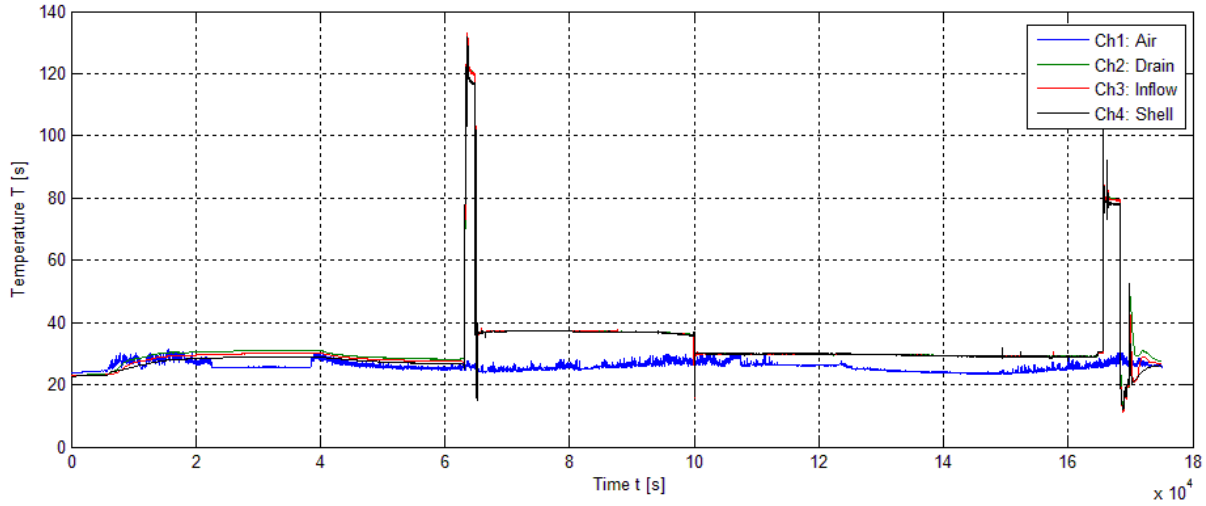


Abbildung 8-3: Zeitlicher Verlauf der Temperaturen der zweiten Fermentation

Im Unterschied zur ersten Fermentation begann diese Messung einige Stunden vor der eigentlichen Fermentation. Die Sterilisierung, welche vor der Fermentation durchgeführt wird beginnt zum Zeitpunkt bei dem die Temperaturen aller gemessenen Reaktorteile auf 121℃ steigen.

Die Sonde in der Luft (Kanal 1) wurde dieses Mal etwas weiter entfernt vom Reaktor platziert und somit ist in dieser Messung auch ein viel kleinerer Einfluss der Wärmestrahlung des Reaktors zu sehen.

Diesmal Verläuft die Temperatur während der Fermentation etwas anders. Wie zu sehen ist bleibt der Reaktor nach der Sterilisation auf 37℃ und wird nach ca. 10 Stunden auf 31℃ gekühlt.

Am Schluss der Messung ist wiederum die Reinigung des Reaktors ersichtlich.

Da bei der Sterilisation die grössten Temperaturunterschiede ersichtlich sind soll diese in der speziell aufgezeigt werden.

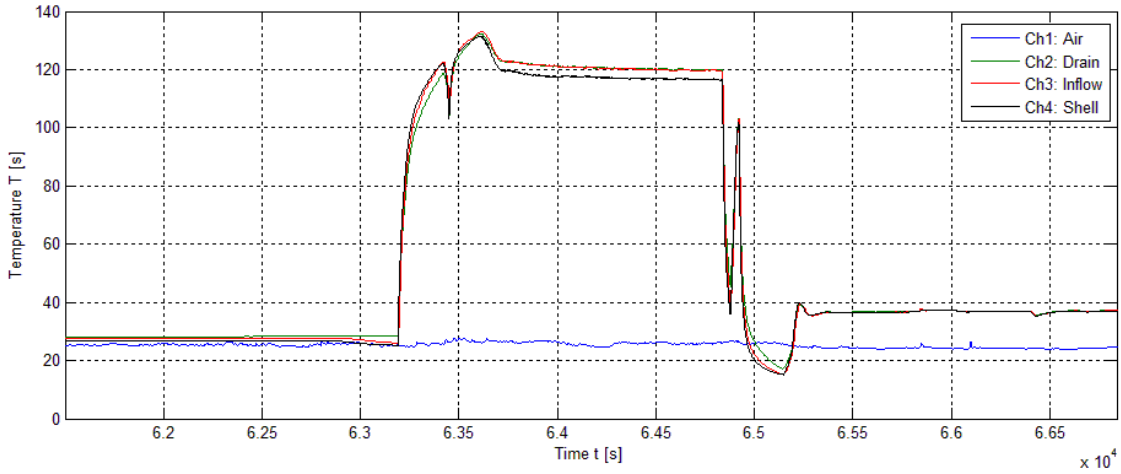


Abbildung 8-4: Sterilisation vor der zweiten Fermentation

Ersichtlich ist, in welcher kurzen Zeit die Temperatur auf die erforderlichen 121℃ steigt und wie lange der Reaktor auf dieser Temperatur gehalten wird. Zu erkennen sind auch die Unterbrüche im Verlauf der Temperatur, welche mehrere Ursachen haben können.

8.4.3 Auswertung und Resultate

Durch die Messung eines gesamten Versuches von der Sterilisation über die Fermentation könnte nun mit dem thermischen Modell noch Informationen über den Wärmeaustausch während der Sterilisation gewonnen werden.

Die Messung ergab auch, dass die Wärmestrahlung des Reaktors nicht intensiv ist, da durch die Entfernung der Luftsonde um ca. 50cm der Einfluss beinahe verschwindet.

Die Regelung der Temperatur ist anhand der Abbildung der Sterilisation auch gut zu erkennen. Der Fehler, welcher während der Regelung vorhanden ist, könnte so bestimmt und korrigiert werden.

8.5 Fermentation 3

8.5.1 Bedingungen und Methode

Die dritte Fermentation wurde mit derselben Hardware und derselben Installation wie die erste und zweite durchgeführt.

Die Aufzeichnung der Messdaten dauerte vom 29.06.2010 um 17:00Uhr bis zum 01.07.2010 um 11:00Uhr. Die Messung dauerte 41h44min35s, wodurch sich 150‘275 Messwerte ergeben. Die Fermentation selber begann nach der Sterilisation, welche am 29.06.2010 ca. um 17:15Uhr durchgeführt wurde.

Diese Mal wurden 8 Kanäle aufgezeichnet. Bei den 8 Kanälen handelt es sich um die einzelnen Elemente der Stabsonde. Die Kanäle 1 bis 8 entsprechenden den Elementen 1 bis 8 der Stabsonde. Der Kanal 1 ist das unterste Element und dementsprechend der Kanal 8 das oberste.

Mit der Wahl dieser Sonden ist es möglich die thermischen Vorgänge innerhalb des Reaktors zu bestimmen.

Das Offset der einzelnen Kanäle wird bei dieser Messung manuell aufgehoben, da die Unterschiede zwischen den Elementen der Stabsonde zu klein sind und vom Offset verfälscht werden.

8.5.2 Messung

Die Abbildung 8-5 zeigt die gesamte Fermentation inklusive der Sterilisation, welche vor der Fermentation durchgeführt wurde.

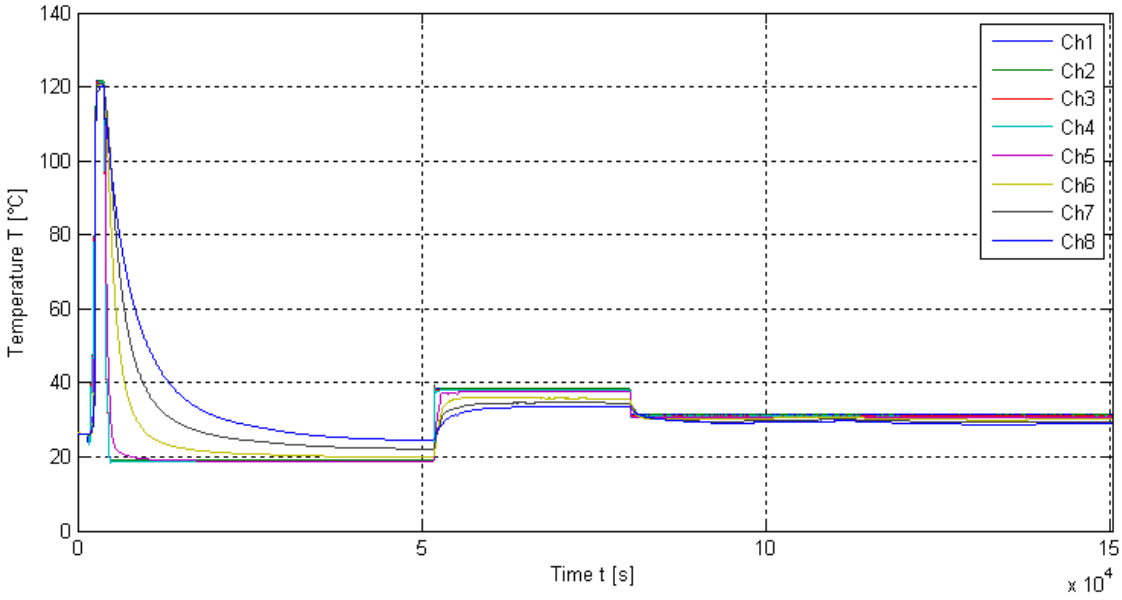


Abbildung 8-5: Zeitlicher Verlauf der Temperaturen der dritten Fermentation

Der zeitliche Verlauf der Messung zeigt zuerst die Sterilisation bei der der gesamte Innenraum des Reaktors auf die erforderlichen 121℃ steigt. Danach wird der Reaktor unter 20℃ gekühlt, damit er steril bleibt.

Den Beginn der Fermentation kann ohne Probleme ausgemacht werden. Die Fermentation beginnt zum Zeitpunkt an dem die Kanäle, welche sich in der Flüssigkeit befinden, auf 37℃ steigen. Die anderen Kanäle steigen auch an, da der leere Raum des Reaktors auch erwärmt wird.

So kann herausgefunden werden, welche Kanäle in der Flüssigkeit sind und welche nicht. Zudem zeigt die Abkühlung nach der Sterilisation dies auch. Wie zu sehen ist brauchen die Kanäle 5, 6, 7 und 8 länger um abzukühlen. Dies bedeutet, diese befinden sich nicht in der Flüssigkeit.

Die Abbildung 8-6 zeigt die Sterilisation von näherem und belegt die Wirkung dieser.

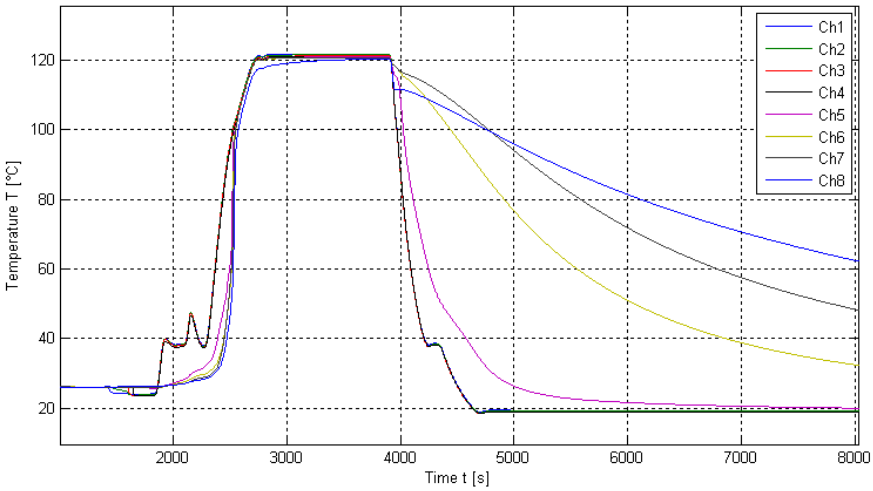


Abbildung 8-6: Sterilisation vor der dritten Fermentation

Erkennbar ist, dass bei den Kanälen, welche sich nicht in der Flüssigkeit befinden die Temperatur langsamer steigt oder sinkt, aber immer auch die Werte der Flüssigkeit erreichen. Dies soll heissen, dass die Sterilisation im gesamten Innenraum des Reaktors vorgenommen wird.

Die oberen Kanäle, welche sich nicht in der Flüssigkeit befinden, haben klare Temperaturunterschiede. Innerhalb der Flüssigkeit allerdings sind die Differenzen nicht gross. Deshalb wird in der Abbildung 8-7 hier ein Teil der Fermentation vergrössert dargestellt werden.

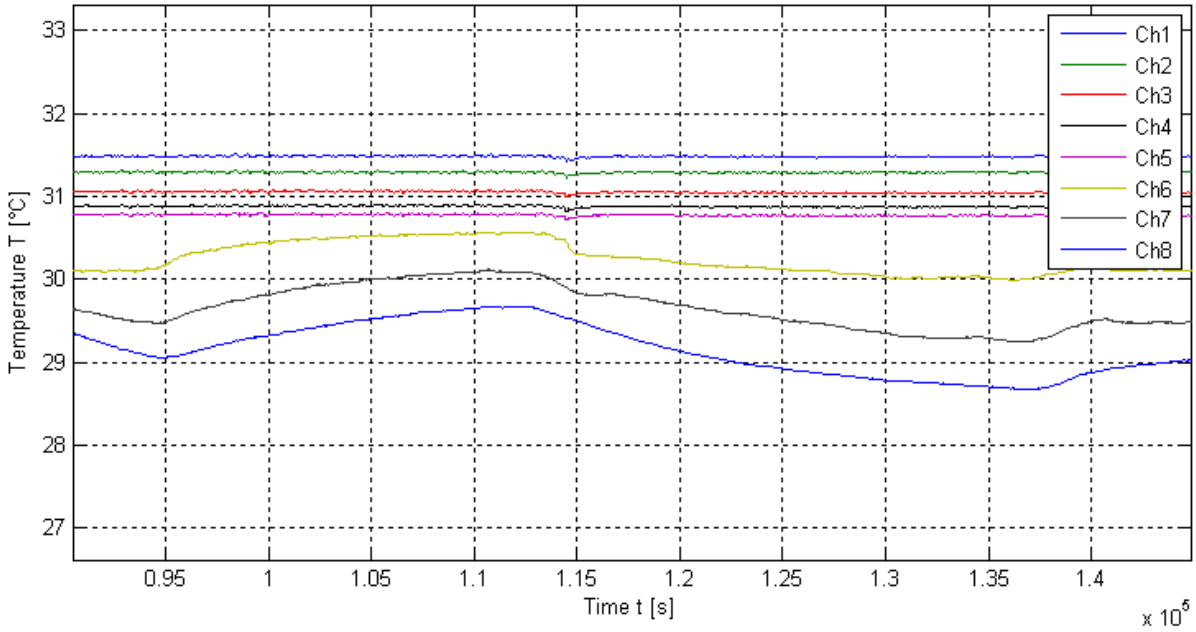


Abbildung 8-7: Wachstumsphase der dritten Fermentation

Diese Messung zeigt einen Ausschnitt der Wachstumsphase der Zellen. Die Kanäle 1 bis 4 haben kleinere Temperaturdifferenzen als die Kanäle 5 bis 8. Dies ist auf Grund der besseren Wärmeverteilung in der Flüssigkeit so.

Bemerkt werden kann auch, dass am tiefsten gemessenen Punkt die höchste Temperatur herrscht. Beim zweiten Element ist die Temperatur am zweithöchsten, und so weiter.

8.5.3 Auswertung und Resultate

Mit Hilfe des thermischen Modells des Reaktors könnte nun eine genaue Bilanz des Wärmeaustausches im Innern des Reaktors erstellt werden.

So können aber trotzdem einige Erkenntnisse gewonnen werden.

Die Sterilisation erfüllt ihre Aufgabe von unten bis oben. Der gesamte Reaktorinnerraum wird auf die benötigte Temperatur geheizt.

Wie sich der Innenraum des Reaktors, welcher nicht mit Flüssigkeit gefüllt ist, verhält konnte auch gesehen werden. Dabei ist interessant, dass dieser sich der Temperatur der Flüssigkeit nur sehr langsam anpasst.

Selbst in der Flüssigkeit sind Temperaturunterschiede messbar. So erhält man durch die Messung mit der Stabsonde genügend Daten über das Verhalten des Innenraums des Reaktors im Bezug auf die Wärme.

9. Schlussfolgerungen

9.1 Zusammenfassung

Zum Schluss der Projektarbeit lässt sich sagen, dass eine Messkette erstellt werden konnte welche den Vorstellungen entspricht.

Insgesamt wurde die Hardware für 8 Kanäle hergestellt, wobei am Bioreaktor 16 Sonden angebracht worden sind. Falls also die Hardware für die restlichen 8 Kanäle erstellt wird kann im Bezug auf die Temperatur der ganze Bioreaktor ausgemessen werden. Die hergestellte Stabsonde funktioniert auch wie sie soll und hält den extremen Bedingungen im Bioreaktor stand.

Die Schaltung in der FPGA funktioniert in der erstellten Konfiguration, welche das Auslesen der Daten erlaubt, wie dies geplant war. In der FPGA ist zudem noch genügend Platz für die Implementierung weitere Funktionen vorhanden.

Die mit Matlab erstellte Software erfüllt ihren Zweck. Sie nimmt die Daten auf und speichert diese lokal ab. Die Messdaten können dann mit Hilfe der Software ausgewertet werden.

Die Installation an einem Bioreaktor verlief auch erfolgreich. Wie bereits erwähnt wurden 16 Sonden platziert. 8 im Reaktor und 8 ausserhalb des Reaktors. Die aufgebaute Messkette funktioniert auch wie sie soll und ist fähig während 3 Tagen ohne Unterbruch Daten aufzuzeichnen.

Für die Erstellung eines thermischen Modells des Bioreaktors blieb zu wenig Zeit. Die Messungen, mit welchen dann das Modell validiert werden könnte wurden jedoch gemacht und dokumentiert. Zur vollständigen Validierung eines Modells müssten allerdings noch mehr Messungen durchgeführt werden. So wäre eine Durchflussmessung des Waser des Wärmetauscher-Kreislaufs sehr hilfreich.

Während der Projektarbeit ergaben sich neue Ideen und Probleme welche in einer nächsten Version verbessert werden können. Um was für Verbesserungen es sich dabei handelt wird im folgenden Kapitel gezeigt.

9.2 Probleme und Verbesserungsmöglichkeiten

Hier soll erklärt werden wie welche Probleme korrigiert werden können. Die Korrekturen wurden auf Grund Zeitmangels nicht mehr durchgeführt.

Zudem sollen Ideen, welche im späteren Verlauf des Projektes entstanden sind, notiert werden, damit diese später bei Bedarf umgesetzt werden können.

9.2.1 Temperaturkennlinie

Wie unter *2.1.3 Temperaturmessung* gezeigt ist die reale Kennlinie des PT1000-Widerstandsthermometers nicht linear. Da es sich aber um eine fast lineare Kennlinie handelt wurde hier während des Projektes nichts unternommen.

Eine Korrektur dieser Nichtlinearität könnte durch die Anpassung der Konstantstromquelle durchgeführt werden. Hierzu müsste ein negativer Ausgangswiderstand erstellt werden.

Wie dies gemacht werden kann wird im Anhang unter *14.3.3 Berechnung der Stromquelle* erklärt. Ein mathematischer Beweis der Korrektur der Kennlinie ist nicht vorhanden.

9.2.2 Schaltung der Erweiterungskarte

Die eigentliche Schaltung der Temperaturmessung könnte durch eine andere Verbindung zum den Sensoren verbessert werden. Mit der aktuellen vorhandenen Verbindung wird der Leitungswiderstand mit gemessen. Dies wurde zwar in der Software korrigiert. Trotzdem gibt es noch Fehler, da der Leitungswiderstand aufgrund nicht identischer Leitungslängen nicht überall identisch ist. Zudem ist der Leitungswiderstand auch von der Temperatur abhängig. Dies wurde auch nicht beachtet.

Mit einem Anschluss mit 4 Leitern könnte dieser Fehler behoben werden. Die Abbildung 9-1 zeigt dieses Prinzip.

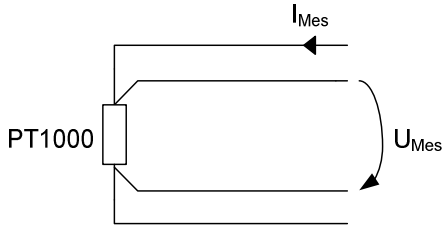


Abbildung 9-1: Verbindung zum Sensor mit 4 Leiter

Diese Änderung würde eine Anpassung der Schaltung des Anschlusses des Sensors erfordern.

Des Weiteren könnten dann die Leitungen, über welchen die Messspannung U_{Mes} übertragen wird noch geschirmt werden, um Störungen zu vermeiden.

Eventuell könnte durch eine bessere Wahl der Komponenten die Schaltung im Bezug auf den Störabstand verbessert werden.

9.2.3 Analog-Digital-Wandlung

Der A/D-Wandler erfüllt die Spezifikationen nicht ganz. Dies könnte durch die soeben erwähnten Korrekturen verbessert werden. Zudem könnte durch eine andere Konfiguration der Störabstand der A/D-Wandlung verbessert werden.

Mit der Konfiguration ist vor allem die Erhöhung des System-Takts des A/D-Wandlers von 1024kHz auf 4096kHz und die entsprechende Anpassung der Filterkonfigurationen gemeint.

Zudem könnte die im A/D-Wandler enthaltene Kalibrierung verwendet werden um den Offset-Fehler, wie auch einen möglichen Bereichsausschlagfehler zu korrigieren.

Konkret würde dies bedeuten, dass die FPGA-Schaltung entsprechend erweitert würde und eine bessere Konfiguration für den A/D-Wandler anhand des Datenblattes ausgewählt werden würde.

9.2.4 FPGA-Schaltung

Wie soeben erwähnt wurde, sollten die A/D-Wandler konfigurierbar sein. Dies würde eine Implementierung dieser Konfigurations-Möglichkeit in der FPGA erfordern. Um dies zu verwirklichen müsste der Weg der Daten von der USB-Schnittstelle zu den A/D-Wandlern fertig implementiert werden.

Die Schaltung könnte auch noch im Bezug auf die eigene Konfigurierbarkeit erweitert werden. Mit der eigenen Konfigurierbarkeit ist die Mittelung gemeint. Da die A/D-Wandler durch eine Erweiterung konfiguriert werden könnten, wäre ein internes Register erforderlich, welches eine entsprechende Anzahl der Werte, welche in die Berechnung des Mittelwertes einbezogen werden, enthalten würde. Die Anzahl der Werte ist derzeit mit Hilfe eines generischen Parameters festgelegt.

Um die Schaltung mit weniger Material synthetisieren zu können, wäre es möglich die Zähler, welche sich in den einzelnen Blöcken der parallelen Datenverarbeitung (dataShifter und chanMeanCalc) befinden, zu generalisieren. Dies bedeutet, dass die Zähler ausgelagert werden würden und so für alle Blöcke desselben Typs nur ein Zähler verwendet werden würde.

9.2.5 Offset

Woher das unter *6.3.1.3 Offset* beschrieben Offsetproblem stammt konnte nicht herausgefunden werden. Um eine Offset-Korrektur in der Software oder mit Hilfe der Konfiguration des A/D-Wandlers durchführen zu können müsste zuerst dessen Ursprung exakt definiert werden. Danach könnte die in den anderen Kapiteln der Verbesserungsvorschläge beschriebenen Lösungen angewendet werden um das Offset zu korrigieren.

9.2.6 Software

Damit die vorher Beschrienen Operationen zur Kalibrierung und Konfiguration der Messekette dann auch verwendet werden können müsste die Software entsprechend angepasst werden.

Diese Anpassungen würden eine Erweiterung des GUI und eine Erweiterung der Funktionalität erfordern.

Die würde konkret einen neuen Teil erfordern, welcher die Datenrate einstellen lässt. Zudem müsste ein neuer Programmteil zur Kalibrierung des Offsets vorgesehen werden. Dies zum Beispiel in Form einer Taste, wobei dann das Programm selber die gesamte Kalibrierung durchführen würde.

Interessant wäre auch eine Echtzeit-Darstellung der Daten. Die würde die Update Plot Operation erübrigen und würde dem Nutzer die Möglichkeit geben die Messung in Echtzeit zu betrachten.

9.2.7 Verbesserung der Installation

Wie unter *6.4 Feldmessungen* zu sehen war kann es durch die Kondensation trotz des Silikons, welches verwendet wurde, zu Kurzschlüssen kommen. Des Weiteren sind die angebrachten Sonden nicht sehr stabil.
Diese beiden negativen Eigenschaften könnten beispielsweise durch die Verwendung industriell hergestellter Sonden mit PT1000-Widerstandsthermometer korrigiert werden.

9.2.8 Median-Filterung von Ausschlägen

Der Medianfilter gleicht zurzeit eine Messung am Ende der Messkette aus. Dies entspricht der Ersetzung von 256 Messwerten.

Um diese Filterung zu Optimieren müsste die Dauer einer Störung der Temperaturmessung, also eines solchen Ausschlags, gemessen werden. Falls die Ausschläge kurz sind könnte sich ein Median-Filter in der FPGA befinden, damit nicht zu viel der gemessenen Werte überschrieben wird.

9.3 Perspektiven

Dieses Projekt wird zu einem späteren Zeitpunkt fortgesetzt. Mitarbeiter der Hes-so Valais / Wallis Sion werden das System korrigieren und optimieren. Dabei werden die Besprochenen Punkte korrigiert und verbessert. Die Messkette kann stets bei Fermentationen der Hes-so Valais / Wallis Sion getestet und ausgewertet werden.

Sobald das System soweit gereift ist, dass man es kommerziell vertreiben könnte, wäre es auch denkbar, dass das Projekt zum Produkt werden wird. Hierzu müsste allerdings, aus finanziellen Gründen, ein Auftrag aus der Industrie vorhanden sein. Der Preis einer Erweiterungskarte liegt bei den Prototypen bei rund 800CHF. Dieser Preis kann durch den Einkauf von grossen Stückzahlen der Komponenten und durch Verträge mit den Herstellern reduziert werden.

Zudem müsste das System nochmals umstrukturiert werden, da bei einem allfälligen Produkt keine FPGA-EBS-Karte dabei sein wird. Somit wäre eine neue Hardware für die FPGA nötig, was zusätzliche Kosten verursachen würde.

9.4 Schlusswort

Die Diplomarbeit erlaubte es viel des während des Studiums erhaltenen Wissens anzuwenden. Zudem war es sehr interessant ein Projekt mit einem gewissen Umfang durchzuführen. Das Projekt war durch seine Vielseitigkeit sehr interessant. Analoge und digitale Elektronik, Matlab-Programmierung usw. wurden studiert. Zudem reicht das Projekt bis hin zur Biotechnologie, was die Arbeit zusätzlich interessant machte.

Die während der Zusammenarbeit mit den Mitarbeitern der Hes-so Valais / Wallis Sion gesammelten Erfahrungen sind sicherlich wertvoll und bilden einen würdigen Abschluss des Studiums.

Sitten, 12.07.2010

10. Quellenverzeichnis

Die Form der Quellenangaben wird folgend gezeigt.

Autor der Quelle; *Titel der Quelle*; Ort an dem die Quelle vorzufinden ist; Datum der Herausgabe der Quelle; Datum der Nutzung der Quelle

[1] Texas Instruments; *Datenblatt des A/D-Wandlers ADS128*; http://focus.ti.com/lit/ds/symlink/ads1281.pdf; März 2009; März bis Juni 2010 (zuletzt 30.06.2010)

[2] Praplan Charles; *SEIS-AC-HEVs-PRTR-03*; Server der Hes-so unter I:\Institut\Projets\SEIS\Briding Phase\PCB\board1\ ; 08.10.2004

[3] Mathieu Schroeter; *Mesure de déplacement à haute résolution*; Archiv der Hes-so Valais / Wallis Sion; 30.10.2009; Mai und Juni 2010 (zuletzt 23.06.2010)

[4] Sartoretti Pascal; *FPGA-EBS BOARD*; Server der Hes-so unter P:\PCB\Produit\Logical\FPGA-EBS\doc; 08.04.2003; März und April 2010 (zuletzt 10.06.2010)

[5] The MathWorks; *Matlab-Dokumentation*; http://www.mathworks.com/access/helpdesk/help/techdoc/; -; Mai und Juni 2010 (zuletzt 28.06.2010)

[6] Oxford Complex Systems; *Skript des Median-Filters*; http://www.eng.ox.ac.uk/samp/members/max/software/; -; 21.06.2010

[8] Moerschell Joesph, Der ideale Operationsverstärker; Kurs Elekronik 2 Hes-so Valais / Wallis Sion; 04.04.2008; 17.03.2010

11. Abbildungsverzeichnis

12. Tabellenverzeichnis

13. Inhalt der CD-Rom

doc/	Enthält den Bericht mit den Anhängen.
datasheet/	Die bei der Entwicklung der Messkette, insbesondere des Schemas, benötigten Datenblätter der einzelnen Komponenten.
matlab/	Die mit Matlab erstellte Software und die benötigten Hilfsskripte.
measure/	Die Daten der durchgeführten Messungen in Form einer von Matlab (.mat) Dateien. Auch die im Bericht und Anhang gezeigten Abbildungen der Messungen sind vorhanden.
pcad/	Schemas und Layouts der erstellten Erweiterungskarte.
vhdl/	Das HDL-Designer Projekt, das Xilinx ISE Projekt und die Konfigurations-Dateien der FPGA und des PROM-Speichers sind vorhanden.

14. Anhang

14.1 Zwischenbericht

Diplomarbeiten 2010

Studiengang	Vertiefungsrichtung	Verantwortliche/r Dozent/in
Systemtechnik	*Infotronics*	*Joseph Moerschell*

Zwischenbericht der Diplomarbeit

Diplomand/in **Dominic Furrer**

TempArray
Thermische Bilanz durch verteilte Temperaturmessung

Hauptzielsetzungen

Mit Hilfe einer hochauflösenden Temperaturmesssung (<10^{-4}K) mit einer Reiher verteilter Sensoren soll die Wärmeabgabe einer industriellen Anlage, im Beispiel eines Bioreaktors, charakterisiert werden.

Die Arbeit hat folgende Ziele:
- Realisierung einer hochauflösenden Temperaturmesskette mit mindestens 4 Kanälen
- Installation der Sensoren an einem Bioreaktor und Inbetriebnahme der Messkette
- Validierung eines thermischen Modells des Reaktors während einer Fermentation

Ausgeführte Arbeiten

Da so bald als möglich eine funktionierende Hardware vorhanden sein sollte, begann man mit dem Studium eines ähnlichen Projektes. Bei dem studierten Projekt wurde auch eine hochauflösende Temperaturmessung ausgeführt. Man entschied sich für einen ähnlichen Aufbau. D.h. das Schema der Hardware wurde teilweise übernommen und angepasst. Die Schaltung umfasst vier Kanäle mit je einer Temperaturmessung mit einem PT1000 Sensor und je einem Delta-Sigma-Wandler. Man entschied sich hier für Delta-Sigma-Wandler von Texas Instruments, welche gute Störabstände und auch eine gute Linearität bieten. Neben den Kanälen wurde auch noch das Schema für die Versorgung des Boards erstellt. Dies beinhaltet eine Einspeisung und die Bereitstellung einer Referenzspannung.

Des Weiteren wurde auch schon früh entschieden, dass man für die Aufnahme der Messwerte und zur ersten Verarbeitung deren eine FPGA verwenden will. Zudem können so die bereits vorhandenen FPGA-Boards der Hes-so verwendet werden, um den Prototypen zu erstellen.
Aus dieser Entscheidung heraus ergab sich dann der letzte Teil des Schemas, welcher die Verbindung zwischen den Delta-Sigma-Wandlern und der FPGA herstellt.
Die schlussendliche Auswertung der Daten geschieht dann auf einem PC. Mit Hilfe der auf den FPGA-Boards vorhandenen USB-Schnittstelle wird über die USB Verbindung ein serieller Port simuliert. So ist es mit MatLab mit relativ geringem Aufwand möglich die Daten einzulesen und auszuwerten.

Die beschrieben Komponenten ergeben den in der folgenden Abbildung dargestellten prinzipiellen Aufbau des Systems.

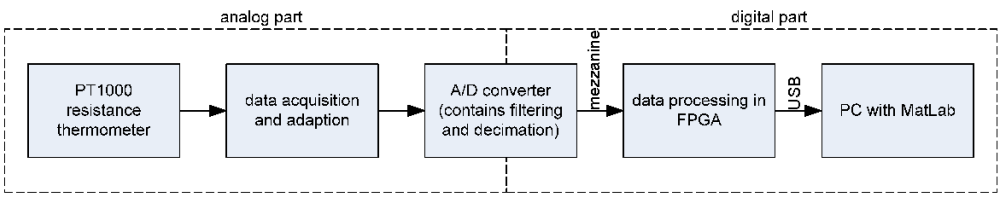


Abbildung 1: Aufbau des Systems

Sitten, 7. Mai 2010 Seite 1

Nachdem das Schema fertiggestellt war, übergab man dieses an das Atelier AE01. Die Mitarbeiter dieses Ateliers erstellten dann das Layout und führten die Bestellung des PCBs durch. Die PCBs werden von einer externen Firma namens EuroPCB hergestellt, da man für eine präzise Signalverarbeitung eine möglichst störungsunempfindliche Hardware braucht. Aus dem Grund entschied man sich für ein PCB mit vier Schichten. Wenn man ein störungsempfindliches Layout bzw. PCB verwenden würde, würde dies die hohe Genauigkeit und den grossen Störabstand, welcher durch die leistungsstarken Wandler vorhanden ist, wieder abschwächen. Zudem wurde noch ein Chassis-Ground hinzugefügt, welcher auch den Einfluss von Störungen bei der Messung mindern soll. Es wurden vier PCBs bestellt. Der Aufbau der PCBs ist in der folgenden Abbildung ersichtlich.

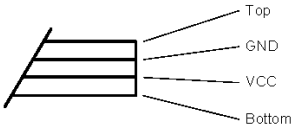


Abbildung 2: Aufbau des PCBs

Damit die Hardware dann auch hergestellt werden kann wurde noch eine Bestellliste erstellt, welche an die Mitarbeiter des Ateliers übergeben wurde.

Nach dem die Entwicklung der Hardware an einem Punkt angelangt war, an dem man auf die Lieferung der Komponenten warten muss, begann man mit der Konfiguration der FPGA. Diese soll die gemessenen Werte aufnehmen, anpassen und aus diesen Werten einen Mittelwert pro Sekunde berechnen. Dies geschieht parallel für alle vorhandenen Kanäle. Nach Anpassung und Mittelung der Werte werden diese per USB an den PC gesendet. Auf explizite Filter in der FPGA wurde vorläufig verzichtet, da in den Delta-Sigma-Wandlern bereits mehrere Filter (Sinc, FIR, IIR) vorhanden sind. Falls später weitere Filter benötigt werden, können diese immer noch erstellt werden.

Das System wurde für maximal 16 Kanäle konzipiert. Eine Erweiterungskarte, welche per Mezzanine-Verbindung mit der FPGA verbunden ist umfasst 4 Kanäle. So kann das System durch Erweiterungskarten mit je 4 Kanälen erweitert werden. D.h. man stapelt diese Karten aufeinander und stellt die Mezzanine-Verbindung her.
Bis anhin wurde ein nicht konfigurierbares System erstellt, welches die Daten nur ausliest und nicht verwertet. Dies wurde so gemacht, damit die Funktionalität des Systems bereits getestet werden kann. Allerdings reichte die Zeit nicht mehr für die Fertigstellung der FPGA-Konfiguration.

Probleme

Bei der Planung und Realisierung wurden bisher noch keine Probleme festgestellt.

Beim Auswerten der Messkette wird es dann allerdings schwierig sein die Genauigkeit zu überprüfen, da solch genaue Messgeräte nicht vorhanden sind. Allerdings kann die Qualität der Messkette anhand der Linearität und einer genauen Bestimmung des Verhaltens bei Störungen analysiert werden.

Auszuführende Arbeiten

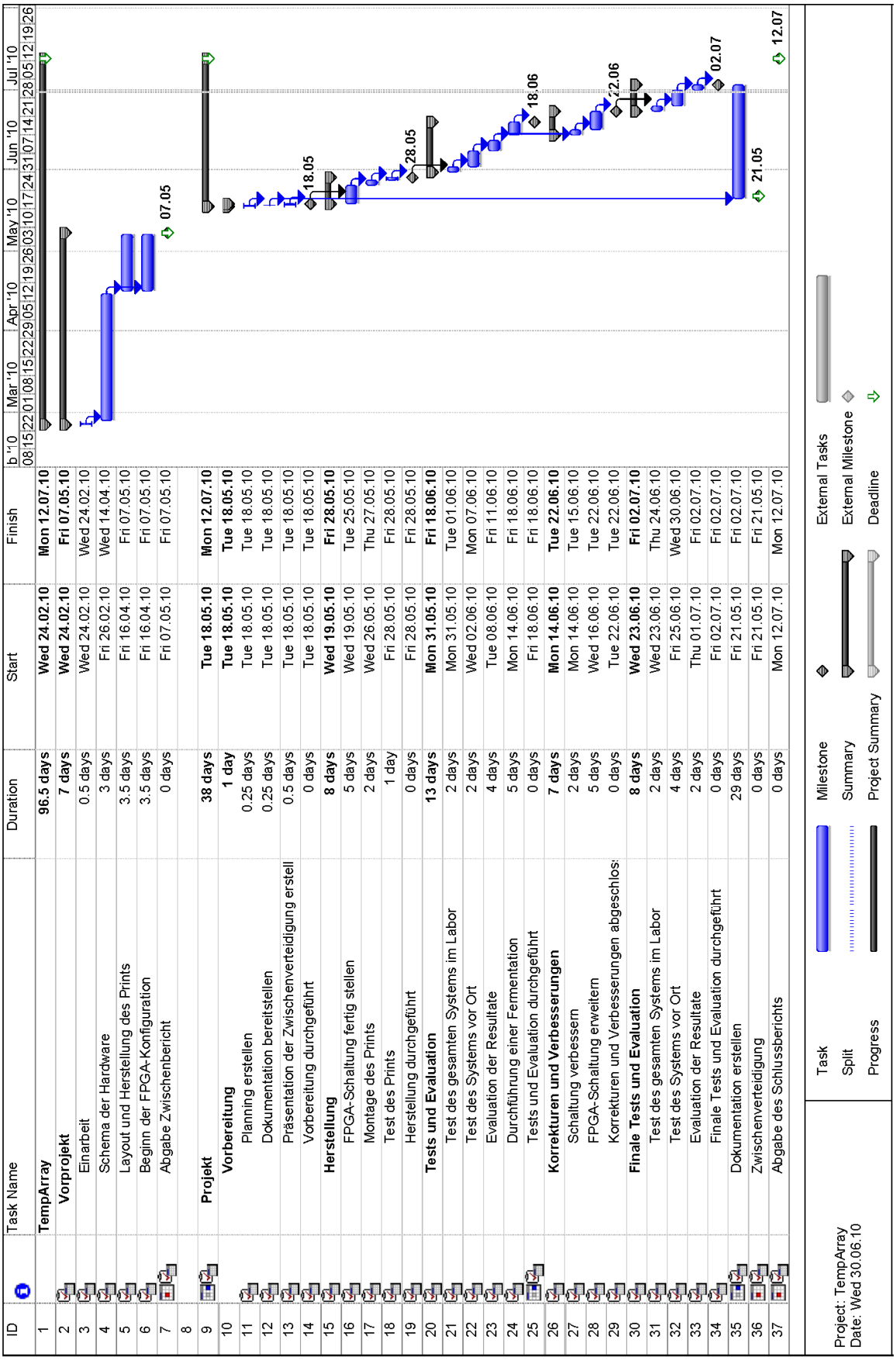
Der nächste Schritt beinhaltet die Fertigstellung und Simulation der FPGA-Konfiguration. Anschliessend wird die Montage der Hardware ausgeführt, Während der Herstellung werden erste Hardwaretests durchgeführt. Nach Abschluss der Hardwareherstellung werden alle grundlegen Funktionalitäten getestet. Sobald dies gemacht ist wird die Messkette als ganzen getestet. Hierzu muss noch die MatLab-Applikation erstellt werden.

[1)]Danach folgt eine erste Validation der Qualität des Systems. Diese Beurteilung wird allerdings in einer Laborumgebung stattfinden und wird dementsprechend lediglich Informationen, welche den idealen Laborbedingungen entsprechen, liefern.
Danach muss die Messkette an dem vorgesehen Ort installiert werden. Nach Fertigstellung der Installation kann eine weitere Validation an dem vorgesehen Ort durchgeführt werden.

Die Konfiguration der FPGA soll zu einem späteren Zeitpunkt noch erweitert werden. Man möchte, dass das System konfigurierbar, informativer, etc. wird.

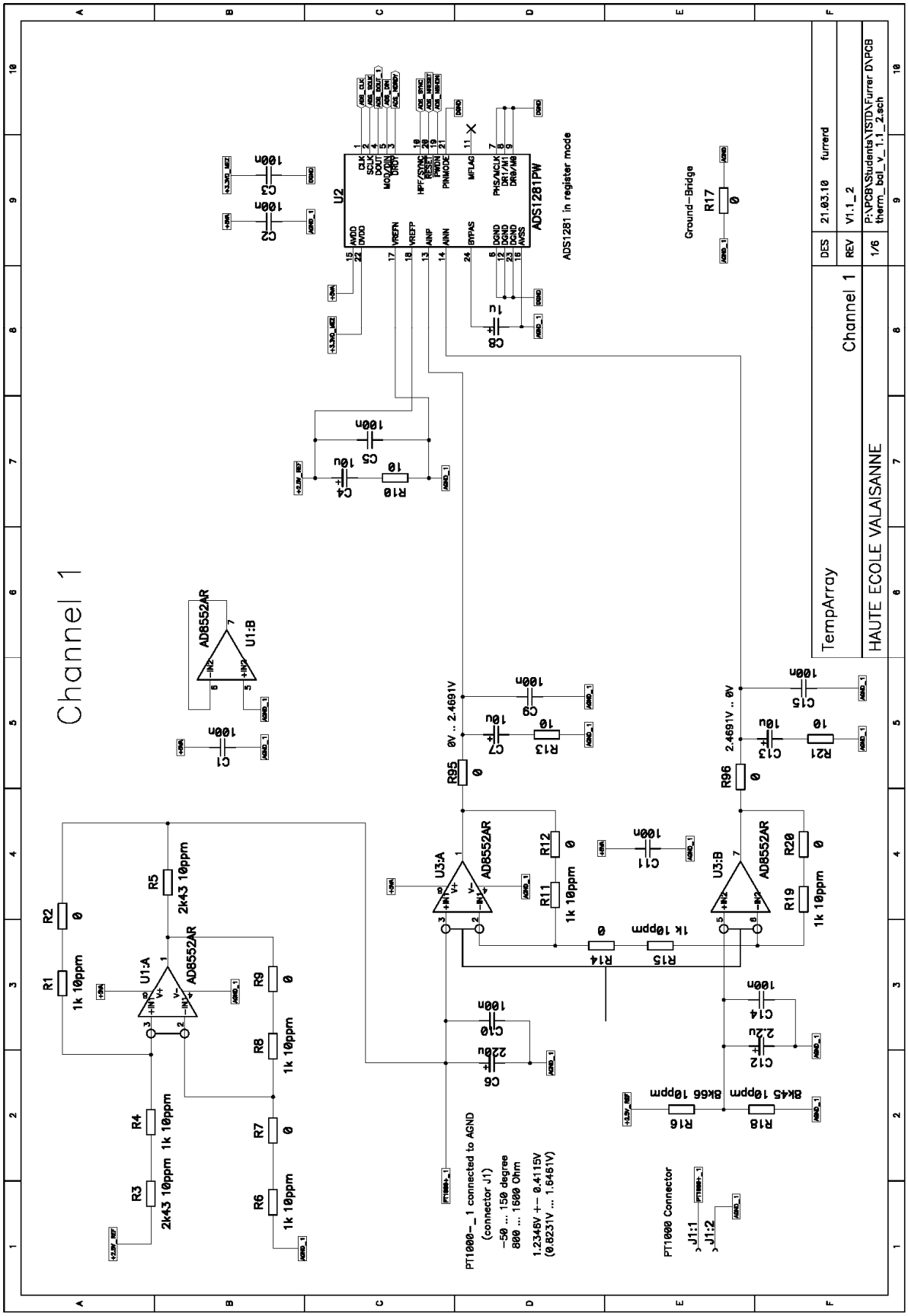
Nach der Erweiterung der FPGA müssen dann wieder dieselben Tests und Validationen vorgenommen werden wie vorhin. D.h. man muss erneut bei [1)] ansetzten und das gesamte Vorgehen wiederholen.

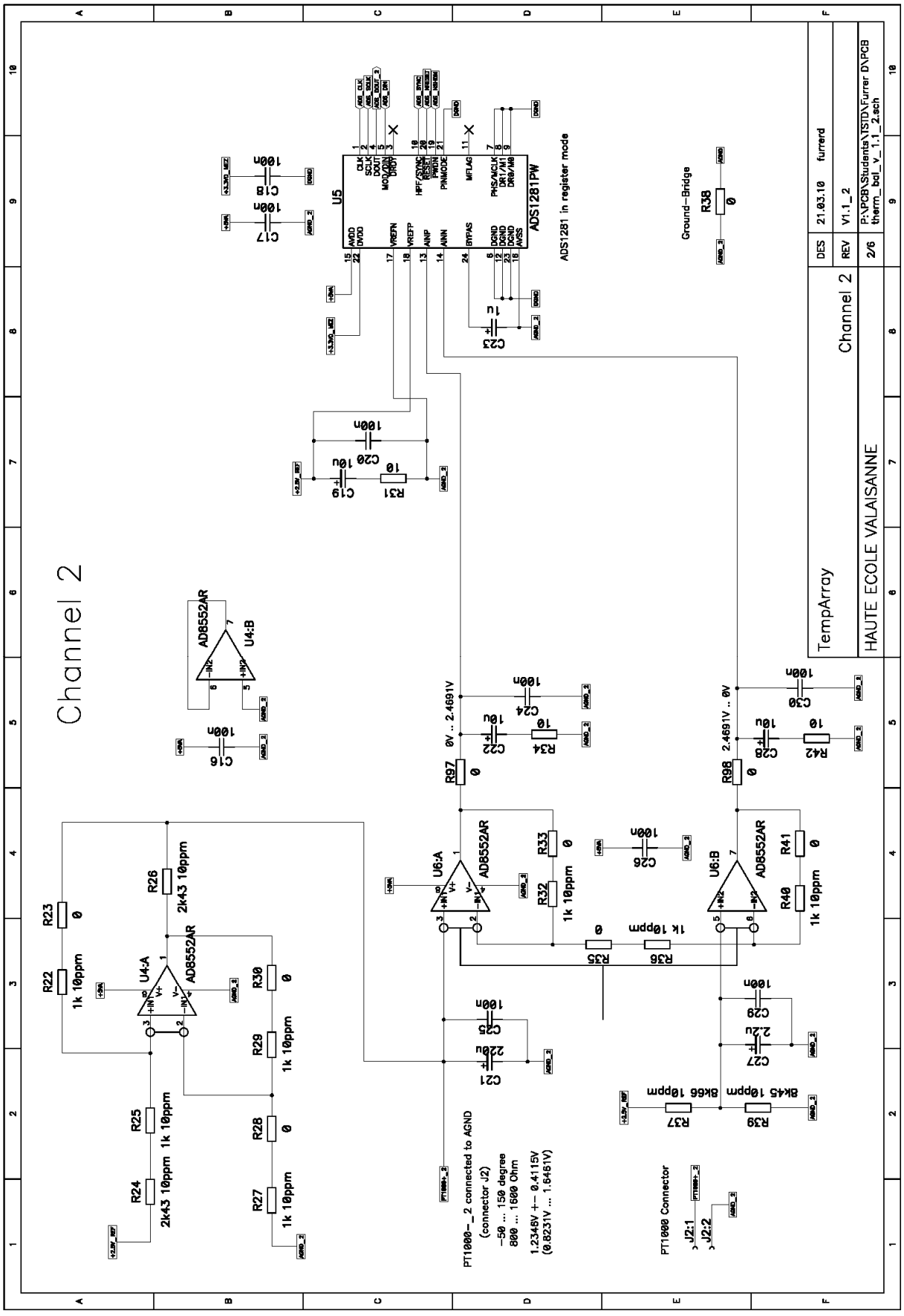
14.2 Planung

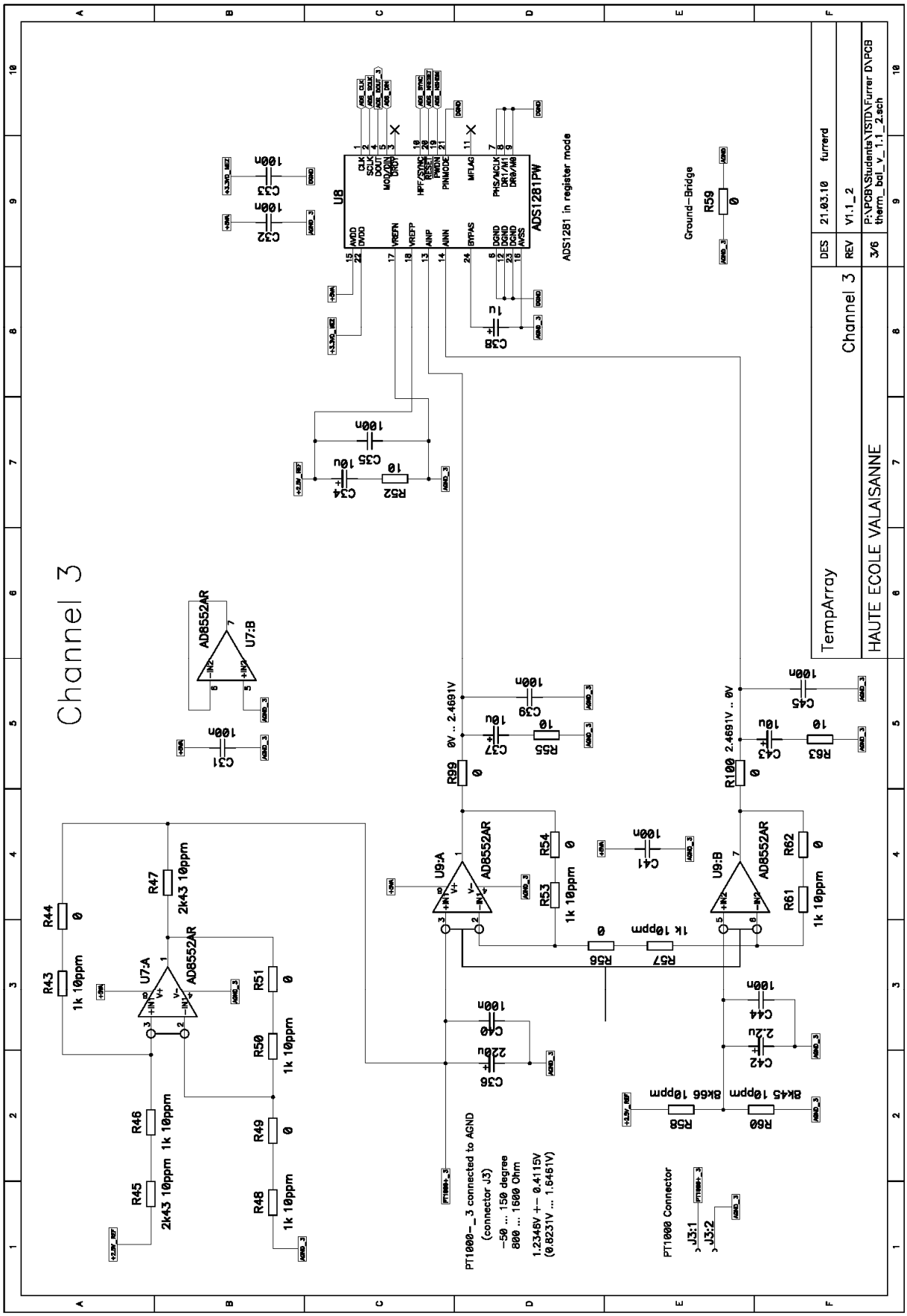

ID
Task Name
Duration
Start
Finish
1 TempArray 96.5 days Wed 24.02.10 Mon 12.07.10
2 Vorprojekt 7 days Wed 24.02.10 Fri 07.05.10
3 Einarbeit 0.5 days Wed 24.02.10 Wed 24.02.10
4 Schema der Hardware 3 days Fri 26.02.10 Wed 14.04.10
5 Layout und Herstellung des Prints 3.5 days Fri 16.04.10 Fri 07.05.10
6 Beginn der FPGA-Konfiguration 3.5 days Fri 16.04.10 Fri 07.05.10
7 Abgabe Zwischenbericht 0 days Fri 07.05.10 Fri 07.05.10
8
9 Projekt 38 days Tue 18.05.10 Mon 12.07.10
10 Vorbereitung 1 day Tue 18.05.10 Tue 18.05.10
11 Planning erstellen 0.25 days Tue 18.05.10 Tue 18.05.10
12 Dokumentation bereitstellen 0.25 days Tue 18.05.10 Tue 18.05.10
13 Präsentation der Zwischenverteidigung erstell 0.5 days Tue 18.05.10 Tue 18.05.10
14 Vorbereitung durchgeführt 0 days Tue 18.05.10 Tue 18.05.10
15 Herstellung 8 days Wed 19.05.10 Fri 28.05.10
16 FPGA-Schaltung fertig stellen 5 days Wed 19.05.10 Tue 25.05.10
17 Montage des Prints 2 days Wed 26.05.10 Thu 27.05.10
18 Test des Prints 1 day Fri 28.05.10 Fri 28.05.10
19 Herstellung durchgeführt 0 days Fri 28.05.10 Fri 28.05.10
20 Tests und Evaluation 13 days Mon 31.05.10 Fri 18.06.10
21 Test des gesamten Systems im Labor 2 days Mon 31.05.10 Tue 01.06.10
22 Test des Systems vor Ort 2 days Wed 02.06.10 Mon 07.06.10
23 Evaluation der Resultate 4 days Tue 08.06.10 Fri 11.06.10
24 Durchführung einer Fermentation 5 days Mon 14.06.10 Fri 18.06.10
25 Tests und Evaluation durchgeführt 0 days Fri 18.06.10 Fri 18.06.10
26 Korrekturen und Verbesserungen 7 days Mon 14.06.10 Tue 22.06.10
27 Schaltung verbessern 2 days Mon 14.06.10 Tue 15.06.10
28 FPGA-Schaltung erweitern 5 days Wed 16.06.10 Tue 22.06.10
29 Korrekturen und Verbesserungen abgeschlos: 0 days Tue 22.06.10 Tue 22.06.10
30 Finale Tests und Evaluation 8 days Wed 23.06.10 Fri 02.07.10
31 Test des gesamten Systems im Labor 2 days Wed 23.06.10 Thu 24.06.10
32 Test des Systems vor Ort 4 days Fri 25.06.10 Wed 30.06.10
33 Evaluation der Resultate 2 days Thu 01.07.10 Fri 02.07.10
34 Finale Tests und Evaluation durchgeführt 0 days Fri 02.07.10 Fri 02.07.10
35 Dokumentation erstellen 29 days Fri 21.05.10 Fri 02.07.10
36 Zwischenverteidigung 0 days Fri 21.05.10 Fri 21.05.10
37 Abgabe des Schlussberichts 0 days Mon 12.07.10 Mon 12.07.10
b '10 08 15 22
Mar '10 01 08 15 22
Apr '10 29 05 12 19
May '10 26 03 10 17 24 31
Jun '10 07 14 21 28
Jul '10 05 12 19 26
07.05
18.05
28.05
18.06
22.06
02.07
21.05
12.07
Project: TempArray
Date: Wed 30.06.10
Task
Split
Progress
Milestone
Summary
Project Summary
External Tasks
External Milestone
Deadline

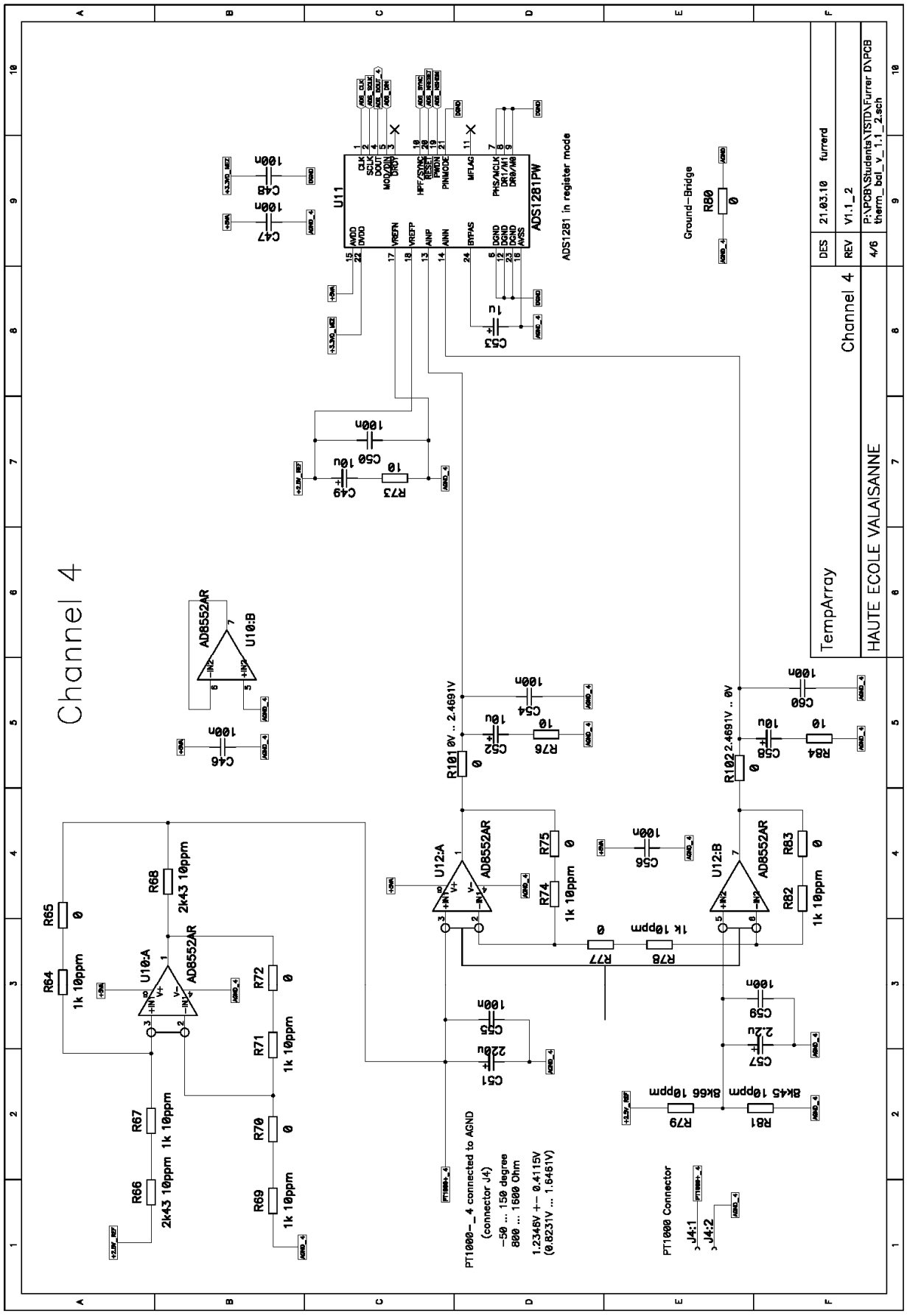
14.3 Schema

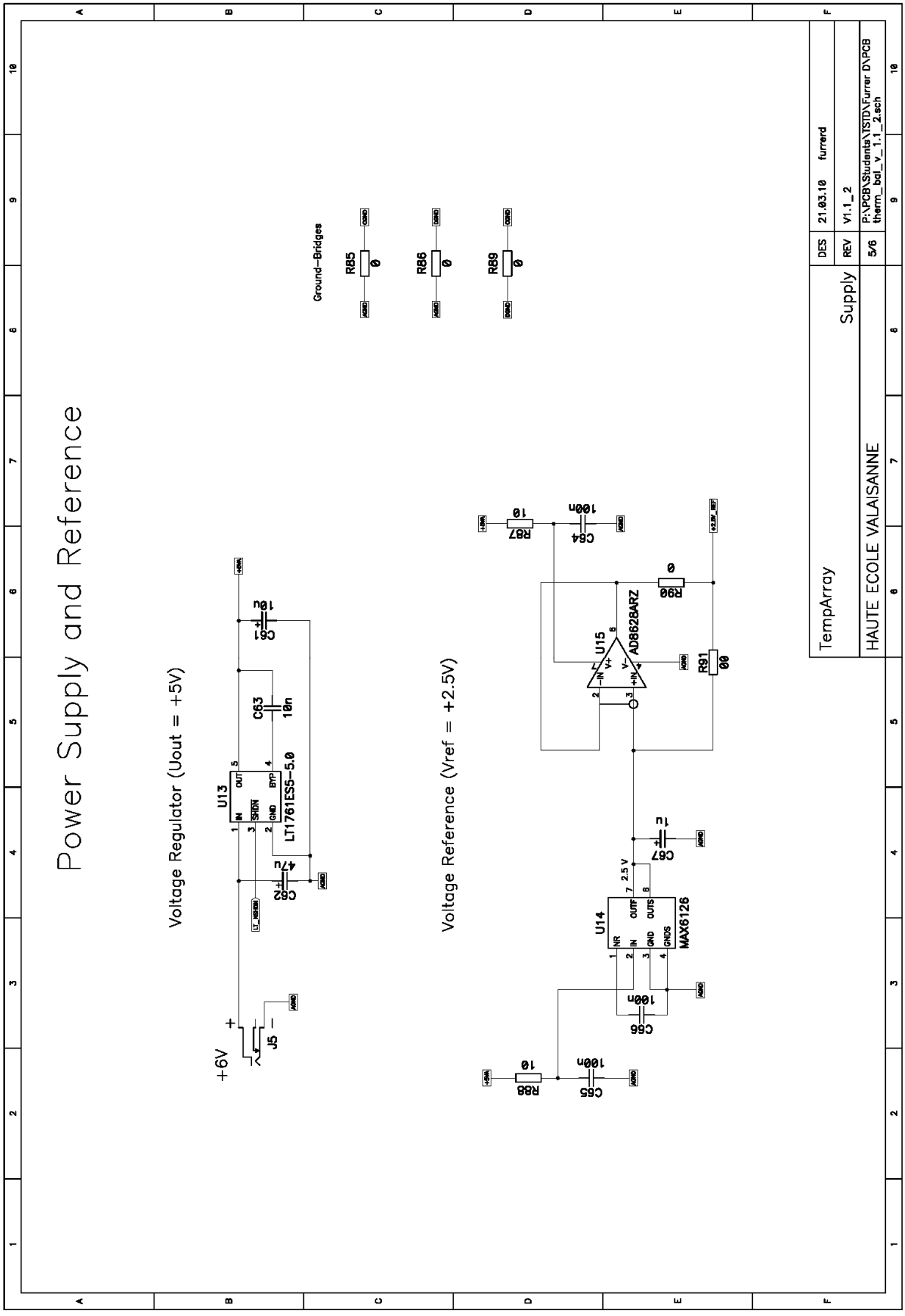
14.3.1 Elektrisches Schema der Erweiterungskarte (Version 1.1_2)

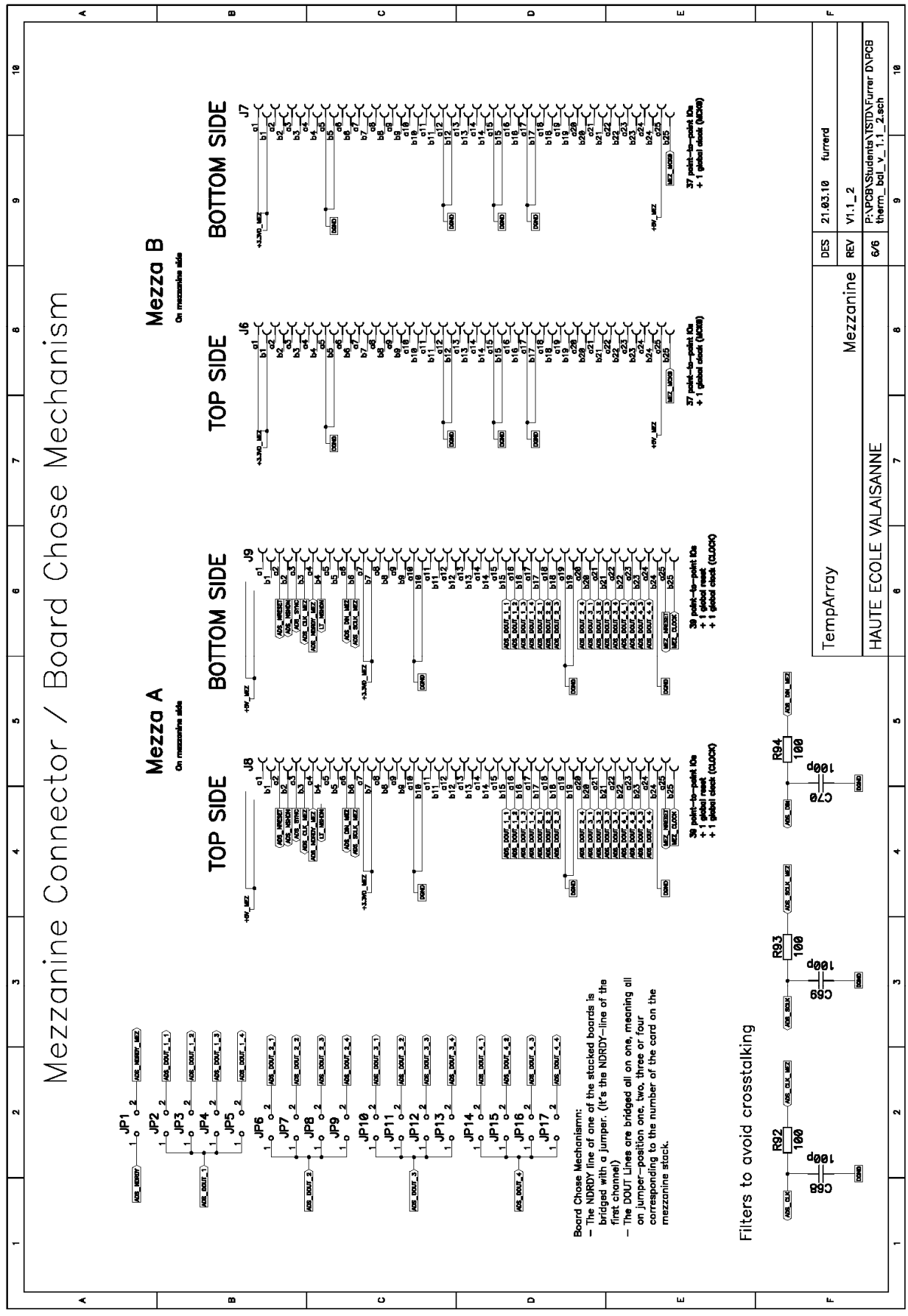



Channel 2
TempArray
HAUTE ECOLE VALAISANNE
Channel 2
ADS1281 in register mode
AD8552AR
PT1000 Connector
Ground-Bridge

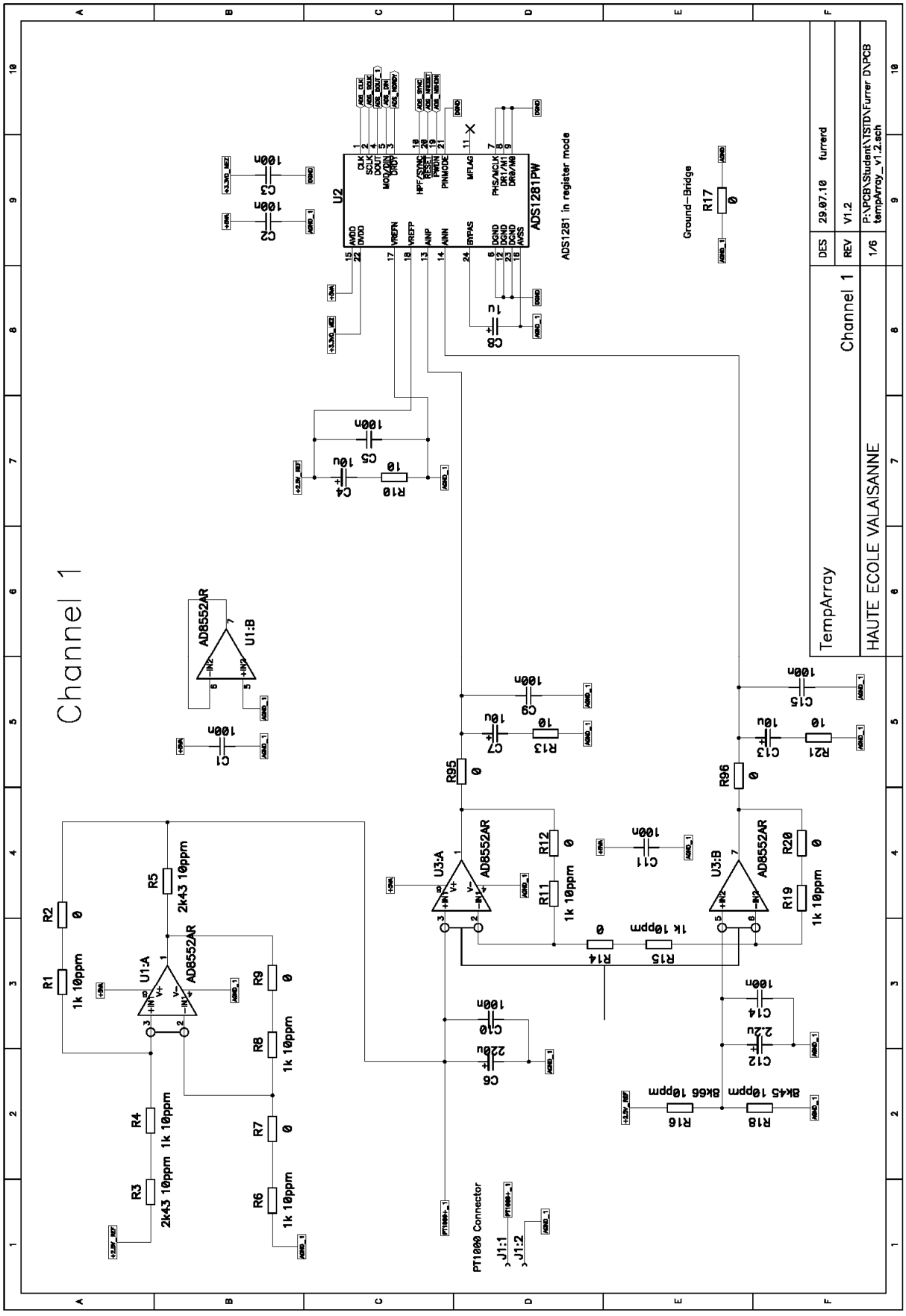

Channel 3
TempArray
Channel 3
HAUTE ECOLE VALAISANNE
DES 21.03.10 furrerd
REV V1.1_2
3/6 P:\PCB\Students\TSTD\Furrer D\PCB therm_bal_v_1.1_2.sch
U8
ADS1281PW
ADS1281 in register mode
U7:A
U7:B
U9:A
U9:B
AD8552AR
R43 1k 10ppm
R44 0
R45 2k43 10ppm
R46 1k 10ppm
R47 2k43 10ppm
R48 1k 10ppm
R49 0
R50 1k 10ppm
R51 0
R52 10
R53 1k 10ppm
R54 0
R55 10
R57 1k 10ppm
R58 8k45 10ppm
R59 0
R60 8k66 10ppm
R61 1k 10ppm
R62 0
R63 10
R99 0
R100 0
C31 100n
C32 100n
C33 100n
C34 10u
C35 100n
C36 22u
C37 10u
C38 1u
C39 100n
C40 100n
C41 100n
C42 2.2u
C43 10u
C44 100n
C45 100n
0V .. 2.4691V
2.4691V .. 0V
Ground-Bridge
PT1000_3 connected to AGND
(connector J3)
-50 ... 150 degree
800 ... 1600 Ohm
1.2346V +- 0.4115V
(0.8231V ... 1.6461V)
PT1000 Connector
J3:1
J3:2

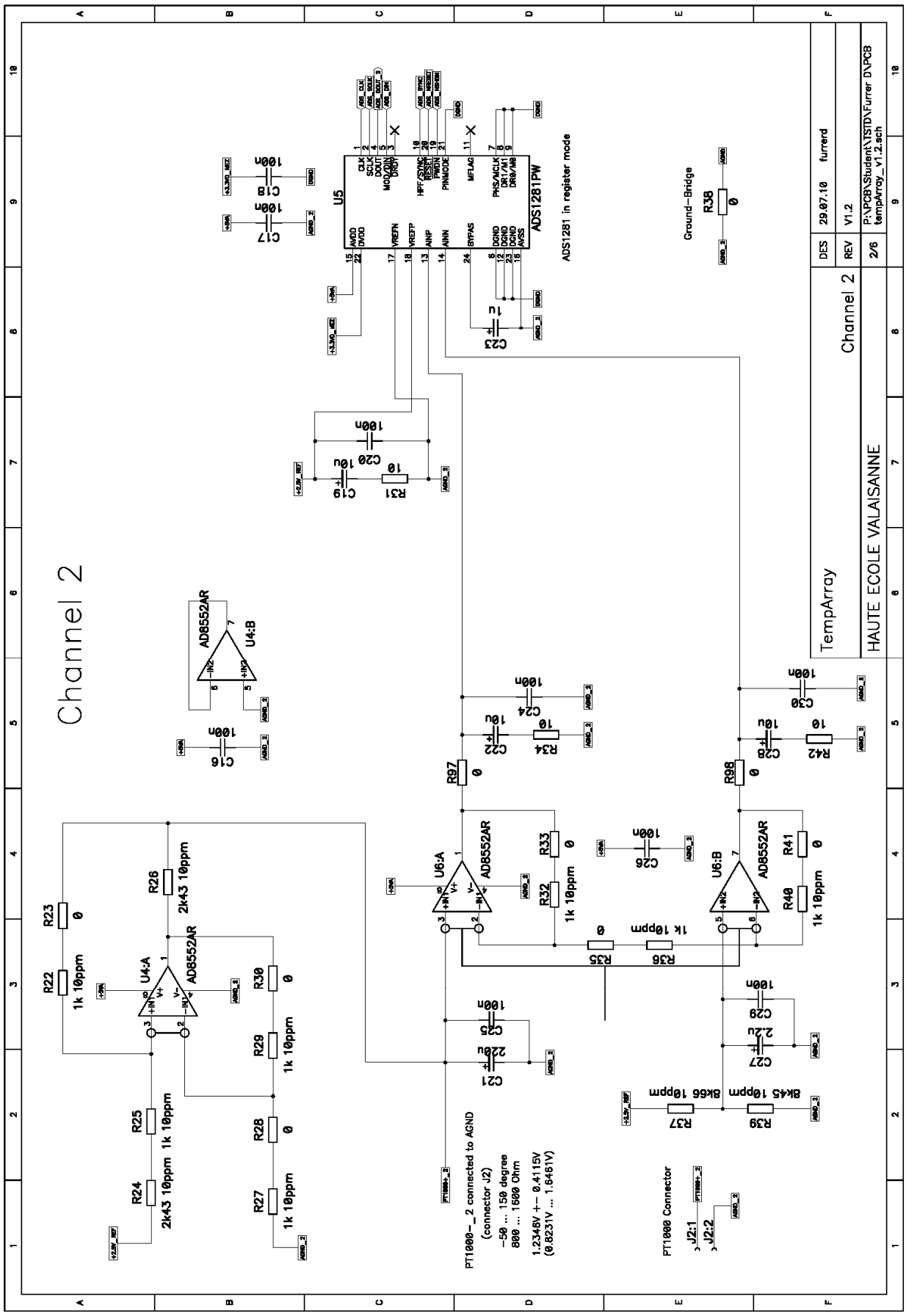

Channel 4
TempArray
Channel 4
HAUTE ECOLE VALAISANNE
DES 21.03.10 furrerd
REV V1.1_2
4/6
P:\PCB\Students\TSTD\Furrer D\PCB therm_bd_v_1.1_2.sch
U11
ADS1281PW
ADS1281 in register mode
U10:A
U10:B
U12:A
U12:B
AD8552AR
R65
R64
R68
2k43 10ppm
1k 10ppm
R72
R71
R70
R69
R67
R66
2k43 10ppm 1k 10ppm
R101 0V .. 2.4691V
R102 2.4691V .. 0V
R73
R74
R75
R76
R77
R78
1k 10ppm
R79
R81
R82
R83
R84
8k45 10ppm
8k66 10ppm
R80
Ground-Bridge
C46
C47
C48
C49
C50
C51
C52
C53
C54
C55
C56
C57
C58
C59
C60
100n
10u
22u
2.2u
PT1000-_4 connected to AGND
(connector J4)
-50 ... 150 degree
800 ... 1600 Ohm
1.2346V +- 0.4115V
(0.8231V ... 1.6461V)
PT1000 Connector
J4:1
J4:2

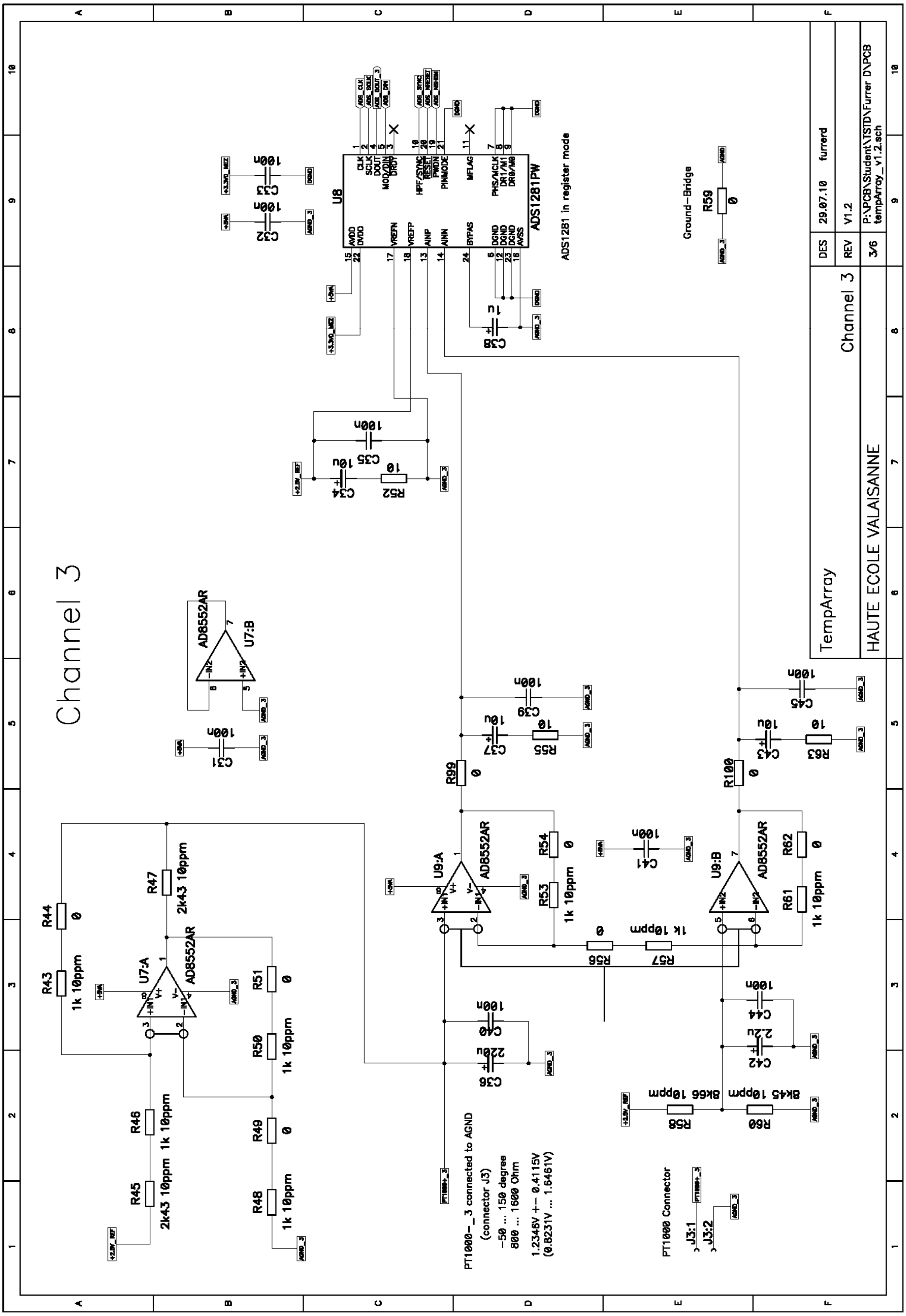

Power Supply and Reference
Voltage Regulator (Uout = +5V)
Voltage Reference (Vref = +2.5V)
Ground-Bridges
R85
0
R86
0
R89
0
U13
LT1761ES5-5.0
C61
10u
C63
10n
C62
4.7u
J5
+6V
U15
AD8628ARZ
R87
10
C64
100n
R90
0
R91
0
C67
1u
U14
MAX6126
2.5 V
C66
100n
R88
10
C65
100n
TempArray
Supply
HAUTE ECOLE VALAISANNE
DES 21.03.10 furrerd
REV V1.1_2
5/6
P:\PCB\Students\TSTD\Furrer D\PCB therm_bal_v_1.1_2.sch

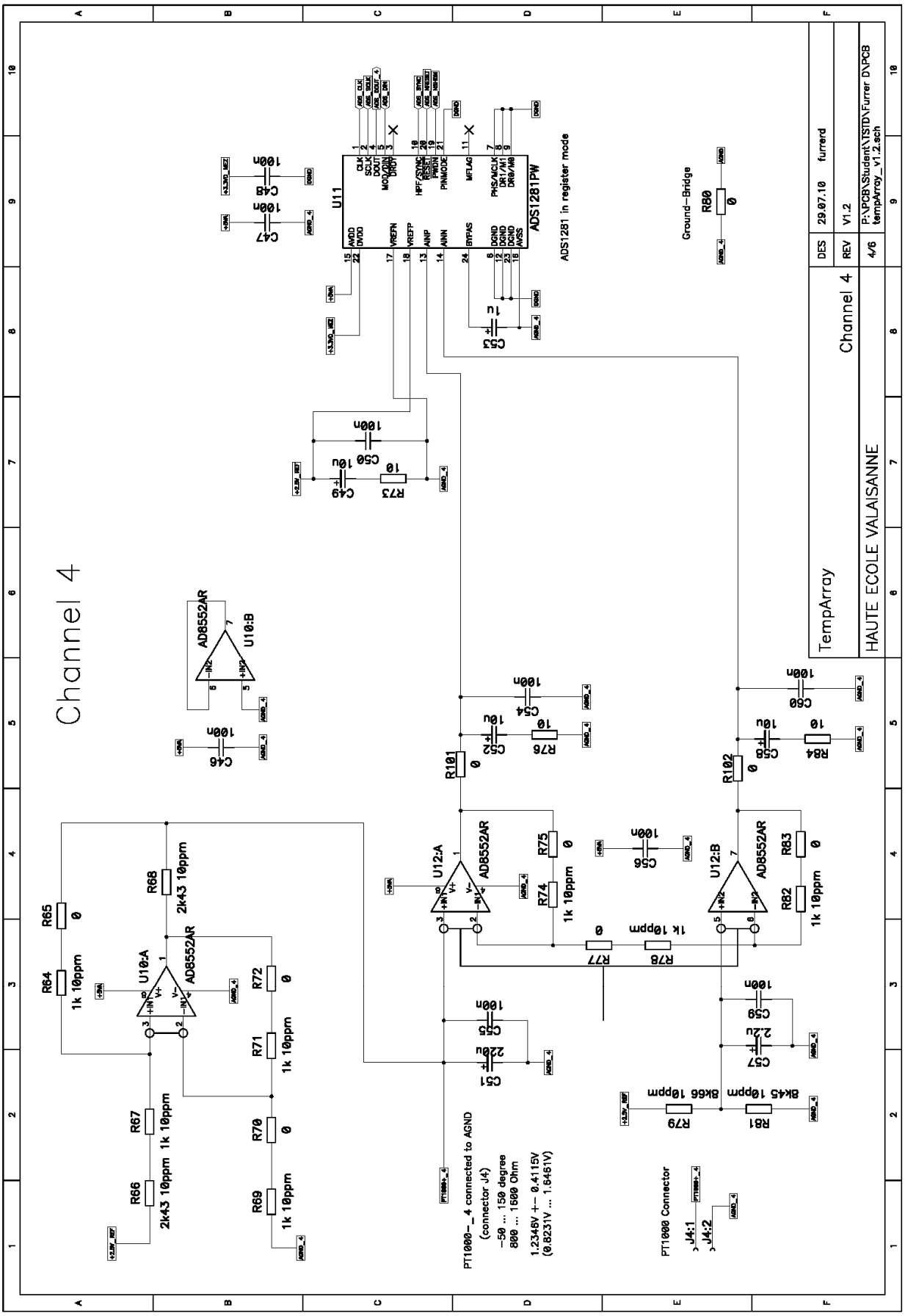

Mezzanine Connector / Board Chose Mechanism
Mezza A
On mezzanine side
Mezza B
On mezzanine side
TOP SIDE
BOTTOM SIDE
TOP SIDE
BOTTOM SIDE
J8
J9
J6
J7
JP1
JP2
JP3
JP4
JP5
JP6
JP7
JP8
JP9
JP10
JP11
JP12
JP13
JP14
JP15
JP16
JP17
R92
R93
R94
C68
C69
C70
100
100p
Board Chose Mechanism:
– The NDRDY line of one of the stacked boards is bridged with a jumper. (It's the NDRDY–line of the first channel)
– The DOUT Lines are bridged all on one, meaning all on jumper–position one, two, three or four corresponding to the number of the card on the mezzanine stack.
Filters to avoid crosstalking
TempArray
Mezzanine
HAUTE ECOLE VALAISANNE
DES 21.03.10 furrerd
REV V1.1_2
6/6
P:\PCB\Students\TSTD\Furrer D\PCB therm_bal_v_1.1_2.sch

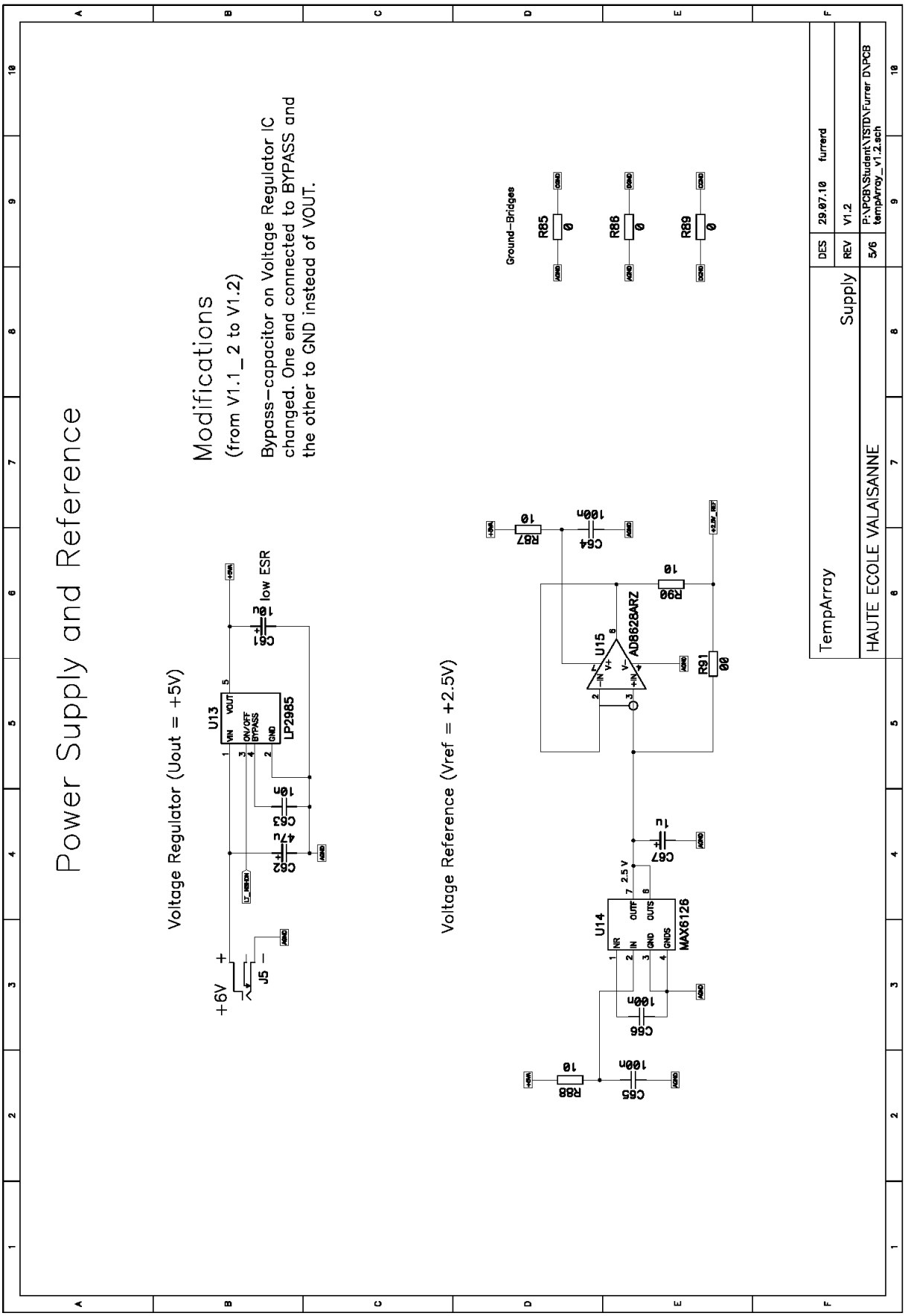
14.3.2 Elektrisches Schema der Erweiterungskarte (Version 1.2)

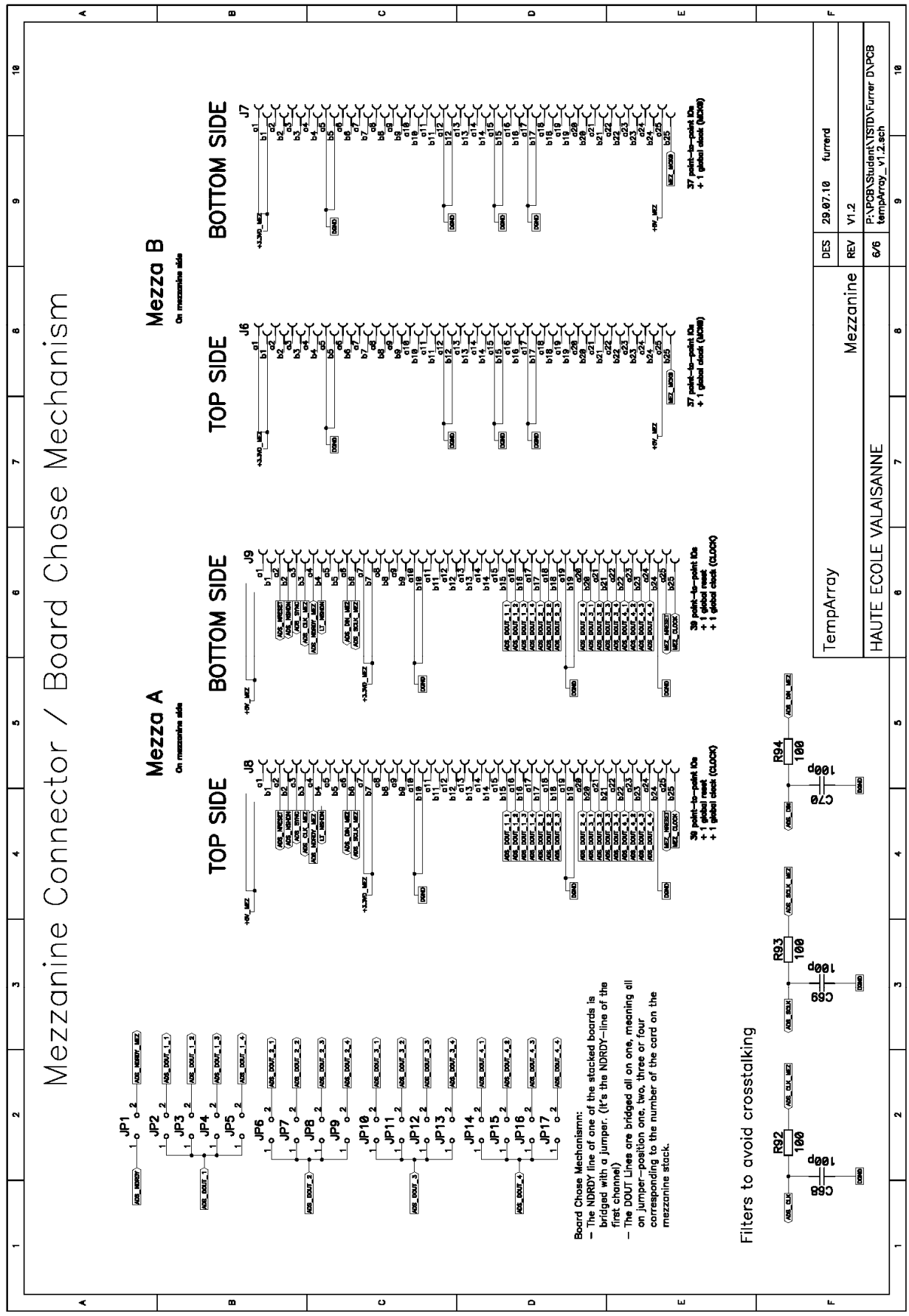



Channel 2
TempArray
Channel 2
HAUTE ECOLE VALAISANNE
DES 29.07.10 furrerd
REV V1.2
2/6
P:\PCB\Student\TSTD\Furrer DVPCB\tempArray_v1.2.sch
ADS1281 in register mode
Ground-Bridge
PT1000-_2 connected to AGND
(connector J2)
-50 ... 150 degree
800 ... 1600 Ohm
1.2346V +- 0.4115V
(0.8231V ... 1.6461V)
PT1000 Connector


Channel 3
TempArray
HAUTE ECOLE VALAISANNE
Channel 3
DES 29.07.10 furrerd
REV V1.2
3/6 P:\PCB\Student\TSTD\Furrer D\PCB tempArray_v1.2.sch
U8
ADS1281PW
ADS1281 in register mode
Ground-Bridge
R59 0
U7:A AD8552AR
U7:B AD8552AR
U9:A AD8552AR
U9:B AD8552AR
PT1000-_3 connected to AGND
(connector J3)
-50 ... 150 degree
800 ... 1600 Ohm
1.2346V +- 0.4115V
(0.8231V ... 1.6461V)
PT1000 Connector
J3:1
J3:2

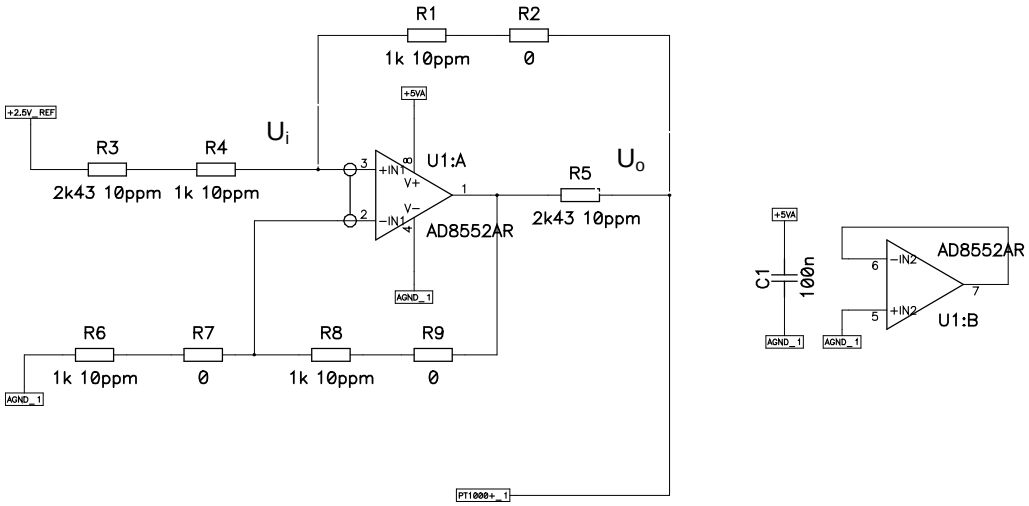

Channel 4
TempArray
Channel 4
HAUTE ECOLE VALAISANNE
DES 29.07.10 furrerd
REV V1.2
4/6
P:\PCB\Student\TSTD\Furrer D\PCB tempArray_v1.2.sch
U11
ADS1281PW
ADS1281 in register mode
U10:A
U10:B
U12:A
U12:B
AD8552AR
R64 1k 10ppm
R65 0
R66 2k43 10ppm
R67 1k 10ppm
R68 2k43 10ppm
R69 1k 10ppm
R70 0
R71 1k 10ppm
R72 0
R73 10
R74 1k 10ppm
R75 0
R76 10
R77 0
R78 1k 10ppm
R79 8k45 10ppm
R80 0
R81 8k66 10ppm
R82 1k 10ppm
R83 0
R84 10
R101 0
R102 0
C46 100n
C47 100n
C48 100n
C49 10u
C50 100n
C51 22u
C52 10u
C53 1u
C54 100n
C55 100n
C56 100n
C57 2.2u
C58 10u
C59 100n
C60 100n
PT1000-_ 4 connected to AGND
(connector J4)
-50 ... 150 degree
800 ... 1600 Ohm
1.2346V +- 0.4115V
(0.8231V ... 1.6461V)
PT1000 Connector
J4:1
J4:2
Ground-Bridge


Power Supply and Reference
Modifications
(from V1.1_2 to V1.2)
Bypass-capacitor on Voltage Regulator IC changed. One end connected to BYPASS and the other to GND instead of VOUT.
Voltage Regulator (Uout = +5V)
Voltage Reference (Vref = +2.5V)
Ground-Bridges
low ESR
LP2985
AD8628ARZ
MAX6126
TempArray
Supply
HAUTE ECOLE VALAISANNE
DES 29.07.10 furrerd
REV V1.2
5/6
P:\PCB\Student\TSTD\Furrer DVPCB\tempArray_v1.2.sch


Mezzanine Connector / Board Chose Mechanism
Mezza A
Mezza B
TOP SIDE
BOTTOM SIDE
Board Chose Mechanism:
– The NDRDY line of one of the stacked boards is bridged with a jumper. (It's the NDRDY–line of the first channel)
– The DOUT Lines are bridged all on one, meaning all on jumper–position one, two, three or four corresponding to the number of the card on the mezzanine stack.
Filters to avoid crosstalking
TempArray
Mezzanine
HAUTE ECOLE VALAISANNE
DES 29.07.10 furrerd
REV V1.2
6/6
P:\PCB\Student\TSTD\Furrer DNPCB\tempArray_v1.2.sch

14.3.3 Berechnung der Stromquelle

Stromquelle mit kleinem Verlust und negativem Ausgangswiderstand zur Korrektur der PT1000-Kennlinie.



Schema der Stromquelle

$R_\alpha = R_6 + R_7 = R_1 + R_2$

$R_\beta = R_4 = R_8 + R_9$

$R_{shunt} = R_{sh} = R_3 = R_5$

Die Knotenregel wird auf alle Knoten die nicht vom Operationsverstärker gespeist werden angewandt.

Knoten bei U_i: $0 = \frac{U_{in}-U_i}{R_\beta+R_{sh}} + \frac{U_{out}-U_i}{R_\alpha}$ 1)

Knoten bei U_o: $0 = \frac{U_o-U_{out}}{R_{sh}} + \frac{U_i-U_{out}}{R_\alpha} - I_{out}$ 2)

Knoten zwischen R_α und R_β (-IN1): $0 = -\frac{U_i}{R_\alpha} + \frac{U_o-U_i}{R_\beta}$ 3)

Mit 1) lässt sich U_i berechnen.

$$U_i\left(\frac{1}{R_\beta+R_{sh}} + \frac{1}{R_\alpha}\right) = \frac{U_{in}\cdot R_\alpha + U_{out}\cdot(R_\beta+R_{sh})}{R_\alpha\cdot(R_\beta+R_{sh})}$$

$$U_i\frac{R_\alpha+R_\beta+R_{sh}}{R_\alpha\cdot(R_\beta+R_{sh})} = \frac{U_{in}\cdot R_\alpha + U_{out}\cdot(R_\beta+R_{sh})}{R_\alpha\cdot(R_\beta+R_{sh})}$$

$$U_i = \frac{U_{in}\cdot R_\alpha + U_{out}\cdot(R_\beta+R_{sh})}{R_\alpha+R_\beta+R_{sh}} \quad 4)$$

Aus 3) und 4) ergibt sich U_o.

$$U_i\frac{R_\alpha+R_\beta}{R_\alpha\cdot R_\beta} = \frac{U_o}{R_\beta} \qquad U_o = U_i\frac{R_\alpha+R_\beta}{R_\alpha}$$

$$U_o = \frac{U_{in} \cdot R_\alpha + U_{out} \cdot (R_\beta + R_{sh})}{R_\alpha + R_\beta + R_{sh}} \cdot \frac{R_\alpha + R_\beta}{R_\alpha}$$

$$U_o = \frac{\left(U_{in} \cdot R_\alpha + U_{out} \cdot (R_\beta + R_{sh})\right) \cdot (R_\alpha + R_\beta)}{R_\alpha \cdot (R_\alpha + R_\beta + R_{sh})}$$

Mit Hilfe der Gleichung 2) lässt sich die Gleichung für den Ausgangsstrom I_{out} aufstellen.

$$I_{out} = \frac{U_o - U_{out}}{Rsh} + \frac{U_i - U_{out}}{R_\alpha}$$

$$I_{out} = \frac{\left(U_{in} \cdot R_\alpha + U_{out} \cdot (R_\beta + R_{sh})\right) \cdot (R_\alpha + R_\beta)}{R_\alpha \cdot Rsh \cdot (R_\alpha + R_\beta + Rsh)} - \frac{U_{out}}{Rsh} + \frac{U_{in} \cdot R_\alpha + U_{out} \cdot (R_\beta + R_{sh})}{R_\alpha \cdot (R_\alpha + R_\beta + Rsh)} - \frac{U_{out}}{R_\alpha}$$

$$I_{out} = U_{in} \frac{R_\alpha \cdot (R_\alpha + R_\beta) + R_\alpha \cdot R_{sh}}{R_\alpha \cdot Rsh \cdot (R_\alpha + R_\beta + R_{sh})} + U_{out} \frac{(R_\beta + R_{sh}) \cdot (R_\alpha + R_\beta) - R_\alpha \cdot (R_\alpha + R_\beta + R_{sh}) + R_{sh} \cdot (R_\beta + R_{sh}) - R_{sh} \cdot (R_\alpha + R_\beta + R_{sh})}{R_\alpha \cdot Rsh \cdot (R_\alpha + R_\beta + R_{sh})}$$

$$I_{out} = U_{in} \frac{R_\alpha + R_\beta + R_{sh}}{R_{sh} \cdot (R_\alpha + R_\beta + R_{sh})} + U_{out} \frac{R_\alpha \cdot R_\beta + {R_\beta}^2 + R_\alpha \cdot R_{sh} + R_\beta \cdot R_{sh} - {R_\alpha}^2 - R_\alpha \cdot R_\beta - R_\alpha \cdot R_{sh} + R_\beta \cdot R_{sh} + {R_{sh}}^2 - R_\alpha \cdot R_{sh} - R_\beta \cdot R_{sh} - {R_{sh}}^2}{R_\alpha \cdot R_{sh} \cdot (R_\alpha + R_\beta + R_{sh})}$$

$$I_{out} = Uin \frac{1}{R_{sh}} + U_{out} \frac{{R_\beta}^2 + R_\beta \cdot R_{sh} - {R_\alpha}^2 - R_\alpha \cdot R_{sh}}{R_\alpha \cdot R_{sh} \cdot (R_\alpha + R_\beta + R_{sh})}$$

$$I_{out} = \frac{Uin}{R_{sh}} + U_{out} \frac{{R_\beta}^2 - {R_\alpha}^2 + R_\beta \cdot R_{sh} - R_\alpha \cdot R_{sh}}{R_\alpha \cdot R_{sh} \cdot (R_\alpha + R_\beta + R_{sh})}$$

$$I_{out} = \frac{Uin}{R_{sh}} + U_{out} \frac{(R_\beta + R_\alpha) \cdot (R_\beta - R_\alpha) + R_{sh} \cdot (R_\beta - R_\alpha)}{R_\alpha \cdot R_{sh} \cdot (R_\alpha + R_\beta + R_{sh})}$$

$$I_{out} = \frac{Uin}{R_{sh}} + U_{out} \frac{(R_\beta + R_\alpha + R_{sh}) \cdot (R_\beta - R_\alpha)}{R_\alpha \cdot R_{sh} \cdot (R_\alpha + R_\beta + R_{sh})}$$

$$I_{out} = \frac{Uin}{R_{sh}} + U_{out} \frac{(R_\beta - R_\alpha)}{R_\alpha \cdot R_{sh}}$$

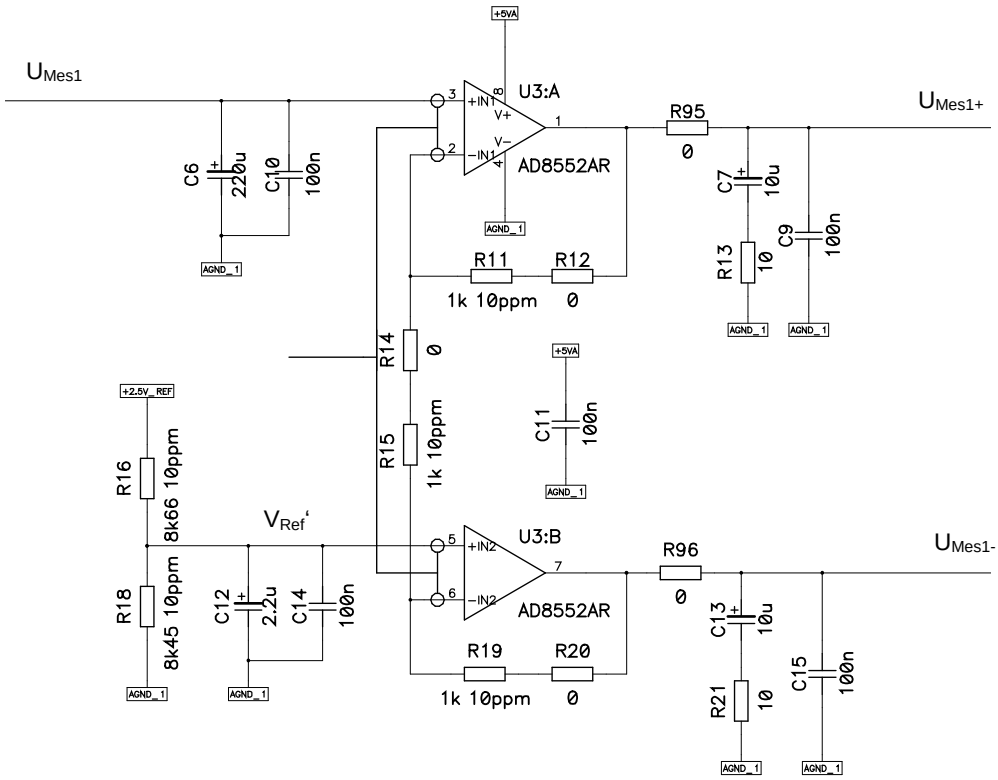
Es ist ersichtlich, dass I_{out} nur von R_{sh} abhängt, insofern $R_\alpha = R_\beta$ gilt.

$$I_{out} = \frac{V_{Ref}}{R_{sh}} = \frac{2.5V}{2.43k\Omega} = 1.029mA$$

Um die Kennlinie des PT1000-Widerstandsthermometers zu korrigieren wählt man R_α ein wenig grösser als R_β. So entsteht ein negativer Ausgangswiderstand. Wie dies die nicht lineare Kennlinie korrigieren kann wird hier nicht gezeigt.

14.3.4 Berechnung der Verstärkerstufe

Verstärkerstufe zur Anpassung der Spannung des gemessenen Signals.



Schema der Verstärkerstufe

Die Eingangsspannung U_{Mes1} ist die mit dem PT1000-Widerstandsthermometer erstellte Spannung.

Die zweite Eingangsspannung V_{Ref}‘ der Verstärkerstufe wird mit V_{Ref} und einem Spannungsteiler erstellt.

$$U_{In2} = V_{Ref} \cdot \frac{R_{18}}{R_{16} + R_{18}} = 2.5V \cdot \frac{8450\Omega}{8450\Omega + 8660\Omega} = 1.2347V$$

Die Gleichung der Ausgangsspannung des Operationsverstärkers mit U3:A ist die folgende.

$$U_{Mes1+} = U_{Mes1} \cdot \left(1 + \frac{R_{11} + R_{12}}{R_{14} + R_{15}}\right) - V_{Ref}{}' \cdot \frac{R_{11} + R_{12}}{R_{14} + R_{15}}$$

$$U_{Mes1+} = U_{Mes1} \cdot \left(1 + \frac{1k\Omega + 0\Omega}{1k\Omega + 0\Omega}\right) - V_{Ref}{}' \cdot \frac{1k\Omega + 0\Omega}{1k\Omega + 0\Omega}$$

$$U_{Mes1+} = 2 \cdot U_{Mes1} - V_{Ref}{}'$$

Die Gleichung der Ausgangsspannung des Operationsverstärkers mit U3:B ist die folgende.

$$U_{Mes1-} = V_{Ref}{}' \cdot \left(1 + \frac{R_{19} + R_{20}}{R_{14} + R_{15}}\right) - U_{Mes1} \cdot \left(\frac{R_{19} + R_{20}}{R_{14} + R_{15}}\right)$$

$$U_{Mes1-} = V_{Ref}{}' \cdot \left(1 + \frac{1k\Omega + 0\Omega}{1k\Omega + 0\Omega}\right) - U_{Mes1} \cdot \left(\frac{1k\Omega + 0\Omega}{1k\Omega + 0\Omega}\right)$$

$U_{Mes1-} = 2 \cdot V_{Ref}' - U_{Mes1}$

Die differentielle Ausgangsspannung der Verstärkerstufe kann nun berechnet werden.

$U_{Mes2} = U_{Mes1+} - U_{Mes1-} = \left(2 \cdot U_{Mes1} - V_{Ref}'\right) - \left(2 \cdot V_{Ref}' - U_{Mes1}\right)$

$U_{Mes2} = 3 \cdot U_{Mes1} - 3 \cdot V_{Ref}' = 3 \cdot \left(U_{Mes1} - V_{Ref}'\right)$

Dies ist die Spannungsgleichung am Eingang des A/D-Wandlers.

$U_{Mes2} = 3 \cdot (U_{Mes1} - 1.2347V) = 3 \cdot U_{Mes1} - 3.704V$

14.3.5 Berechnung des Stromverbrauch der Erweiterungskarte

		durchschnittlicher Verbrauch [mA]		maximaler Verbrauch [mA]	
Typ	**Anzahl**	**1 Komponent**	**Total**	**1 Komponent**	**Total**
ADS1281	4	2.6	10.4	3.8	15.2
AD8628	8	1	8	1.2	9.6
MAX6126	1	0.725	0.725	0.725	0.725
AD8551	1	1	1	1.075	1.075
Total			**20.125**		**26.6**

14.4 Datenblätter

14.4.1 LP2985

Übersicht

National Semiconductor

November 25, 2009

LP2985

Micropower 150 mA Low-Noise Ultra Low-Dropout Regulator in SOT-23 and micro SMD Packages *Designed for Use with Very Low ESR Output Capacitors*

General Description

The LP2985 is a 150 mA, fixed-output voltage regulator designed to provide ultra low-dropout and low noise in battery powered applications.

Using an optimized VIP® (Vertically Integrated PNP) process, the LP2985 delivers unequalled performance in all specifications critical to battery-powered designs:

Dropout Voltage: Typically 300 mV @ 150 mA load, and 7 mV @ 1 mA load.

Ground Pin Current: Typically 850 µA @ 150 mA load, and 75 µA @ 1 mA load.

Enhanced Stability: The LP2985 is stable with output capacitor ESR as low as 5 mΩ, which allows the use of ceramic capacitors on the output.

Sleep Mode: Less than 1 µA quiescent current when ON/OFF pin is pulled low.

Smallest Possible Size: SOT-23 and micro SMD packages use absolute minimum board space.

Precision Output: 1% tolerance output voltages available (A grade).

Low Noise: By adding a 10 nF bypass capacitor, output noise can be reduced to 30 µV (typical).

Multiple voltage options, from 2.5V to 5.0V, are available as standard products. Consult factory for custom voltages.

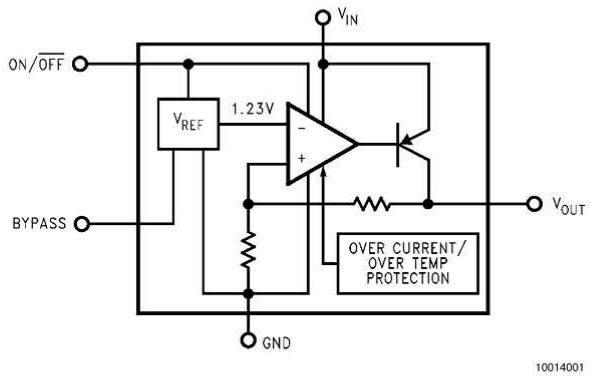
Features

- Ultra low dropout voltage
- Guaranteed 150 mA output current
- Smallest possible size (SOT-23, micro SMD package)
- Requires minimum external components
- Stable with low-ESR output capacitor
- <1 µA quiescent current when shut down
- Low ground pin current at all loads
- Output voltage accuracy 1% (A Grade)
- High peak current capability
- Wide supply voltage range (16V max)
- Low Z_{OUT}: 0.3Ω typical (10 Hz to 1 MHz)
- Overtemperature/overcurrent protection
- −40°C to +125°C junction temperature range
- Custom voltages available

Applications

- Cellular Phone
- Palmtop/Laptop Computer
- Personal Digital Assistant (PDA)
- Camcorder, Personal Stereo, Camera

Block Diagram



VIP® is a registered trademark of National Semiconductor Corporation.

LP2985 Micropower 150 mA Low-Noise Ultra Low-Dropout Regulator in SOT-23 and micro SMD Packages

Anwendungshinweise

LP2985

Application Hints

EXTERNAL CAPACITORS

Like any low-dropout regulator, the LP2985 requires external capacitors for regulator stability. These capacitors must be correctly selected for good performance.

Input Capacitor

An input capacitor whose capacitance is ≥ 1 µF is required between the LP2985 input and ground (the amount of capacitance may be increased without limit).

This capacitor must be located a distance of not more than 1 cm from the input pin and returned to a clean analog ground. Any good quality ceramic, tantalum, or film capacitor may be used at the input.

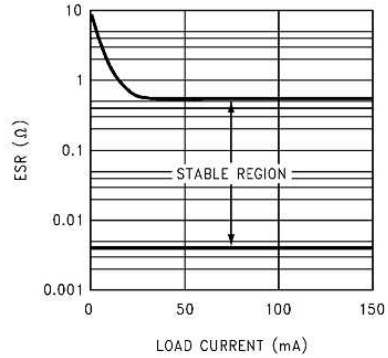
Important: Tantalum capacitors can suffer catastrophic failure due to surge current when connected to a low-impedance source of power (like a battery or very large capacitor). If a Tantalum capacitor is used at the input, it must be guaranteed by the manufacturer to have a surge current rating sufficient for the application.

There are no requirements for ESR on the input capacitor, but tolerance and temperature coefficient must be considered when selecting the capacitor to ensure the capacitance will be ≥ 1 µF over the entire operating temperature range.

Output Capacitor

The LP2985 is designed specifically to work with ceramic output capacitors, utilizing circuitry which allows the regulator to be stable across the entire range of output current with an output capacitor whose ESR is as low as 5 mΩ. It may also be possible to use Tantalum or film capacitors at the output, but these are not as attractive for reasons of size and cost (see next section Capacitor Characteristics).

The output capacitor must meet the requirement for minimum amount of capacitance and also have an ESR (equivalent series resistance) value which is within the stable range. Curves are provided which show the stable ESR range as a function of load current (see ESR graph below).



Important: The output capacitor must maintain its ESR within the stable region over the full operating temperature range of the application to assure stability.

The LP2985 requires a minimum of 2.2 µF on the output (output capacitor size can be increased without limit).

It is important to remember that capacitor tolerance and variation with temperature must be taken into consideration when selecting an output capacitor so that the minimum required amount of output capacitance is provided over the full operating temperature range. It should be noted that ceramic capacitors can exhibit large changes in capacitance with temperature (see next section, *Capacitor Characteristics*).

The output capacitor must be located not more than 1 cm from the output pin and returned to a clean analog ground.

Noise Bypass Capacitor

Connecting a 10 nF capacitor to the Bypass pin significantly reduces noise on the regulator output. It should be noted that the capacitor is connected directly to a high-impedance circuit in the bandgap reference.

Because this circuit has only a few microamperes flowing in it, any significant loading on this node will cause a change in the regulated output voltage. For this reason, DC leakage current through the noise bypass capacitor must never exceed 100 nA, and should be kept as low as possible for best output voltage accuracy.

The types of capacitors best suited for the noise bypass capacitor are ceramic and film. High-quality ceramic capacitors with either NPO or COG dielectric typically have very low leakage. 10 nF polypropolene and polycarbonate film capacitors are available in small surface-mount packages and typically have extremely low leakage current.

CAPACITOR CHARACTERISTICS

The LP2985 was designed to work with ceramic capacitors on the output to take advantage of the benefits they offer: for capacitance values in the 2.2 µF to 4.7 µF range, ceramics are the least expensive and also have the lowest ESR values (which makes them best for eliminating high-frequency noise). The ESR of a typical 2.2 µF ceramic capacitor is in the range of 10 mΩ to 20 mΩ, which easily meets the ESR limits required for stability by the LP2985.

One disadvantage of ceramic capacitors is that their capacitance can vary with temperature. Most large value ceramic capacitors (≥ 2.2 µF) are manufactured with the Z5U or Y5V temperature characteristic, which results in the capacitance dropping by more than 50% as the temperature goes from 25° C to 85°C.

This could cause problems if a 2.2 µF capacitor were used on the output since it will drop down to approximately 1 µF at high ambient temperatures (which could cause the LM2985 to oscillate). If Z5U or Y5V capacitors are used on the output, a minimum capacitance value of 4.7 µF must be observed.

A better choice for temperature coefficient in ceramic capacitors is X7R, which holds the capacitance within ±15%. Unfortunately, the larger values of capacitance are not offered by all manufacturers in the X7R dielectric.

Tantalum

Tantalum capacitors are less desirable than ceramics for use as output capacitors because they are more expensive when comparing equivalent capacitance and voltage ratings in the 1 µF to 4.7 µF range.

Another important consideration is that Tantalum capacitors have higher ESR values than equivalent size ceramics. This means that while it may be possible to find a Tantalum capacitor with an ESR value within the stable range, it would have to be larger in capacitance (which means bigger and more costly) than a ceramic capacitor with the same ESR value.

It should also be noted that the ESR of a typical Tantalum will increase about 2:1 as the temperature goes from 25°C down to –40°C, so some guard band must be allowed.

14.4.2 ADS1281

Übersicht

TEXAS INSTRUMENTS

ADS1281

www.ti.com SBAS378C – AUGUST 2007 – REVISED MARCH 2009

High-Resolution Analog-to-Digital Converter

FEATURES

- **High Resolution:**
  - **130dB SNR (250SPS)**
  - **127dB SNR (500SPS)**
- **High Accuracy:**
  - **THD: –122dB (typ), –115dB (max)**
  - **INL: 0.6ppm**
- **Inherently Stable Modulator with Fast Responding Over-Range Detection**
- **Flexible Digital Filter:**
  - **Sinc + FIR + IIR (Selectable)**
  - **Linear or Minimum Phase Response**
  - **Programmable High-Pass Filter**
  - **Selectable FIR Data Rates:**
    - **250SPS to 4kSPS**
- **Filter Bypass Option**
- **Low Power Consumption:**
  - **Operating: 12mW**
  - **Shutdown: 10μW**
- **Calibration Engine for Offset and Gain Correction**
- **SYNC Input**
- **Analog Supply:**
  - **Unipolar (+5V) or Bipolar (±2.5V)**
- **Digital Supply: 1.8V to 3.3V**

APPLICATIONS

- **Energy Exploration**
- **Seismic Monitoring**
- **High-Accuracy Instrumentation**

DESCRIPTION

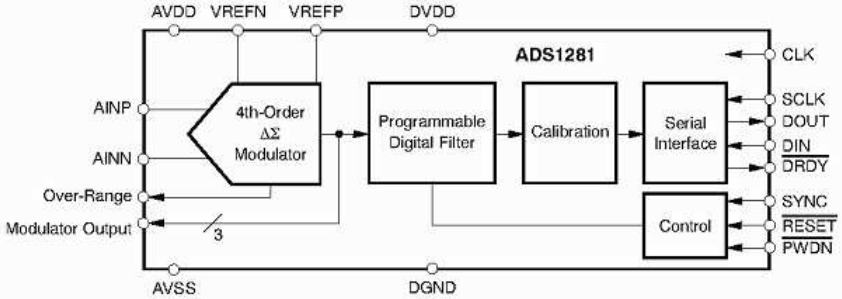
The ADS1281 is an extremely high-performance, single-chip analog-to-digital converter (ADC) designed for the demanding needs of energy exploration and seismic monitoring environments. The single-chip design promotes board area savings for improvements in high-density applications.

The converter uses a fourth-order, inherently stable, delta-sigma (ΔΣ) modulator that provides outstanding noise and linearity performance. The modulator is used either in conjunction with the on-chip digital filter, or can be bypassed for use with post-processing filters.

The digital filter consists of sinc and finite impulse response (FIR) low-pass stages followed by an infinite impulse response (IIR) high-pass filter (HPF) stage. Selectable decimation provides data rates from 250 to 4000 samples per second (SPS). The FIR low-pass stage provides both linear and minimum phase response. The HPF features an adjustable corner frequency. On-chip gain and offset scaling registers support system calibration.

The synchronization input (SYNC) can be used to synchronize the conversions of multiple ADS1281s. The SYNC input also accepts a clock input for continuous alignment of conversions from an external source.

Together, the modulator and filter dissipate only 12mW. The ADS1281 is available in a compact TSSOP-24 package and is fully specified from –40°C to +85°C, with a maximum operating range to +125°C.



Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

All trademarks are the property of their respective owners.

PRODUCTION DATA information is current as of publication date. Products conform to specifications per the terms of the Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

Kommunikation über SPI

READING DATA

The ADS1281 has two ways to read conversion data: Read Data Continuous and Read Data By Command.

Read Data Continuous

In the Read Data Continuous mode, the conversion data are shifted out directly from the device without the need for sending a read command. This mode is the default mode at power-on. This mode is also enabled by the RDATAC command. When $\overline{DRDY}$ goes low, indicating that new data are available, the MSB of data appears on DOUT, as shown in Figure 47. The data are normally read on the rising edge of SCLK and at the occurrence of the first falling edge of SCLK, $\overline{DRDY}$ returns high. After 32 bits of data have been shifted out, further SCLK transitions cause DOUT to go low. If desired, the read operation may be stopped at 24 bits. The data shift operation must be completed within four CLK periods before $\overline{DRDY}$ falls again or the data may be corrupted.

The Read Data Continuous mode is the default data mode for Pin mode. When a Stop Read Data Continuous command is issued, the $\overline{DRDY}$ output is blocked but the ADS1281 continues conversions. In stop continuous mode, the data can only be read by command.

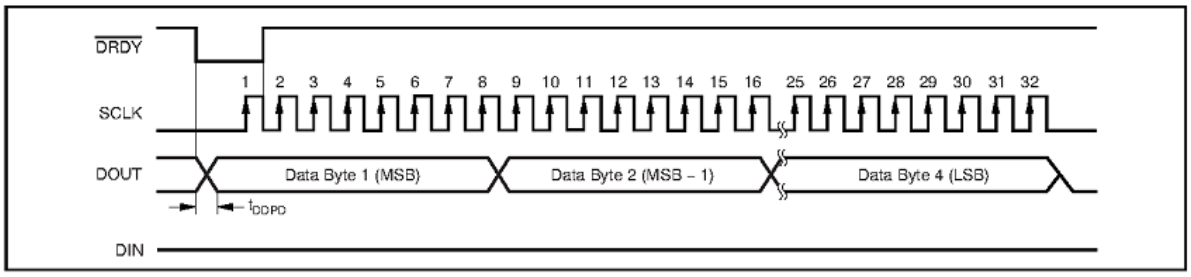


Figure 47. Read Data Continuous

Table 14. Timing Data for Figure 47

PARAMETER	DESCRIPTION	MIN	TYP	MAX	UNITS
t_{DDPD}	$\overline{DRDY}$ to valid MSB on DOUT propagation delay(1)			100	ns

(1) Load on DOUT = 20pF || 100kΩ.

14.5 FPGA

14.5.1 VHDL-Dateien des Blockes FPGA_tempArray

Architektur des Blockes DFF

```
--------------------------------------------------------------------
-- PROJECT: TempArray
--------------------------------------------------------------------
-- AUTHOR: Furrer Dominic
-- DATE: 21.04.2010
--------------------------------------------------------------------
-- FUNCTION: D-Flip-Flop used for synchronisation.
--------------------------------------------------------------------
ARCHITECTURE RTL OF DFF IS
BEGIN

   dff: process(CLK, CLR)
   begin
      if CLR = '1' then
         Q <= '0';
      elsif rising_edge(CLK) then
         Q <= D;
      end if;
   end process dff;

END ARCHITECTURE RTL;
```

Architektur des Blockes inverter

```
--------------------------------------------------------------------
-- PROJECT: TempArray
--------------------------------------------------------------------
-- AUTHOR: Furrer Dominic
-- DATE: 21.04.2010
--------------------------------------------------------------------
-- FUNCTION: Inverter used to adapt inputs and outputs logic.
--------------------------------------------------------------------
ARCHITECTURE RTL OF inverter IS
BEGIN
   out1 <= not(in1);
END ARCHITECTURE RTL;
```

Architektur des Blockes triBuffLogicV

```
--------------------------------------------------------------------
-- PROJECT: TempArray
--------------------------------------------------------------------
-- AUTHOR: Furrer Dominic
-- DATE: 21.04.2010
--------------------------------------------------------------------
-- FUNCTION: Tri-State-buffer used for buses etc.
--------------------------------------------------------------------
ARCHITECTURE RTL OF triBuffLogicV IS

BEGIN

   tristate: process(in1, oe)
   begin
      if oe = '1' then
         out1 <= To_StdLogicVector(in1);
      else
         out1 <= (others => 'Z');
      end if;
   end process tristate;

END ARCHITECTURE RTL;
```

14.5.2 Testbank der Simulation des Blockes tempArray

```
-------------------------------------------------------------------
-- PROJECT: TempArray
-------------------------------------------------------------------
-- AUTHOR: Furrer Dominic
-- DATE: 21.04.2010
-------------------------------------------------------------------
-- FUNCTION: Testbench that simulates the test conditions for the
--                    tempArray-Block.
-------------------------------------------------------------------
ARCHITECTURE test OF tempArray_tester IS

  type TestIntArrayType is array (3 downto 0) of integer;
  type TestUVArrayType is array (3 downto 0) of unsigned(23 downto 0);

  constant clockFrequency: real := 66.0E6;
  constant clockPeriod: time := (1.0/clockFrequency) * 1 sec;
  signal sClock: std_uLogic := '1';

  constant sampleClockPeriodNb2: positive := 528;
  signal sample : TestIntArrayType;
  signal sendSample: std_ulogic;
  signal sampleBinary : TestUVArrayType;

BEGIN
   ---------------------------------------------------------------------------
                                                          -- clock and reset
   sClock <= not sClock after clockPeriod/2;
   clock <= transport sClock after clockPeriod*9/10;
   reset <= '1', '0' after 2*clockPeriod;

   ---------------------------------------------------------------------------
                                                            -- test sequence
   process
   begin
            -- set up conditions
      sendSample <= '0';

      fifoDataUSB <= (others => 'Z');
      txeUSB <= '1';
             -- send data on the data lines
      wait for 300  us;

      for j in 1 to 200 loop

         sample(0) <= 16#123478#;
         sample(1) <= 16#7752A4#;
         sample(2) <= 16#287652#;
         sample(3) <= 16#1597AC#;
         sendSample <= '1', '0' after 1 ns;

         wait for 3 ms;

         sample(0) <= 16#15983D#;
         sample(1) <= 16#4AEF56#;
         sample(2) <= 16#100FE3#;
         sample(3) <= 16#32165A#;
         sendSample <= '1', '0' after 1 ns;

         wait for 3 ms;

         sample(0) <= 16#159DC1#;
         sample(1) <= 16#0ABCF0#;
         sample(2) <= 16#167935#;
         sample(3) <= 16#123122#;
         sendSample <= '1', '0' after 1 ns;
```

```
         wait for 3 ms;

         sample(0) <= 16#100F23#;
         sample(1) <= 16#167935#;
         sample(2) <= 16#7752D4#;
         sample(3) <= 16#1598BD#;
         sendSample <= '1', '0' after 1 ns;

         wait for 3 ms;

      end loop;

      wait;
   end process;

   ------------------------------------------------------------------------------
                                                                  -- bus serializer
   process
   begin

                   -- set condition and serilaize data for the data lines

      dataRdyADS <= '0';

      dataInADS00 <= '0';
      dataInADS01 <= '0';
      dataInADS02 <= '0';
      dataInADS03 <= '0';
      dataInADS10 <= '0';
      dataInADS11 <= '0';
      dataInADS12 <= '0';
      dataInADS13 <= '0';
      dataInADS20 <= '0';
      dataInADS21 <= '0';
      dataInADS22 <= '0';
      dataInADS23 <= '0';
      dataInADS30 <= '0';
      dataInADS31 <= '0';
      dataInADS32 <= '0';
      dataInADS33 <= '0';

      wait until rising_edge(sendSample);
      sampleBinary(0) <= to_unsigned(sample(0), sampleBinary(0)'length);
      sampleBinary(1) <= to_unsigned(sample(1), sampleBinary(1)'length);
      sampleBinary(2) <= to_unsigned(sample(2), sampleBinary(2)'length);
      sampleBinary(3) <= to_unsigned(sample(3), sampleBinary(3)'length);

      for index in sampleBinary(0)'range loop
         if index = sampleBinary(0)'high then
            dataInADS00 <= '0';
            dataInADS01 <= '0';
            dataInADS02 <= '0';
            dataInADS03 <= '0';
           dataRdyADS <= '1';
         else
            dataInADS00 <= sampleBinary(0)(index);
            dataInADS01 <= sampleBinary(1)(index);
            dataInADS02 <= sampleBinary(2)(index);
            dataInADS03 <= sampleBinary(3)(index);
            dataRdyADS <= '0';
         end if;


         wait for sampleClockPeriodNb2*clockPeriod;
      end loop;
   end process;

END ARCHITECTURE test;
```

14.5.3 VHDL-Dateien des Blockes clockDivider

```
----------------------------------------------------------------------
-- PROJECT: TempArray
----------------------------------------------------------------------
-- AUTHOR: Furrer Dominic
-- DATE: 21.04.2010
----------------------------------------------------------------------
-- FUNCTION: Creates the needed clocks for the ADCs.
----------------------------------------------------------------------
ARCHITECTURE RTL OF clockDivider IS

	signal sClkCount : unsigned(31 DOWNTO 0);
	signal clkCount : unsigned(31 DOWNTO 0);

BEGIN

	-- Create the system clock for the ADCs
	clkDiv: process(reset, clock)
	begin
		if reset = '1' then
			clkCount <= (others => '0');
		elsif rising_edge(clock) then
			if clkCount < clkDivider-1 then
				clkCount <= clkCount + 1;
			else
				clkCount <= (others => '0');
			end if;
		end if;
	end process clkDiv;

	-- Create the serial clock for the ADC-communication
	sClkDiv: process(reset, clock)
	begin
		if reset = '1' then
			sClkCount <= (others => '0');
		elsif rising_edge(clock) then
			if enableSClk = '1' then
				if sClkCount < sClkDivider-1 then
					sClkCount <= sClkCount + 1;
				else
					sClkCount <= (others => '0');
				end if;
			else
				sClkCount <= (others => '0');
			end if;
		end if;
	end process sClkDiv;

	-- Put out the generate clocks
	output: process(clkCount, sClkCount)
	begin
		if SHIFT_LEFT(clkCount,1) > clkDivider-1 then
			clk <= '1';
		else
			clk <= '0';
		end if;

		if SHIFT_LEFT(sClkCount,1) > sClkDivider-1 then
			sClk <= '1';
		else
			sClk <= '0';
		end if;
	end process output;
```

END ARCHITECTURE RTL;

14.5.4 Testbank der Simulation des Blockes clockDivider

```
-------------------------------------------------------------------
-- PROJECT: TempArray
-------------------------------------------------------------------
-- AUTHOR: Furrer Dominic
-- DATE: 21.04.2010
-------------------------------------------------------------------
-- FUNCTION: Testbench to test the clockDivider.
-------------------------------------------------------------------
library ieee;

ARCHITECTURE test OF clockDivider_tester IS

  constant clockFrequency: real := 66.0E6;
  constant clockPeriod: time := (1.0/clockFrequency) * 1 sec;
  signal sClock: std_uLogic := '1';

BEGIN
  ---------------------------------------------------------------------------
                                                        -- clock and reset
  sClock <= not sClock after clockPeriod/2;
  clock <= transport sClock after clockPeriod*9/10;
  reset <= '1', '0' after 2*clockPeriod;

  ---------------------------------------------------------------------------
                                                          -- test sequence
  process
  begin

  -- enable and disable sclock to test it
       enableSClk <= '0';
       wait for 10 us;
       enableSClk <= '1';
       wait for 100 us;
       enableSClk <= '0';
       wait for 100 us;

       wait;

  end process;

END ARCHITECTURE test;
```

14.5.5 VHDL-Dateien des Blockes ADSHandler

Block dataShifter

```
----------------------------------------------------------------------
-- PROJECT: TempArray
----------------------------------------------------------------------
-- AUTHOR: Furrer Dominic
-- DATE: 21.04.2010
----------------------------------------------------------------------
-- FUNCTION: This Block makes a 32 bit word out of the bitstream.
--                      Every time a dataRdy comes up, the last serialized
--                      value is put on the output.
----------------------------------------------------------------------
ARCHITECTURE RTL OF dataShifter IS

  signal oldSClk : std_ulogic;
  signal oldDataRdy : std_ulogic;
  signal data : std_ulogic_vector(dataBitNb-1 DOWNTO 0);
  signal counter : unsigned(6 downto 0); -- 6 bit counter: 0, 1, 2, ..., 30, 31,
32, 0, 1, etc.

BEGIN

  -- Main task. Shift the dataIn into the 32bit shiftregister
  dataShifter: process(clock, reset)
  begin
    if reset = '1' then
      data <= (others => '0');
      counter <= (others => '0');
    elsif rising_edge(clock) then
       if dataInRdy = '1' AND oldDataRdy = '0' then
          data <= (others => '0');
            counter <= (others => '0');
       else
          -- On every rising edge of sClk
          if sClk = '1' AND oldSClk = '0' then
              counter <= counter + 1;
            data(0) <= sDataIn;
            data(data'high downto 1) <= data(data'high-1 downto 0);
          end if;
          end if;
    end if;
  end process dataShifter;

       -- Put deserialized data on the output
       output: process(clock, reset)
       begin
          if reset = '1' then
            pDataOut <= (others => '0');
             elsif rising_edge(clock) then
             if counter >= dataBitNb then
                    pDataOut <= data;
                    dataOutRdy <= '1';
             else
                    dataOutRdy <= '0';
             end if;
      end if;
       end process output;

       -- Simple shift register for the rising edge detection
       dRdyShifter: process(clock, reset)
       begin
             if reset = '1' then
                    oldDataRdy <= '0';
             elsif rising_edge(clock) then
                    oldDataRdy <= dataInRdy;
             end if;
```

```
	end process dRdyShifter;

	-- Simple shift register for the rising edge detection
	sClkShifter: process(clock, reset)
	begin
		if reset = '1' then
			oldSClk <= '0';
		elsif rising_edge(clock) then
			oldSClk <= sClk;
		end if;
	end process sClkShifter;

END ARCHITECTURE RTL;
```

Block chanMeanCalc

```
-------------------------------------------------------------------
-- PROJECT: TempArray
-------------------------------------------------------------------
-- AUTHOR: Furrer Dominic
-- DATE: 21.04.2010
-------------------------------------------------------------------
-- FUNCTION: This block calculates the mean of one channel and
--                      informs the next block when the mean is calculated.
-------------------------------------------------------------------
LIBRARY TempArray;
USE TempArray.TempArrayPkg.all;

ARCHITECTURE RTL OF chanMeanCalc IS

	signal sum : unsigned(dataBitNb+8-1 downto 0);--
unsigned(dataBitNb+requiredBitNb(nbOfVals)-1 downto 0);
	signal intSumDone : std_ulogic;
	signal intDataIn : unsigned (dataBitNb-1 downto 0);
	signal counter : unsigned(requiredBitNb(nbOfVals)+1 downto 0);
	signal oldDataInRdy : std_ulogic;

BEGIN

   dataInput: process(reset, dataIn)
   begin
      if reset = '1' then
         intDataIn <= (others => '0');
      else
         intDataIn <= unsigned(dataIn);
      end if;
   end process dataInput;


	-- Sum the new value to the old on every dataInRdy pulse
	sumCalc: process(clock, reset)
	begin
		if reset = '1' then
			sum <= (others => '0');
		elsif rising_edge(clock) then
		   if dataInRdy = '1' AND oldDataInRdy = '0' then
				sum <= (sum + resize(intDataIn,sum'length));--sum <=
resize(intDataIn,sum'length);
		   elsif intSumDone = '1' then
				sum <= (others => '0');
			end if;
		 end if;
	end process sumCalc;
```

```
	-- Counter to simulate the sumDone, can be placed exterior later to reduce
material
	count: process(clock, reset)
	begin
		if reset = '1' then
			counter <= (others => '0');
			intSumDone <= '0';
		elsif rising_edge(clock) then
			if counter < nbOfVals then
				if dataInRdy = '1' AND oldDataInRdy = '0' then
					counter <= counter + 1;
				end if;
				intSumDone <= '0';
			else
				counter <= (others => '0');
				intSumDone <= '1';
			end if;
		end if;
	end process count;


	-- Set the output as sum is done
	output: process(clock, reset)
	begin
		if reset = '1' then
			dataOut <= (others => '0');
			dataOutRdy <= '0';
		elsif rising_edge(clock) then
		if intSumDone = '1' then
			dataOut <=
std_ulogic_vector(RESIZE(SHIFT_RIGHT(sum,requiredBitNb(nbOfVals)),dataBitNb));--
dataOut <= std_ulogic_vector(resize(sum,dataOut'length));
			dataOutRdy <= '1';
		else
			dataOutRdy <= '0';
		end if;
	      end if;
	end process output;


	-- Simple shift register for the rising edge detection
	dataRdyShifter: process(clock, reset)
	begin
		if reset = '1' then
			oldDataInRdy <= '0';
		elsif rising_edge(clock) then
			oldDataInRdy <= dataInRdy;
		end if;
	end process dataRdyShifter;


END ARCHITECTURE RTL;
```

Block meanDataHandler

```
-------------------------------------------------------------------
-- PROJECT: TempArray
-------------------------------------------------------------------
-- AUTHOR: Furrer Dominic
-- DATE: 21.04.2010
-------------------------------------------------------------------
-- FUNCTION: This Block handles the calculated mean data and
--                     saves the values into the FIFO-memory.
-------------------------------------------------------------------
ARCHITECTURE RTL OF meanDataHandler IS
```

```
       signal sel : integer;
       signal go : std_ulogic;
       signal counter : unsigned(((ADSDataHeadBitNb-ADSDataBitNb)-1) downto 0);
       signal oldMeanDataRdy : std_ulogic;
       signal delaying : std_ulogic;
   signal delayCnt : unsigned(6 downto 0);

BEGIN

       -- Chooses next data to be written, like a multiplexer
       getData: process(clock, reset)
       begin
          if reset = '1' then
             dataOut <= (others => '0');
             write <= '0';
          elsif rising_edge(clock) then
             if enable = '1' then
                write <= '0';
             if go = '1' then
                if delaying = '0' then
                                  dataOut(ADSDataBitNb-1 downto 0) <=
meanDataIn(sel);
                                  dataOut(ADSDataHeadBitNB-1 downto ADSDataBitNb) <=
std_ulogic_vector(counter);
                                  write <= '1';
                            end if;
             end if;
       end if;
       end if;
       end process getData;

       -- Counts from 0 to 15 to choose all channels
       count: process(reset, clock)
       begin
             if reset = '1' then
                   sel <= 0;
                   counter <= (others => '0');
             elsif rising_edge(clock) then
                if enable = '1' then
                   if go = '1' then
                      if delaying = '0' then
                          sel <= sel + 1;
                          counter <= counter + 1;
                   end if;
                   else
                      sel <= 0;
                      counter <= (others => '0');
                   end if;
             end if;
             end if;
       end process count;

       -- Controls the action, means if there is data to send, done is 1 otherwise
done is 0
       control: process(clock, reset)
       begin
          if reset = '1' then
             go <= '0';
          elsif rising_edge(clock) then
             if enable = '1' then
              if meanDataRdy = '1' AND oldMeanDataRdy = '0' then
                   go <= '1';
              elsif (sel >= ((nbOfBoards*nbOfChanPerBoard)-1)) AND (delaying =
'0')then
                   go <= '0';
             end if;
         end if;
```

```
      end if;
       end process control;

       -- Simple shift register for the rising edge detection
       dataRdyShifter: process(clock, reset)
       begin
             if reset = '1' then
                   oldMeanDataRdy <= '0';
             elsif rising_edge(clock) then
                   oldMeanDataRdy <= meanDataRdy;
             end if;
       end process dataRdyShifter;

       -- Counts a delay to not overflow the FIFO
   delayCounter: process(clock, reset)
   begin
      if reset = '1' then
         delayCnt <= (others => '0');
         delaying <= '1';
      elsif rising_edge(clock) then
         if go = '1' then
            if delayCnt < 10 then
               delayCnt <= delayCnt + 1;
               delaying <= '1';
            else
               delayCnt <= (others => '0');
               delaying <= '0';
            end if;
         end if;
      end if;
   end process delayCounter;

END ARCHITECTURE RTL;
```

Block serializer

```
--------------------------------------------------------------------
-- PROJECT: TempArray
--------------------------------------------------------------------
-- AUTHOR: Furrer Dominic
-- DATE: 21.04.2010
--------------------------------------------------------------------
-- FUNCTION: This Block has to serialize the given data on the
--                     data-out-line. The data is for the ADCs.
--------------------------------------------------------------------
ARCHITECTURE RTL OF serializer IS

       signal data : unsigned(dataBitNb-1 downto 0);
       signal oldSClock : std_ulogic;
       signal go : std_ulogic;
       signal oldWrite : std_ulogic;

BEGIN

       -- Serialize the input data; shift a bit out on every sClk
   serialize: process(clock, reset)
   begin
      if reset = '1' then
         sDataOut <= '0';
      elsif rising_edge(clock) then
       if go = '1' then
            if write = '0' and oldWrite = '1' then
             data <= unsigned(pDataIn);
         end if;

                   if sClock = '0' AND oldSClock = '1' then
                         sDataOut <= data(data'high);
```

```
                        data(data'high downto 1) <= data(data'high-1 downto 0);
                    end if;
                else
                    sDataOut <= '0';
                end if;
            end if;
        end process serialize;

        -- Set conditions to run or not
        setGo: process(clock, reset)
        begin
            if reset = '1' then
                go <= '0';
            elsif rising_edge(clock) then
                if write = '0' and oldWrite = '1' then
                    go <= '1';
                end if;
                if data = 0 then
                    go <= '0';
                end if;
            end if;
        end process setGo;

        -- Shift the sClk signal to detect a rising edge
        sClockShifter: process(clock, reset)
        begin
            if reset = '1' then
                oldSClock <= '0';
            elsif rising_edge(clock) then
                oldSClock <= sClock;
            end if;
        end process sClockShifter;

        -- Shift the write signal to detect a rising edge
        writeShifter: process(clock, reset)
        begin
            if reset = '1' then
                oldWrite <= '0';
            elsif rising_edge(clock) then
                oldWrite <= write;
            end if;
        end process writeShifter;

END ARCHITECTURE RTL;
```

14.5.6 Testbank der Simulation des Blockes ADSHandler

```
--------------------------------------------------------------------
-- PROJECT: TempArray
--------------------------------------------------------------------
-- AUTHOR: Furrer Dominic
-- DATE: 21.04.2010
--------------------------------------------------------------------
-- FUNCTION: Testbench used to check the funcitonality of the
--                      ADSHandler.
--------------------------------------------------------------------
ARCHITECTURE test OF ADSHandler_tester IS

       type TestIntArrayType is array (3 downto 0) of integer;
       type TestUVArrayType is array (3 downto 0) of unsigned(23 downto 0);

  constant clockFrequency: real := 66.0E6;
  constant clockPeriod: time := (1.0/clockFrequency) * 1 sec;
  signal cClock: std_uLogic := '1';

  constant sampleClockPeriodNb2: positive := 528;
  signal sample : TestIntArrayType;
  signal sendSample: std_ulogic;
  signal sampleBinary : TestUVArrayType;

BEGIN
   ------------------------------------------------------------------------------
                                                              -- clock and reset
   cClock <= not cClock after clockPeriod/2;
   clock <= transport cClock after clockPeriod*9/10;
   reset <= '1', '0' after 2*clockPeriod;

   ------------------------------------------------------------------------------
                                                                -- test sequence
   -- same principle as in testbench of block tempArray
   process
   begin

      enable <= '1';
      sendSample <= '0';

      wait for 30  us;

      for i in 0 to 130 loop

         sample(0) <= 16#134578#;
         sample(1) <= 16#752CD4#;
         sample(2) <= 16#285432#;
         sample(3) <= 16#1553AC#;
         sendSample <= '1', '0' after 1 ns;

         wait for 2 ms;

         sample(0) <= 16#1598BD#;
         sample(1) <= 16#4AE456#;
         sample(2) <= 16#1FE223#;
         sample(3) <= 16#3214FA#;
         sendSample <= '1', '0' after 1 ns;

         wait for 2 ms;

         sample(0) <= 16#159DB1#;
         sample(1) <= 16#0ADEF0#;
         sample(2) <= 16#167935#;
         sample(3) <= 16#123232#;
         sendSample <= '1', '0' after 1 ns;

         wait for 2 ms;
```

```
         sample(0) <= 16#100F23#;
         sample(1) <= 16#169835#;
         sample(2) <= 16#772CD4#;
         sample(3) <= 16#15985B#;
         sendSample <= '1', '0' after 1 ns;

         wait for 2 ms;

      end loop;

      wait;
   end process;

   ------------------------------------------------------------------------------
                                                                    -- bus serializer
   process
   begin

      sClock <= '0';
      sDataRdy <= '0';

      wait until rising_edge(sendSample);
      sampleBinary(0) <= to_unsigned(sample(0), sampleBinary(0)'length);
      sampleBinary(1) <= to_unsigned(sample(1), sampleBinary(1)'length);
      sampleBinary(2) <= to_unsigned(sample(2), sampleBinary(2)'length);
      sampleBinary(3) <= to_unsigned(sample(3), sampleBinary(3)'length);

      for index in sampleBinary(0)'range loop
         if index = sampleBinary(0)'high then
            sDataIn(0) <= '0';
            sDataIn(1) <= '0';
            sDataIn(2) <= '0';
            sDataIn(3) <= '0';
            sDataRdy <= '1';
         else
            sDataIn(0) <= sampleBinary(0)(index);
            sDataIn(1) <= sampleBinary(1)(index);
            sDataIn(2) <= sampleBinary(2)(index);
            sDataIn(3) <= sampleBinary(3)(index);
            sDataRdy <= '0';
         end if;

         wait for sampleClockPeriodNb2*clockPeriod;
            sClock <= '1';
         wait for sampleClockPeriodNb2*clockPeriod;
            sClock <= '0';
      end loop;

   end process;

END ARCHITECTURE test;
```

14.5.7 VHDL-Dateien des Blockes USBHandler

Block TXRXHandler

```
----------------------------------------------------------------------
-- PROJECT: TempArray
----------------------------------------------------------------------
-- AUTHOR: Furrer Dominic
-- DATE: 21.04.2010
----------------------------------------------------------------------
-- FUNCTION: This block acts as the dolmedger between the usb and
--                      the rest of the circuit.
----------------------------------------------------------------------
ARCHITECTURE RTL OF TXRXHandler IS

      constant delay : integer := 120;

      signal intSendData : std_logic_vector(USBIntDataBitNb-1 DOWNTO 0);
      signal cnt : integer;
      signal go : std_ulogic;
      signal oldWriteInt : std_ulogic;
   signal delayCnt : unsigned(6 downto 0);
   signal delaying : std_ulogic;


BEGIN

   intSendData <= dataInInt;

      -- Organize the saving into the fifo
      threatSendData: process(reset, clock)
      begin
            if reset = '1' then
                  cnt <= 0;
            elsif rising_edge(clock) then
               if enable = '1' then
                   if go = '1' then
                         if delaying = '0' then
                          dataOutExt <= intSendData(((((USBIntDataBitNb /
USBExtDataBitNb)-cnt)*USBExtDataBitNb)-1) downto ((((USBIntDataBitNb /
USBExtDataBitNb)-1)-cnt)*USBExtDataBitNb));
                             cnt <= cnt + 1;
                             writeExt <= '1';
                          else
                             writeExt <= '0';
              end if;
                   else
                      writeExt <= '0';
                          cnt <= 0;
                   end if;
         end if;
             end if;
      end process threatSendData;

      -- Controll that there is more or no more data to send to the fifo
      checkSendState: process(clock, reset)
      begin
         if reset = '1' then
            go <= '0';
         elsif rising_edge(clock) then
         if enable = '1' then
             if writeInt = '1' AND oldWriteInt = '0' then
                   go <= '1';
         end if;
             if cnt >= (USBIntDataBitNb / USBExtDataBitNb) then
                   go <= '0';
             end if;
         end if;
```

```
      end if;
       end process checkSendState;

   -- count to wait a certain time for the fifo
   delayCounter: process(clock, reset)
   begin
      if reset = '1' then
         delayCnt <= (others => '0');
         delaying <= '1';
      elsif rising_edge(clock) then
         if go = '1' then
            if delayCnt < 120 then
               delayCnt <= delayCnt + 1;
               delaying <= '1';
            else
               delayCnt <= (others => '0');
               delaying <= '0';
            end if;
         end if;
      end if;
   end process delayCounter;

       -- Shift the write-input to see rising edge
       writeShift: process(clock, reset)
       begin
          if reset = '1' then
             oldWriteInt <= '0';
          elsif rising_edge(clock) then
             if enable = '1' then
                oldWriteInt <= writeInt;
             end if;
      end if;
   end process writeShift;

END ARCHITECTURE RTL;
```

14.5.8 Testbank der Simulation des Blockes USBHandler

```
--------------------------------------------------------------------
-- PROJECT: TempArray
--------------------------------------------------------------------
-- AUTHOR: Furrer Dominic
-- DATE: 21.04.2010
--------------------------------------------------------------------
-- FUNCTION: Testbench that simulates the test conditions for the
--                      USBHandler.
--------------------------------------------------------------------
LIBRARY TempArray;
   USE TempArray.TempArrayPkg.all;

ARCHITECTURE test OF USBHandler_tester IS

  constant clockFrequency: real := 66.0E6;
  constant clockPeriod: time := (1.0/clockFrequency) * 1 sec;
  signal sClock: std_uLogic := '1';

BEGIN
   ------------------------------------------------------------------------------
                                                              -- clock and reset
   sClock <= not sClock after clockPeriod/2;
   clock <= transport sClock after clockPeriod*9/10;
   reset <= '1', '0' after 2*clockPeriod;

   ------------------------------------------------------------------------------
                                                                -- test sequence

   process
   begin

            -- set the conditions to test the device
      write <= '0';
      read <= '0';
      enable <= '0';
      txeUSB <= '0';
      rxfUSB <= '0';
      usbFifoData <= (others => 'Z');
      dataIn <= (others => '0');

      wait for 100 us;

      enable <= '1';
      txeUSB <= '1';
      rxfUSB <= '0';
      write <= '0', '1' after 100 us;

                  -- send data to see the functioning
      wait for 10 us;
      dataIn <= "00011110000011111100000100101111";
      write <= '1';
      wait for 20 ns;
      write <= '0';

      wait for 10 us;
      dataIn <= "00001111101011100000111111010101";
      write <= '1';
      wait for 20 ns;
      write <= '0';

      wait for 10 us;
      dataIn <= "11111111111111000000000000000000";
      write <= '1';
      wait for 20 ns;
      write <= '0';
```

```
      wait for 10 us;
      dataIn <= "10101110001100011010101100101010";
      write <= '1';
      wait for 20 ns;
      write <= '0';

      wait;
   end process;

END ARCHITECTURE test;
```

14.5.9 VHDL-Dateien des Blockes controller

```
--
-- VHDL Architecture TempArray.controller.fsm
--
-- Created:
--          by - furrerd1.UNKNOWN (WE3230)
--          at - 15:02:30 23.06.2010
--
-- Generated by Mentor Graphics' HDL Designer(TM) 2007.1a (Build 13)
--
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.ALL;

ARCHITECTURE fsm OF controller IS

   TYPE STATE_TYPE IS (
      init,
      idle,
      dataToTransfer,
      writeDataToUSB,
      waitForUSB1,
      resetDevice,
      waitForReset,
      unResetDevice,
      nextFIFOValue,
      waitForUSB2,
      waitForConv
   );

   -- Declare current and next state signals
   SIGNAL current_state : STATE_TYPE;
   SIGNAL next_state : STATE_TYPE;

   -- Declare Wait State internal signals
   SIGNAL csm_timer : std_logic_vector(13 DOWNTO 0);
   SIGNAL csm_next_timer : std_logic_vector(13 DOWNTO 0);
   SIGNAL csm_timeout : std_logic;
   SIGNAL csm_to_waitForUSB1 : std_logic;
   SIGNAL csm_to_waitForReset : std_logic;
   SIGNAL csm_to_waitForUSB2 : std_logic;
   SIGNAL csm_to_waitForConv : std_logic;

BEGIN

   -----------------------------------------------------------------
   clocked_proc : PROCESS (
      clock,
      reset
   )
   -----------------------------------------------------------------
   BEGIN
      IF (reset = '1') THEN
         current_state <= init;
         csm_timer <= (OTHERS => '0');
      ELSIF (clock'EVENT AND clock = '1') THEN
         current_state <= next_state;
         csm_timer <= csm_next_timer;
      END IF;
   END PROCESS clocked_proc;

   -----------------------------------------------------------------
   nextstate_proc : PROCESS (
      csm_timeout,
      current_state,
      dataRdyADS,
      dataRdySPI,
```

```
      emptyFIFO
   )
   ----------------------------------------------------------------------
   BEGIN
      -- Default assignments to Wait State entry flags
      csm_to_waitForUSB1 <= '0';
      csm_to_waitForReset <= '0';
      csm_to_waitForUSB2 <= '0';
      csm_to_waitForConv <= '0';
      CASE current_state IS
         WHEN init =>
            next_state <= resetDevice;
         WHEN idle =>
            IF (dataRdyADS = '1') THEN
               next_state <= waitForConv;
               csm_to_waitForConv <= '1';
            ELSE
               next_state <= idle;
            END IF;
         WHEN dataToTransfer =>
            IF (emptyFIFO = '0') THEN
               next_state <= writeDataToUSB;
            ELSIF (emptyFIFO = '1') THEN
               next_state <= idle;
            ELSE
               next_state <= dataToTransfer;
            END IF;
         WHEN writeDataToUSB =>
            next_state <= waitForUSB1;
            csm_to_waitForUSB1 <= '1';
         WHEN waitForUSB1 =>
            IF (csm_timeout = '1') THEN
               next_state <= nextFIFOValue;
            ELSE
               next_state <= waitForUSB1;
            END IF;
         WHEN resetDevice =>
            next_state <= waitForReset;
            csm_to_waitForReset <= '1';
         WHEN waitForReset =>
            IF (csm_timeout = '1') THEN
               next_state <= unResetDevice;
            ELSE
               next_state <= waitForReset;
            END IF;
         WHEN unResetDevice =>
            next_state <= idle;
         WHEN nextFIFOValue =>
            next_state <= waitForUSB2;
            csm_to_waitForUSB2 <= '1';
         WHEN waitForUSB2 =>
            IF (csm_timeout = '1') THEN
               next_state <= dataToTransfer;
            ELSE
               next_state <= waitForUSB2;
            END IF;
         WHEN waitForConv =>
            IF (csm_timeout = '1') THEN
               next_state <= idle;
            ELSIF (dataRdySPI = '1') THEN
               next_state <= dataToTransfer;
            ELSE
               next_state <= waitForConv;
            END IF;
         WHEN OTHERS =>
            next_state <= init;
      END CASE;
   END PROCESS nextstate_proc;
```

```
   ---------------------------------------------------------------------
   output_proc : PROCESS (
      current_state
   )
   ---------------------------------------------------------------------
   BEGIN
      -- Default Assignment
      dataOutSPI <= (others => '0');
      dataOutUSB <= (others => 'Z');
      enableFIFO <= '0';
      enableSClk <= '0';
      enableSPI <= '1';
      enableUSB <= '1';
      readFIFO <= '0';
      readUSB <= '0';
      resetADS <= '0';
      shutdownADS <= '0';
      shutdownLT <= '0';
      syncADS <= '0';
      writeSPI <= '0';
      writeUSB <= '0';

      -- Combined Actions
      CASE current_state IS
         WHEN idle =>
            enableSClk <= '0';
         WHEN dataToTransfer =>
            enableFIFO <= '1';
         WHEN writeDataToUSB =>
            enableFIFO <= '1';
            writeUSB <= '1' ;
         WHEN waitForUSB1 =>
            enableFIFO <= '1';
            writeUSB <= '0';
         WHEN resetDevice =>
            resetADS <= '1' ;
         WHEN unResetDevice =>
            resetADS <= '0' ;
         WHEN nextFIFOValue =>
            readFIFO <= '1';
            enableFIFO <= '1';
         WHEN waitForUSB2 =>
            enableFIFO <= '1';
            readFIFO <= '0';
         WHEN waitForConv =>
            enableSClk <= '1';
         WHEN OTHERS =>
            NULL;
      END CASE;
   END PROCESS output_proc;

   ---------------------------------------------------------------------
   csm_wait_combo_proc: PROCESS (
      csm_timer,
      csm_to_waitForUSB1,
      csm_to_waitForReset,
      csm_to_waitForUSB2,
      csm_to_waitForConv
   )
   ---------------------------------------------------------------------
   VARIABLE csm_temp_timeout : std_logic;
   BEGIN
      IF (unsigned(csm_timer) = 0) THEN
         csm_temp_timeout := '1';
      ELSE
         csm_temp_timeout := '0';
      END IF;
```

```
      IF (csm_to_waitForUSB1 = '1') THEN
         csm_next_timer <= "00001010010011"; -- no cycles(660)-1=659
      ELSIF (csm_to_waitForReset = '1') THEN
         csm_next_timer <= "00010100010011"; -- no cycles(1300)-1=1299
      ELSIF (csm_to_waitForUSB2 = '1') THEN
         csm_next_timer <= "01001110000111"; -- no cycles(5000)-1=4999
      ELSIF (csm_to_waitForConv = '1') THEN
         csm_next_timer <= "11001110001111"; -- no cycles(13200)-1=13199
      ELSE
         IF (csm_temp_timeout = '1') THEN
            csm_next_timer <= (OTHERS=>'0');
         ELSE
            csm_next_timer <= std_logic_vector(unsigned(csm_timer) - 1);
         END IF;
      END IF;
      csm_timeout <= csm_temp_timeout;
   END PROCESS csm_wait_combo_proc;

END fsm;
```

14.5.10 Testbank der Simulation des Blockes controller

```
--------------------------------------------------------------------
-- PROJECT: TempArray
--------------------------------------------------------------------
-- AUTHOR: Furrer Dominic
-- DATE: 21.04.2010
--------------------------------------------------------------------
-- FUNCTION: Testbench to test the FSM of the controller.
--------------------------------------------------------------------
library ieee;

ARCHITECTURE test OF controller_tester IS

  constant clockFrequency: real := 66.0E6;
  constant clockPeriod: time := (1.0/clockFrequency) * 1 sec;
  signal sClock: std_uLogic := '1';

BEGIN
  ------------------------------------------------------------------------------
                                                          -- clock and reset
  sClock <= not sClock after clockPeriod/2;
  clock <= transport sClock after clockPeriod*9/10;
  reset <= '1', '0' after 2*clockPeriod;

  ------------------------------------------------------------------------------
                                                            -- test sequence
  process
  begin

       -- set conditions for testing
       emptyFiFO <= '0';
       fullFIFO <= '0';
       dataRdySPI <= '0';
       dataInUSB <= (others => '0');
       dataRdyUSB <= '0';
       dataRdyADS <= '0';

      -- check empty FIFO
       wait for 100 us;
       dataRdyADS <= '1';
        emptyFIFO <= '1';
       wait for 1 us;
       dataRdyADS <= '0';
       wait for 100 us;
       dataRdySPI <= '1';
       wait for 1 us;
       dataRdySPI <= '0';

   -- check normal funciton
   wait for 2 ms;
   emptyFIFO <= '0';
       wait for 100 us;
       dataRdyADS <= '1';
       wait for 1 us;
       dataRdyADS <= '0';
       wait for 100 us;
       dataRdySPI <= '1';
       wait for 1 us;
       dataRdySPI <= '0';
   wait for 200 us;
   emptyFifo <= '1';

   wait;

  end process;

END ARCHITECTURE test;
```

14.5.11 Bibliothek TempArrayPkg

Definition (Header)

```
--------------------------------------------------------------------
-- PROJECT: TempArray
--------------------------------------------------------------------
-- AUTHOR: Furrer Dominic
-- DATE: 21.04.2010
--------------------------------------------------------------------
-- FUNCTION: Library containig definitions that are used for the
--                      TempArray-project
--------------------------------------------------------------------
LIBRARY ieee;
   USE ieee.std_logic_1164.all;
   USE ieee.numeric_std.all;

PACKAGE TempArrayPkg IS

   -- Type for the serial data signals
   constant spiParLength : positive := 24;
   subtype spiParType is std_ulogic_vector(spiParLength-1 downto 0);
   constant spiParArrayLength : positive := 16;
   type spiParArrayType is array (spiParArrayLength-1 downto 0) of spiParType;

   -- Type for the data ready signals
   subtype spiSerType is std_ulogic;
   constant spiSerArrayLength : positive := 16;
   type spiSerArrayType is array (spiSerArrayLength-1 downto 0) of spiSerType;

   -- Function to know how many bits are necessary for a certain number
   function requiredBitNb (val : integer) return integer;

END TempArrayPkg;
```

Implementierung (Body)

```
PACKAGE BODY TempArrayPkg IS

       -- Taken from Hes-so Valais / Wallis
       -- Counts the required number of bits for a given value
   function requiredBitNb (val : integer) return integer is
      variable powerOfTwo, bitNb : integer;
      begin
      powerOfTwo := 1;
      bitNb := 0;
      while powerOfTwo < val loop
      powerOfTwo := 2 * powerOfTwo;
      bitNb := bitNb + 1;
      end loop;
      return bitNb;
   end requiredBitNb;

END TempArrayPkg;
```

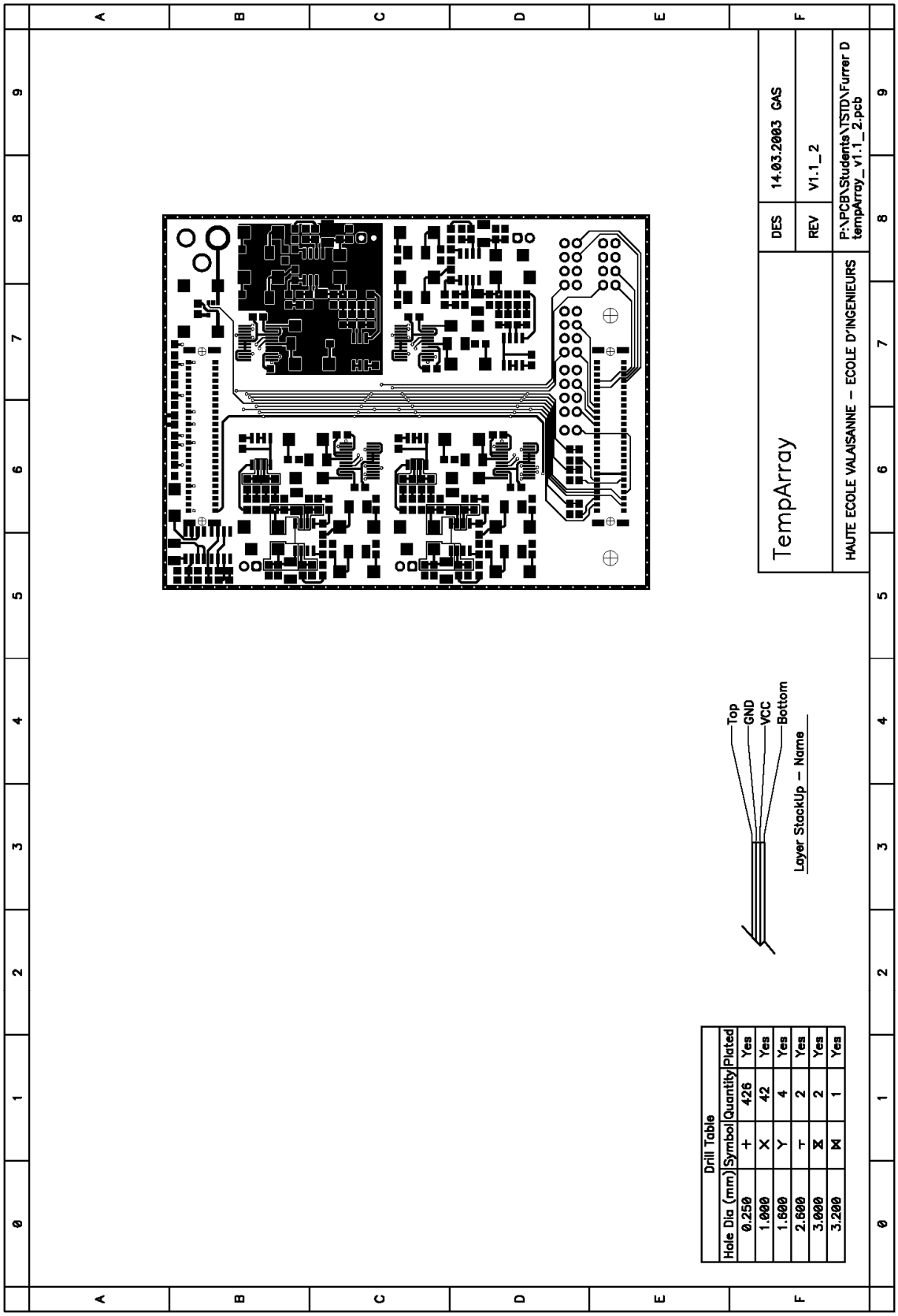
14.5.12 UCF-Datei

```
#------------------------------------------------------------------------------
# Clock and reset
#
NET "clock"  LOC = "P185" | IOSTANDARD = LVTTL;
NET "Nreset"  LOC = "P189" | IOSTANDARD = LVTTL;

#------------------------------------------------------------------------------
# ADS connections
# Inputs
NET "NDataRdyADS"  LOC = "P202" | IOSTANDARD = LVTTL;

NET "dataInADS00"  LOC = "P21" | IOSTANDARD = LVTTL;
NET "dataInADS01"  LOC = "P27" | IOSTANDARD = LVTTL;
NET "dataInADS02"  LOC = "P33" | IOSTANDARD = LVTTL;
NET "dataInADS03"  LOC = "P37" | IOSTANDARD = LVTTL;

NET "dataInADS10"  LOC = "P22" | IOSTANDARD = LVTTL;
NET "dataInADS11"  LOC = "P29" | IOSTANDARD = LVTTL;
NET "dataInADS12"  LOC = "P34" | IOSTANDARD = LVTTL;
NET "dataInADS13"  LOC = "P41" | IOSTANDARD = LVTTL;

NET "dataInADS20"  LOC = "P23" | IOSTANDARD = LVTTL;
NET "dataInADS21"  LOC = "P30" | IOSTANDARD = LVTTL;
NET "dataInADS22"  LOC = "P35" | IOSTANDARD = LVTTL;
NET "dataInADS23"  LOC = "P42" | IOSTANDARD = LVTTL;

NET "dataInADS30"  LOC = "P24" | IOSTANDARD = LVTTL;
NET "dataInADS31"  LOC = "P31" | IOSTANDARD = LVTTL;
NET "dataInADS32"  LOC = "P36" | IOSTANDARD = LVTTL;
NET "dataInADS33"  LOC = "P43" | IOSTANDARD = LVTTL;

#Outputs
NET "syncADS"  LOC = "P200" | IOSTANDARD = LVTTL;
NET "NresetADS"  LOC = "P195" | IOSTANDARD = LVTTL;
NET "NshutdownADS"  LOC = "P199" | IOSTANDARD = LVTTL;
NET "NshutdownLT"  LOC = "P203" | IOSTANDARD = LVTTL;
NET "clockADS"  LOC = "P201" | IOSTANDARD = LVTTL;
NET "sClockADS"  LOC = "P3" | IOSTANDARD = LVTTL;
NET "dataOutADS"  LOC = "P206" | IOSTANDARD = LVTTL;


#------------------------------------------------------------------------------
# USB connections

NET "NreadUSB" LOC = "P179"  | IOSTANDARD = LVTTL;
NET "writeUSB" LOC = "P180"  | IOSTANDARD = LVTTL;
NET "NtxeUSB" LOC = "P181"  | IOSTANDARD = LVTTL;
NET "NrxfUSB" LOC = "P187"  | IOSTANDARD = LVTTL;

# In- and Output
NET "fifoDataUSB<0>"  LOC = "P167" | IOSTANDARD = LVTTL;
NET "fifoDataUSB<1>"  LOC = "P168" | IOSTANDARD = LVTTL;
NET "fifoDataUSB<2>"  LOC = "P172" | IOSTANDARD = LVTTL;
NET "fifoDataUSB<3>"  LOC = "P173" | IOSTANDARD = LVTTL;
NET "fifoDataUSB<4>"  LOC = "P174" | IOSTANDARD = LVTTL;
NET "fifoDataUSB<5>"  LOC = "P175" | IOSTANDARD = LVTTL;
NET "fifoDataUSB<6>"  LOC = "P176" | IOSTANDARD = LVTTL;
NET "fifoDataUSB<7>"  LOC = "P178" | IOSTANDARD = LVTTL;
```

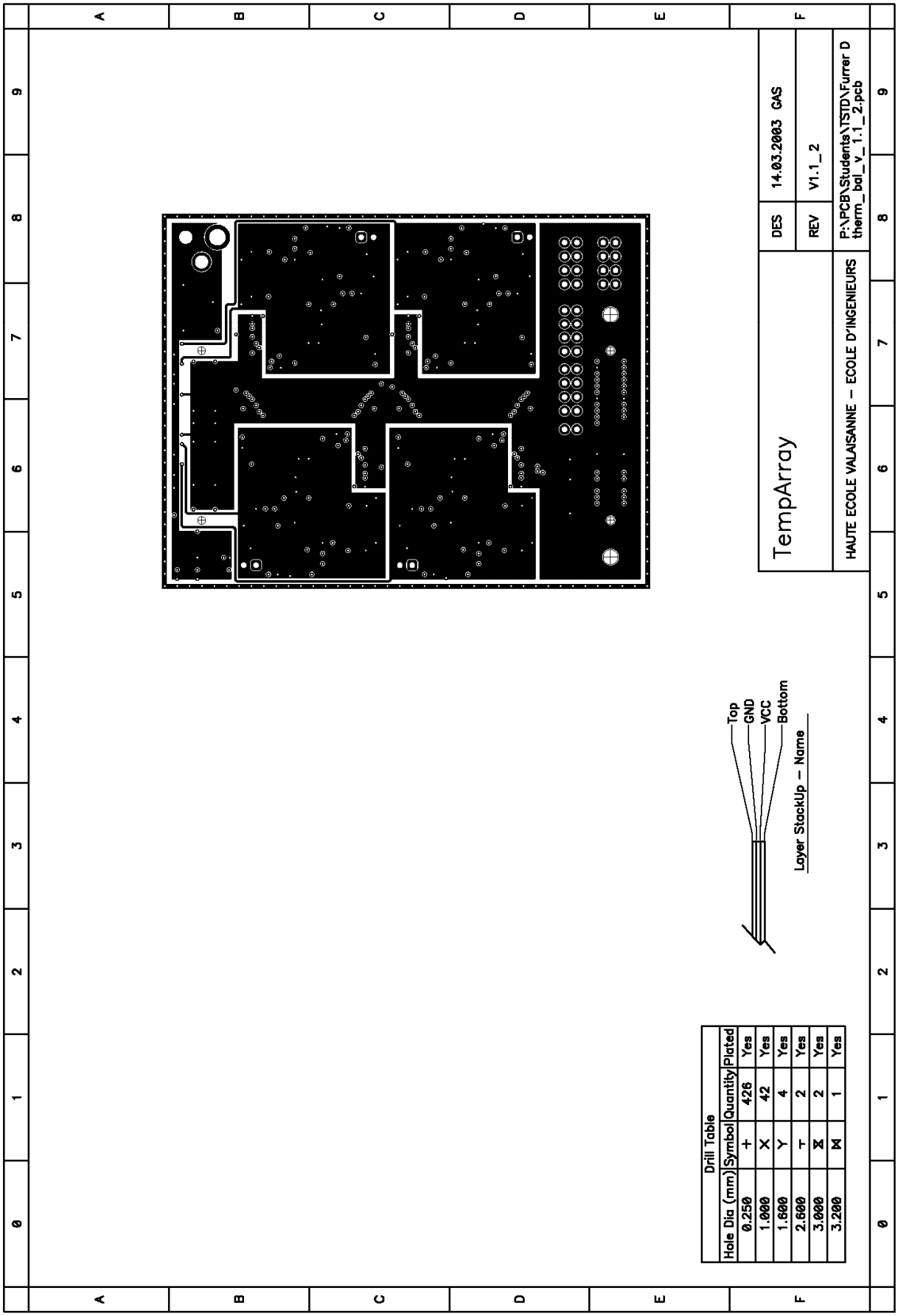
14.6 Herstellung

14.6.1 Layout

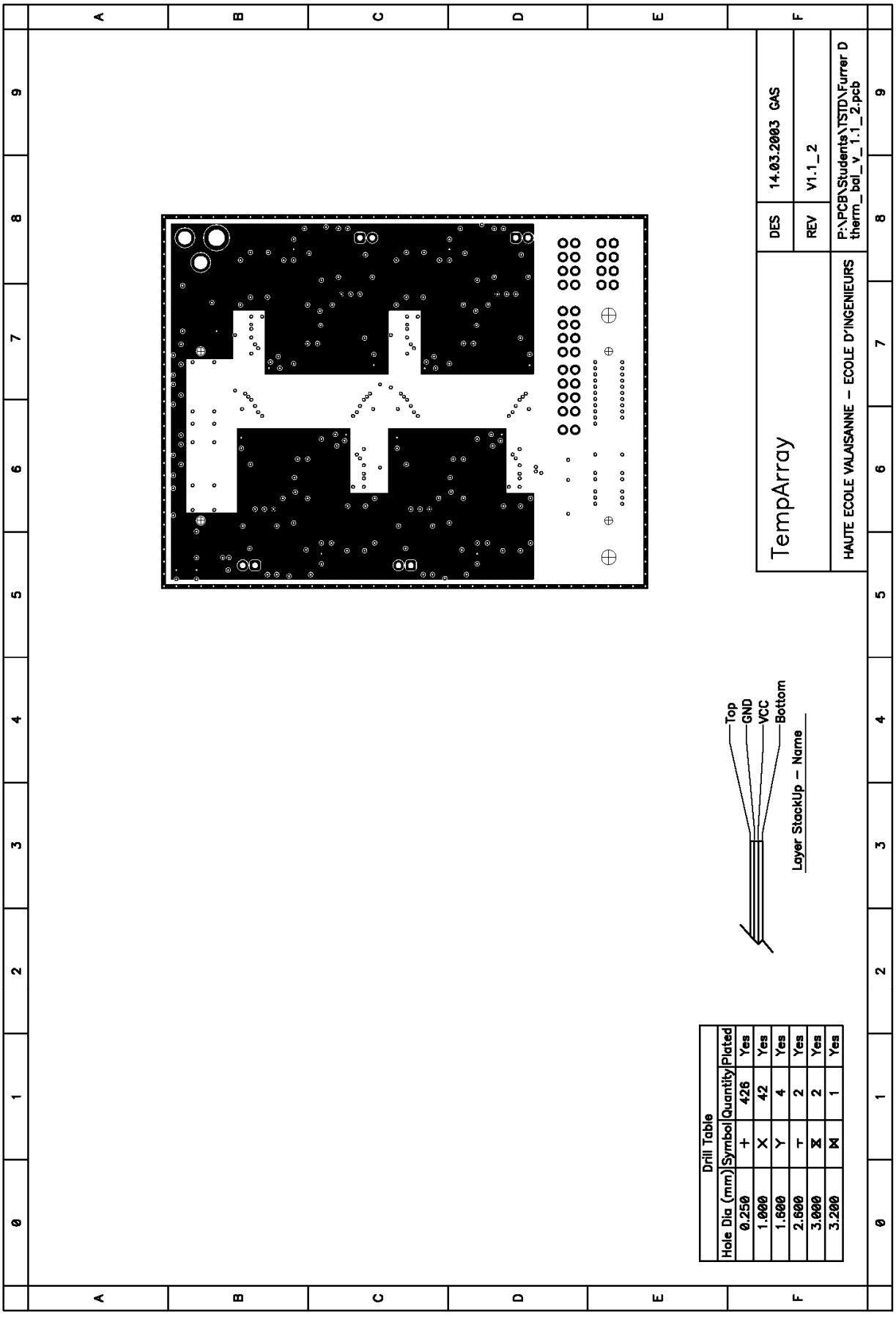
Top-Layer



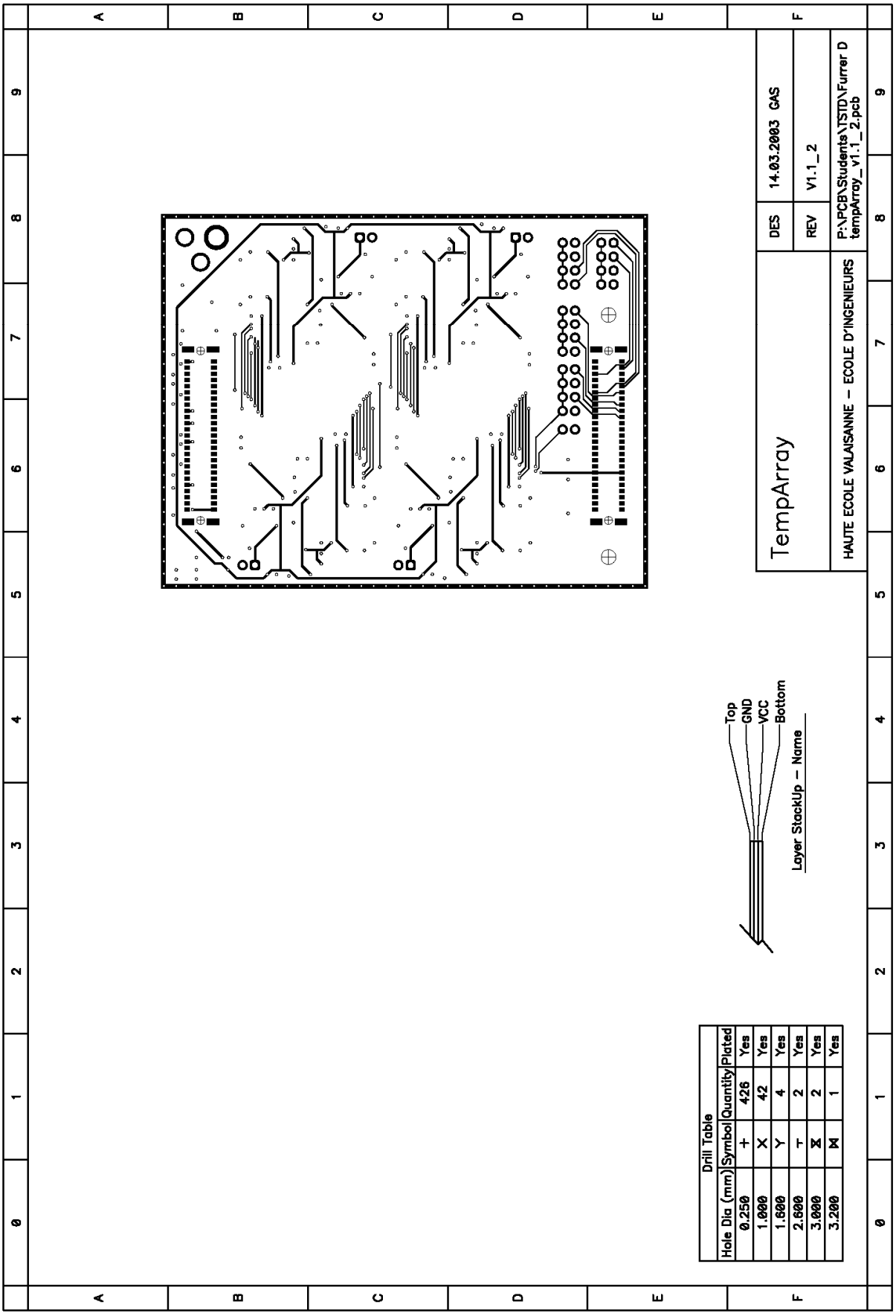
GND-Layer


TempArray
DES 14.03.2003 GAS
REV V1.1_2
P:\PCB\Students\TSTD\Furrer D therm_bal_v_1.1_2.pcb
HAUTE ECOLE VALAISANNE – ECOLE D'INGENIEURS
Top
GND
VCC
Bottom
Layer StackUp – Name
Drill Table
Hole Dia (mm) Symbol Quantity Plated
0.250 + 426 Yes
1.000 X 42 Yes
1.600 Y 4 Yes
2.600 T 2 Yes
3.000 Z 2 Yes
3.200 M 1 Yes

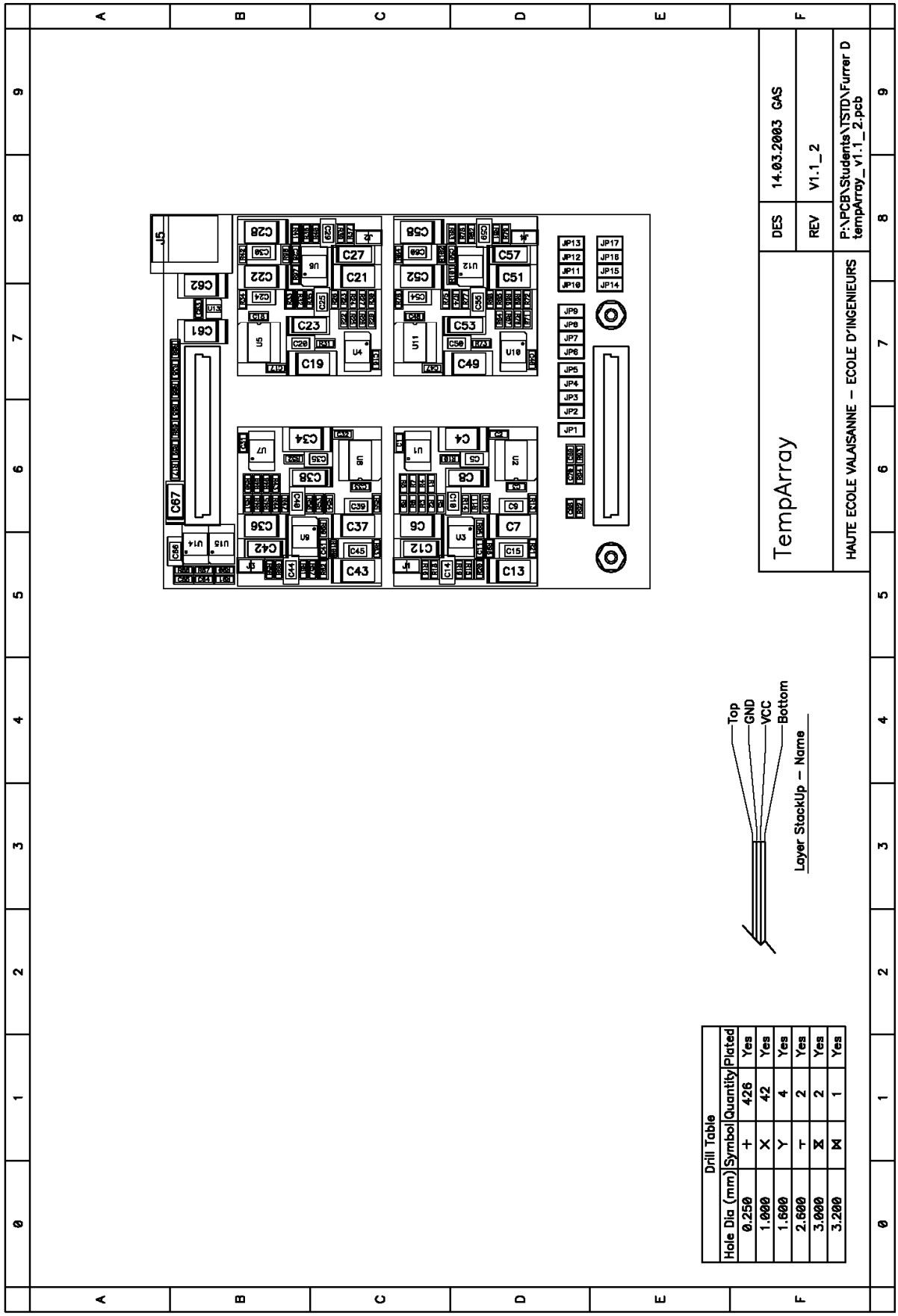
Vcc-Layer


TempArray
HAUTE ECOLE VALAISANNE – ECOLE D'INGENIEURS
DES 14.03.2003 GAS
REV V1.1_2
P:\PCB\Students\TSTD\Furrer D therm_bal_v_1.1_2.pcb
Top
GND
VCC
Bottom
Layer StackUp – Name
Drill Table
Hole Dia (mm) Symbol Quantity Plated
0.250 + 426 Yes
1.000 X 42 Yes
1.600 Y 4 Yes
2.600 T 2 Yes
3.000 W 2 Yes
3.200 M 1 Yes

Bottom-Layer



Bestückungsplan



14.6.2 Komponentenliste

P-CAD Bill of Materials
P:\PCB\Students\TSDS\Furrer D\PCB\tempArray_v1.2.sch

Count	ComponentName	RefDes	Value	Partition
8	AD8552AR	U1		
		U3		
		U4		
		U6		
		U7		
		U9		
		U10		
		U12		
1	AD8628ARZ	U15	AD8628ARZ	
4	ADS1281PW	U2		
		U5		
		U8		
		U11		
4	BAR1X2	J1		
		J2		
		J3		
		J4		
1	CAP30805	C63	10n	
18	CAP30805	C1	100n	
		C2		
		C3		
		C11		
		C16		
		C17		
		C18		
		C26		
		C31		
		C32		
		C33		
		C41		
		C46		
		C47		
		C48		
		C56		
		C64		
		C65		
3	CAP30805	C68	100p	VBE DCS Sensor
		C69		
		C70		
21	CAPS1210	C5	100n	
		C9		
		C10		
		C14		
		C15		
		C20		
		C24		
		C25		
		C29		
		C30		
		C35		
		C39		
		C40		
		C44		
		C45		
		C50		
		C54		
		C55		
		C59		
		C60		
		C66		
5	CAPS6032P:TAJ-C	C8	1u	
		C23		
		C38		
		C53		
		C67		
4	CAPS6032P:TAJ-C	C12	2.2u	
		C27		
		C42		
		C57		
13	CAPS7343P:TAJ-D	C4	10u	
		C7		
		C13		
		C19		
		C22		
		C28		
		C34		
		C37		
		C43		
		C49		
		C52		
		C58		
		C61		
1	CAPS7343P:TAJ-D	C62	47u	
4	CAPS7343P:TAJ-D	C6	220u	
		C21		
		C36		
		C51		
1	DC10B	J5	{Value}	
2	ERNI-SMC-50P-FD	J6		Mezzanine {RefDes2}
		J8		
2	ERNI-SMC-50P-MD	J7		Mezzanine {RefDes2}
		J9		
17	JUMPER	JP1	{Value}	
		JP2		
		JP3		
		JP4		
		JP5		
		JP6		
		JP7		
		JP8		

		JP9		
		JP10		
		JP11		
		JP12		
		JP13		
		JP14		
		JP15		
		JP16		
		JP17		
1	LP2985M5-X.X	U13	LP2985	
1	MAX6126	U14	2.5 V	
16	RS0805	R2	0	
		R7		
		R9		
		R12		
		R14		
		R17		
		R20		
		R23		
		R28		
		R30		
		R33		
		R41		
		R44		
		R49		
		R51		
		R54		
1	RS0805	R91	00	VBB DCS Sensor
23	RS0805	R35	0	
		R38		
		R56		
		R59		
		R62		
		R65		
		R70		
		R72		
		R75		
		R77		
		R80		
		R83		
		R85		
		R86		
		R89		
		R95		
		R96		
		R97		
		R98		
		R99		
		R100		
		R101		
		R102		
28	RS0805	R1	1k 10ppm	
		R4		
		R6		
		R8		
		R11		
		R15		
		R19		
		R22		
		R25		
		R27		
		R29		
		R32		
		R36		
		R40		
		R43		
		R46		
		R48		
		R50		
		R53		
		R57		
		R61		
		R64		
		R67		
		R69		
		R71		
		R74		
		R78		
		R82		
8	RS0805	R3	2k43 10ppm	
		R5		
		R24		
		R26		
		R45		
		R47		
		R66		
		R68		
4	RS0805	R18	8k45 10ppm	
		R39		
		R60		
		R81		
4	RS0805	R16	8k66 10ppm	
		R37		
		R58		
		R79		
2	RS0805	R87	10	VBB DCS Sensor
		R90		
13	RS0805	R10	10	
		R13		
		R21		
		R31		
		R34		
		R42		
		R52		
		R55		
		R63		
		R73		
		R76		
		R84		
		R88		

3 R30805
R92
R93
R94
100
VEE DCS Sensor
30-Jun-10 11:26
Page 1

14.7 Matlab

14.7.1 Code der grafischen Benutzeroberfläche

```
function varargout = tempArray(varargin)
% TEMPARRAY M-file for tempArray.fig
%      TEMPARRAY, by itself, creates a new TEMPARRAY or raises the existing
%      singleton*.
%
%      H = TEMPARRAY returns the handle to a new TEMPARRAY or the handle to
%      the existing singleton*.
%
%      TEMPARRAY('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in TEMPARRAY.M with the given input arguments.
%
%      TEMPARRAY('Property','Value',...) creates a new TEMPARRAY or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before tempArray_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to tempArray_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help tempArray

% Last Modified by GUIDE v2.5 27-Jun-2010 10:05:07

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @tempArray_OpeningFcn, ...
                   'gui_OutputFcn',  @tempArray_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before tempArray is made visible.
function tempArray_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to tempArray (see VARARGIN)
global gLogFlag;
global gFileName;
global gSerialHandle;
global gChannelSelection;
global gReadNWrite;
global gConnectionInitalised;
global gNbrOfChan;

% Initialize global variables
gLogFlag = 0;
gFileName = [];
```

```
gSerialHandle = [];
gChannelSelection = [1,1,1,1,1,1,1,1];
gReadNWrite = 1;
gConnectionInitalised = 0;
gNbrOfChan = 8;

% Set axis title for the graph
xlabel('Time t [s]');
ylabel('Temperature T [°C]');

% Choose default command line output for tempArray
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes tempArray wait for user response (see UIRESUME)
% uiwait(handles.root);


% --- Outputs from this function are returned to the command line.
function varargout = tempArray_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in radio_channel1.
function radio_channel1_Callback(hObject, eventdata, handles)
% hObject    handle to radio_channel1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radio_channel1
global gChannelSelection;
gChannelSelection(1) = get(hObject,'Value');


% --- Executes on button press in radio_channel2.
function radio_channel2_Callback(hObject, eventdata, handles)
% hObject    handle to radio_channel2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radio_channel2
global gChannelSelection;
gChannelSelection(2) = get(hObject,'Value');


% --- Executes on button press in radio_channel3.
function radio_channel3_Callback(hObject, eventdata, handles)
% hObject    handle to radio_channel3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radio_channel3
global gChannelSelection;
gChannelSelection(3) = get(hObject,'Value');


% --- Executes on button press in radio_channel4.
function radio_channel4_Callback(hObject, eventdata, handles)
% hObject    handle to radio_channel4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radio_channel4
global gChannelSelection;
gChannelSelection(4) = get(hObject,'Value');


% --- Executes on button press in radio_channel5.
function radio_channel5_Callback(hObject, eventdata, handles)
% hObject    handle to radio_channel5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radio_channel5
global gChannelSelection;
gChannelSelection(5) = get(hObject,'Value');


% --- Executes on button press in radio_channel6.
function radio_channel6_Callback(hObject, eventdata, handles)
% hObject    handle to radio_channel6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radio_channel6
global gChannelSelection;
gChannelSelection(6) = get(hObject,'Value');


% --- Executes on button press in radio_channel7.
function radio_channel7_Callback(hObject, eventdata, handles)
% hObject    handle to radio_channel7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radio_channel7
global gChannelSelection;
gChannelSelection(7) = get(hObject,'Value');


% --- Executes on button press in radio_channel8.
function radio_channel8_Callback(hObject, eventdata, handles)
% hObject    handle to radio_channel8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radio_channel8
global gChannelSelection;
gChannelSelection(8) = get(hObject,'Value');


% --- Executes on button press in togglebutton_log_data.
function togglebutton_log_data_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton_log_data (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global gLogFlag;
global gFileName;
global gSerialHandle;
global gReadNWrite;
global gConnectionInitalised;

% If the usb-connection isn't already initialised
if gConnectionInitalised == 0
    % Initialise it
    % Get the port
    port = get(handles.edit_com_text, 'String');
    % Create conneciton
```

```
    gSerialHandle = serial(port,'BaudRate',115200,'DataBits',8);
    % Timeout after one second
    set(gSerialHandle, 'Timeout', 1);
    set(gSerialHandle, 'InputBufferSize', 512);
    % Call the funciton to handel the data every second, means after 16
    % arrived bytes
    set(gSerialHandle, 'BytesAvailableFcnCount', 16);
    set(gSerialHandle, 'BytesAvailableFcn', 'tempArrayLogData',
'BytesAvailableFcnMode', 'byte');
    % Set a timer to read the data every second
    set(gSerialHandle, 'TimerPeriod', 1);
    set(gSerialHandle, 'TimerFcn', 'tempArrayLogData');
    set(gSerialHandle, 'ByteOrder', 'bigEndian');

    % Try to open the initialised connection
    try
        fopen(gSerialHandle);
    catch ME
        set(handles.togglebutton_log_data,'Value',0);
        msgbox (ME.message, 'Serial Port connection', 'error', 'modal');
        return;
    end;

    % Set the flag
    gConnectionInitalised = 1;
end;

% Get the value of the togglebutton
if get(hObject, 'Value') == 1

    % If the data should be logged from now open the new file window
    [filename, pathname, filterindex] = uiputfile('d:\tmp\*.bin','Log serial data
as');

    % Check the correctness of the file and open it
    if filename ~= 0
        gFileName = strcat(pathname,filename);
        file = fopen(gFileName,'w');
        fclose(file);
    else
        set(hObject, 'Value', 0);
        gLogFlag = get(hObject, 'Value');
    end;
    gReadNWrite = 0;
else % If data should no longer be logged
    % Flushing buffer
    nBytes = get(gSerialHandle,'BytesAvailable');
    if nBytes > 0
        % Write last data if there is some
        file = fopen(gFileName,'a');
        fwrite(file, fread(gSerialHandle, nBytes));
        % Then close the file
        fclose(file);
    end;
    gReadNWrite = 1;
end;

% Set flag for the tempArrayLogData funciton
gLogFlag = get(hObject, 'Value');


% --- Executes on button press in pushbutton_update_plot.
function pushbutton_update_plot_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_update_plot (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global gTime;
global gTemperatureValues;
```

```
global gChannelSelection;
global gNbrOfChan;

i = 1;
j = 1;

% Calculate the measured or opened data
tempArrayCalcData();

% Create the plot
axes(handles.axes_temp_plot)

% and plot the data
if max(gChannelSelection) > 0
    % Plot data if ther is some
    if gTemperatureValues ~= 0
        plot(gTime,gTemperatureValues);

        % Get the number of measured for the PSD-plot
        nbrOfMesures = size(gTemperatureValues);

        % Clear old legend
        clear stringTab;
        stringTab = {};

        % Create new legend of the lines
        while i < gNbrOfChan+1
            if gChannelSelection(i) == 1
                if nbrOfMesures(1) >= j
                    stringTab{j} = strcat('Channel ',num2str(i));
                    j = j + 1;
                end;
            end;
            i = i + 1;
        end;

        legend(stringTab);
    else
        % If there is no data, clear the plot
        cla;
    end;
else
    % If no channel is selected, clear the plot
    cla;
end;

% Adds a x- and y-axis description
xlabel('Time t [s]');
ylabel('Temperature T [°C]');
% Show the grid
grid on;

% Update the plot
guidata(hObject, handles);


% --- Executes on button press in pushbutton_show_psd.
function pushbutton_show_psd_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_show_psd (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global gTime;
global gTemperatureValues;
global gChannelSelection;
global gNbrOfChan;

% Calculate measured or opened data
tempArrayCalcData();
```

```
% If ther is some data, continue
if gTemperatureValues ~= 0
    % Get the number of measured for the PSD-plot
    nbrOfMesures = size(gTemperatureValues);

    i = 1;
    j = 1;
    %Plot the PSD of each selected channel
    while i < gNbrOfChan+1
        if gChannelSelection(i) == 1
            if nbrOfMesures(1) >= j
                ylab = strcat('Power Spectral Density S(f) of Ch',num2str(i));
                ylab = strcat(ylab,' [°C/\surdHz]');

PlotPSD(gTime,gTemperatureValues(j,1:nbrOfMesures(2)),'YLabel',ylab,'XLabel','Frequ
ency f [Hz]');
                j = j + 1;
            end;
        end;
        i = i + 1;
    end;
end;


% --- Executes on button press in pushbutton_quit.
function pushbutton_quit_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_quit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global gConnectionInitalised;
global gSerialHandle;

% If quit was pushed, close the connection if its openned
if gConnectionInitalised == 1
    fclose(gSerialHandle);
    delete(gSerialHandle);
end;

% And delete the whole programm
delete(handles.root);


% --- Executes when user attempts to close root.
function root_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to root (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
global gConnectionInitalised;
global gSerialHandle;

% If quit was pushed, close the connection if its openned
if gConnectionInitalised == 1
    fclose(gSerialHandle);
    delete(gSerialHandle);
end;

% And delete the whole programm
delete(handles.root);


% --- Executes on button press in pushbutton_open.
function pushbutton_open_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_open (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
global gReadNWrite;
global gFileName;

% If data is not logged now
if gReadNWrite == 1
    % Open window to open a file and get the filename
    [filename, pathname, filterindex] = uigetfile('d:\tmp\*.bin','Open serial data
file');
    if filename ~= 0
        gFileName = strcat(pathname,filename);
    end
end;

function edit_com_text_Callback(hObject, eventdata, handles)
% hObject    handle to edit_com_text (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_com_text as text
%        str2double(get(hObject,'String')) returns contents of edit_com_text as a
double
% No action on entering a text for the com-port


% --- Executes during object creation, after setting all properties.
function edit_com_text_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_com_text (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in togglebutton_filter.
function togglebutton_filter_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton_filter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton_filter
global gMedianFilter;
gMedianFilter = get(hObject,'Value');


% --- Executes on button press in pushbutton_saveWorkspace.
function pushbutton_saveWorkspace_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_saveWorkspace (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global gTemperatureValues;
global gTime;

% Oopen the new file window
[filename, pathname, filterindex] = uiputfile('d:\tmp\*.mat','Save data as');

% Check the correctness of the file and save the variables to it
if filename ~= 0
    newFile = strcat(pathname,filename);
    save(newFile,'gTime','gTemperatureValues','-mat');
end;
```

14.7.2 Funktion tempArrayLogData

```
%--------------------------------------------------------------------------%
% Function: [] = tempArrayLogData()                                        %
%--------------------------------------------------------------------------%
% Description: This function is called everytime, 16 byte are arrived or  %
%              1 sec is over. The data is read then and saved to the       %
%              specified file.                                             %
%--------------------------------------------------------------------------%
% Author: Furrer Dominic                                                   %
%--------------------------------------------------------------------------%
% Version: 1.0                                                             %
%--------------------------------------------------------------------------%
function [] = tempArrayLogData()

% Necessary global variables
global gLogFlag;
global gFileName;
global gSerialHandle;

% If the data has to be logged
if gLogFlag == 1
    % Get the number of bytes to write
    nBytes = get(gSerialHandle,'BytesAvailable');
    if nBytes > 0
        % Write the data
        file = fopen(gFileName,'a');
        tempData = fread(gSerialHandle, nBytes);
        fwrite(file, tempData);
        fclose(file);
    end;
end;
```

14.7.3 Funktion tempArrayCalcData

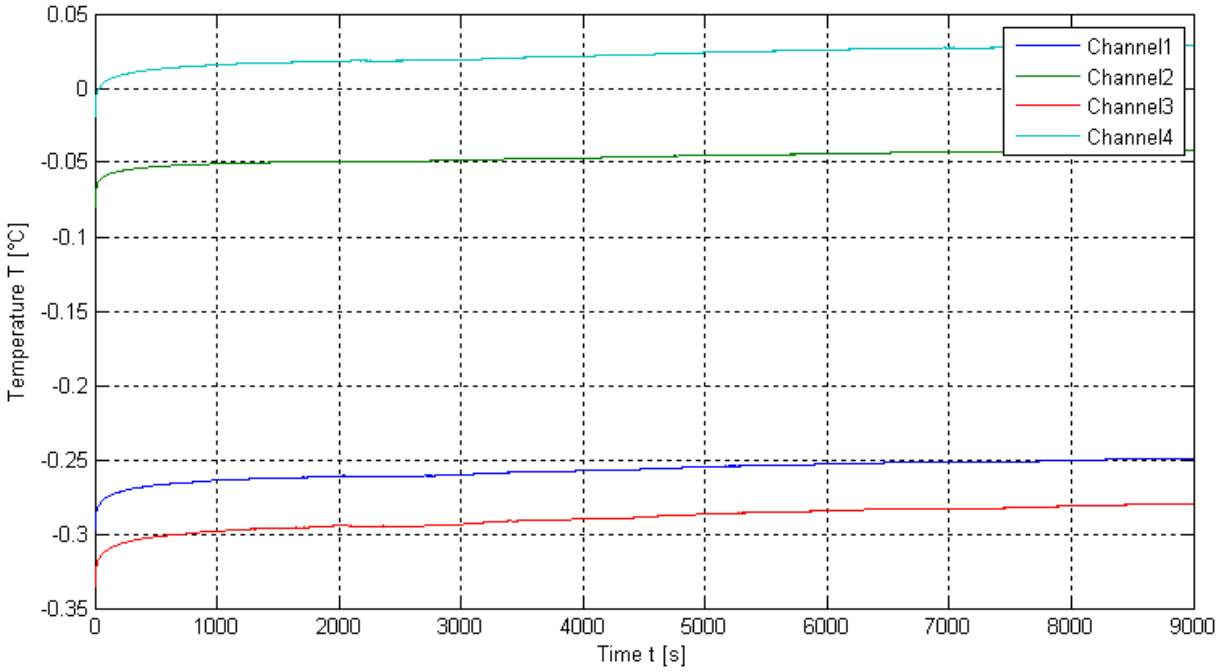
```
%--------------------------------------------------------------------------%
% Function: [] = tempArrayCalcData()                                       %
%--------------------------------------------------------------------------%
% Description: This functions reads measured data from a file and          %
%              calculates the corresponding values. Then the calculated    %
%              values are saved into global variables.                     %
%--------------------------------------------------------------------------%
% Author: Furrer Dominic                                                   %
%--------------------------------------------------------------------------%
% Version: 1.0                                                             %
%--------------------------------------------------------------------------%
function [] = tempArrayCalcData()

% Global variables that are necessary
global gFileName;
global gChannelSelection;
global gTime;
global gTemperatureValues;
global gMedianFilter;
global gNbrOfChan;

% Constant definitions, depending on the ADC, the analog circuit and the
% PT1000 resistance thermometer
vRef = 2.4804;                          % Reference voltage
coeff = (vRef/2)/((2^23)-1);            % Conversion coefficient of the ADC
vRefAcq = vRef * (8450/(8450+8660));    % For the amp stage
iMes = vRef / 2430;                     % Constant current from the source
alpha = 0.00385;                        % PT1000 temperature coefficient
r0 = 1000;                              % Zero point resistance of PT1000
lCable = 3;                             % Cable length in meter
rho = 0.0178;                           % Specific resistance
ACable = (0.5^2)*pi;                    % Cable diameter in mm^2
rCable = rho * (lCable/ACable);         % Cable resistance

% Reset variables
adData = 0;
i = 1;
j = 1;
nbrOfChan = 1;
tempValues = [];
gTemperatureValues = [];
gTime = [];

% If there is a file
if gFileName ~= 0

    % Open the file
    file = fopen(gFileName);

    % Get the file info
    fileInfo = dir(gFileName);
    fileSize = fileInfo.bytes;
    fileSize = fileSize / 32;

    % Count number of selected channels and get their
    % position
    while j < gNbrOfChan+1
        if gChannelSelection(j) == 1
            chanPos(j) = nbrOfChan;
            nbrOfChan = nbrOfChan + 1;
        end;
        j = j + 1;
    end;

    % As long as the file isn't read completely
    while i < fileSize
```

```
        j = 1;
        % Read one measure from every channel
        while j < gNbrOfChan+1
            id = 0;
            val = 0;

            % Get the values from the file
            id = fread(file,1,'uint8');
            val = fread(file,1,'bit24','b');
            %dummy = fread(file,1,'uint8');

            % If the data is incorrect, read until the next correct data
            % arrives
            if id > gNbrOfChan-1
                while id > gNbrOfChan-1
                    id = fread(file,1,'uint8');
                end;
                val = fread(file,1,'bit24','b');
                %dummy = fread(file,1,'uint8');
            end;

            % Only if a channel is selected save the read data
            if gChannelSelection(id+1) == 1
                adData(chanPos(id+1),i) = val;
            end;

            j = j + 1;
        end;
        i = i + 1;
    end;


    % Close the file
    fclose(file);

    % If some data was saved
    nbrOfMesures = size(adData);
    if nbrOfMesures(2) > 0

        % Calculate temperature values
        uAd = adData * coeff;           % Calculate the ADC-Input_voltage
        uMes = (uAd / 3) + vRefAcq;     % Calculate the PT1000-voltage
        rMes = uMes / iMes;             % Calculate PT1000-resistance
        rMes = rMes - rCable;           % Correct cable error
        tMes = ((rMes / r0) - 1) * (1 / alpha); % Finaly, the temperature

        % Filter the data, if user choses so
        if gMedianFilter == 1
            nbrOfMesures = size(tMes);
            j = 1;
            while j < nbrOfChan
                if nbrOfMesures(1) >= j

                    gTemperatureValues(j,1:i-1) = fastmedfilt1d(tMes(j,1:i-1));
                end;
                j = j + 1;
            end;
        else
            gTemperatureValues = tMes;
        end;

        gTime = 0:(nbrOfMesures(2)-1);              % Save the measuretime
    end;
end;
```

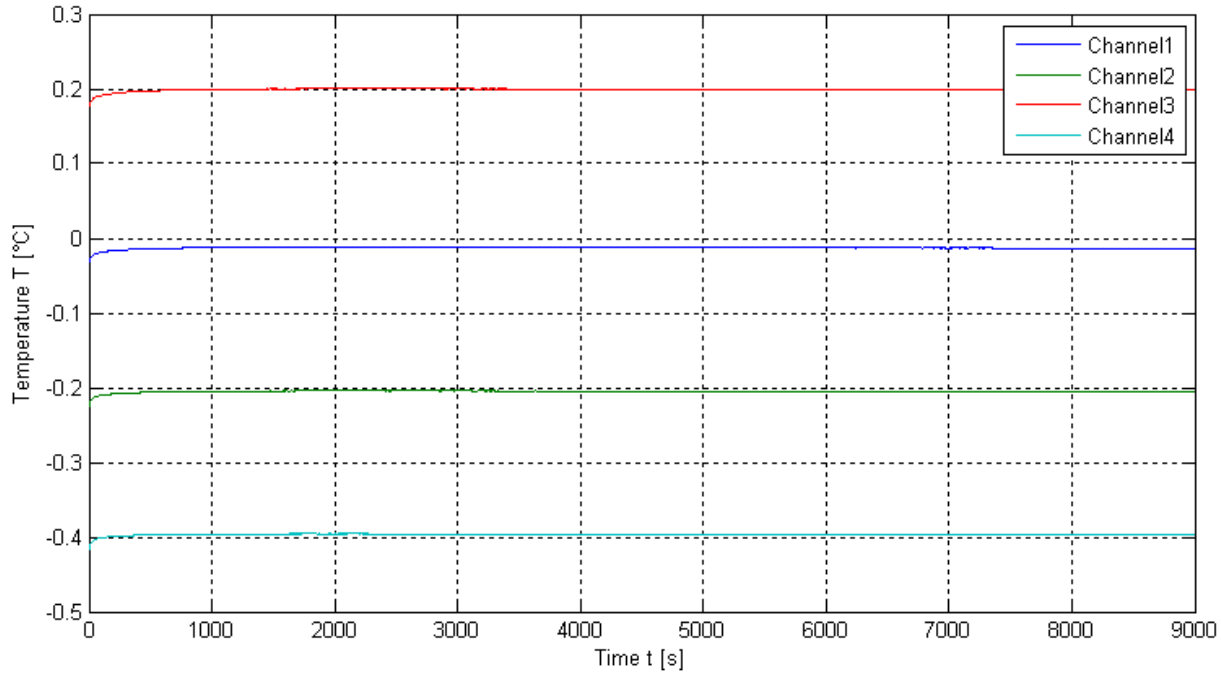
14.8 Messungen

14.8.1 Messungen des Offsets

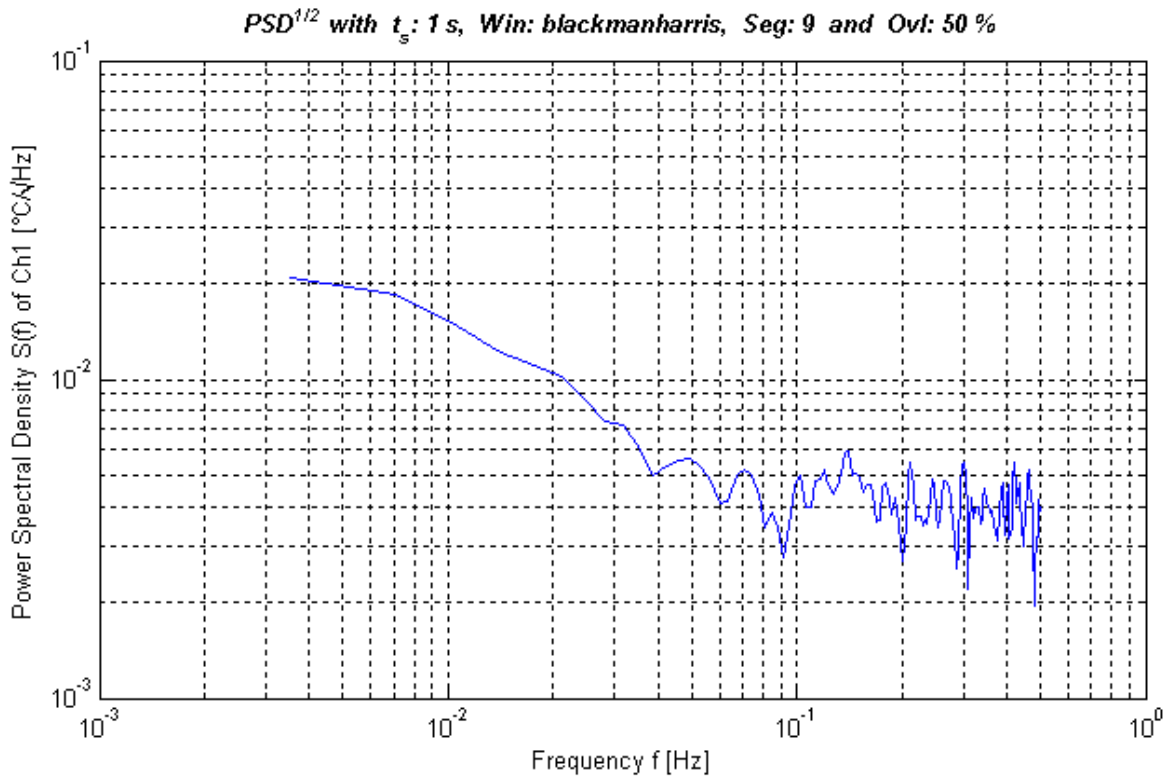
Normal



Vertauschte Stromquellen

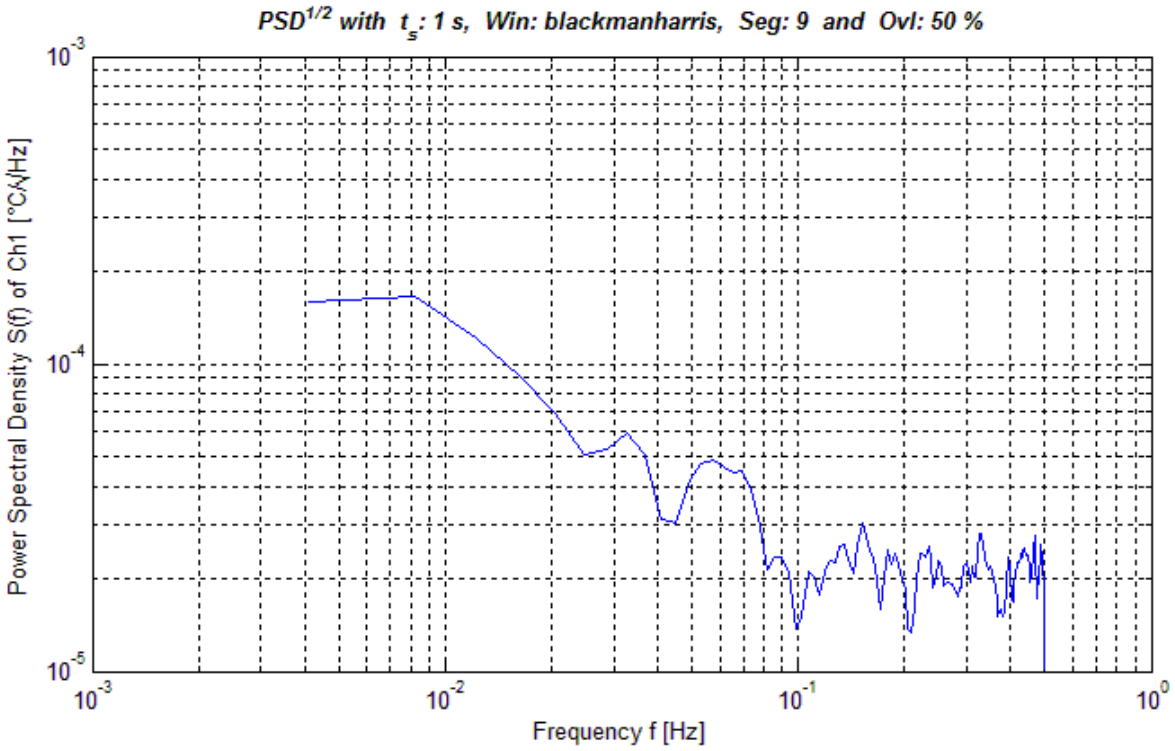


14.8.2 Messungen des Einfluss der Qualität der Spannungsreferenz

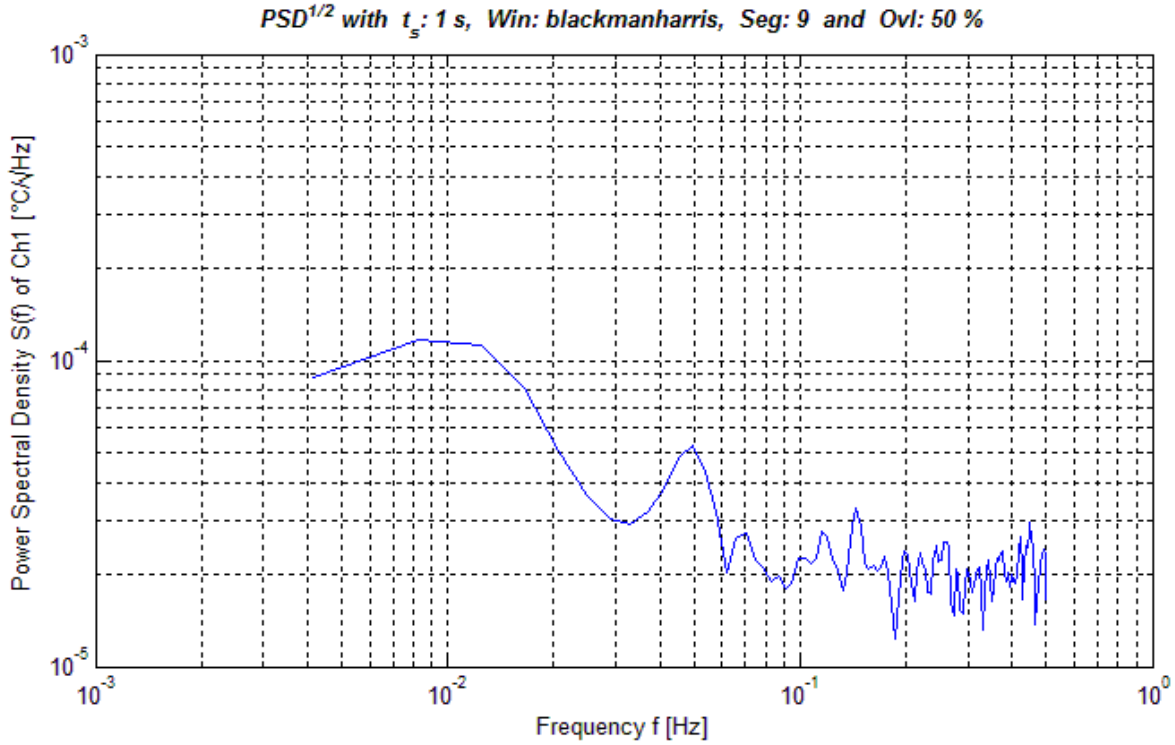


14.8.3 Messungen des Einfluss des Bypass-Kondensators der A/D-Wandler

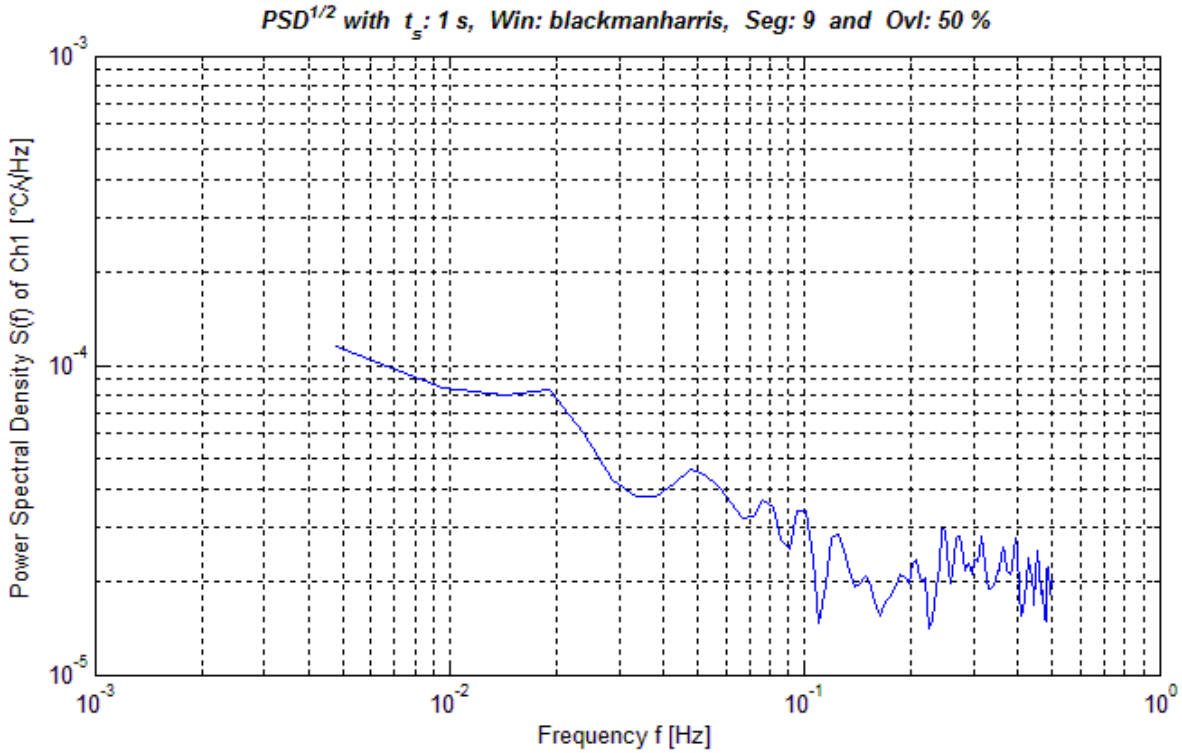
MKC4



MKS4

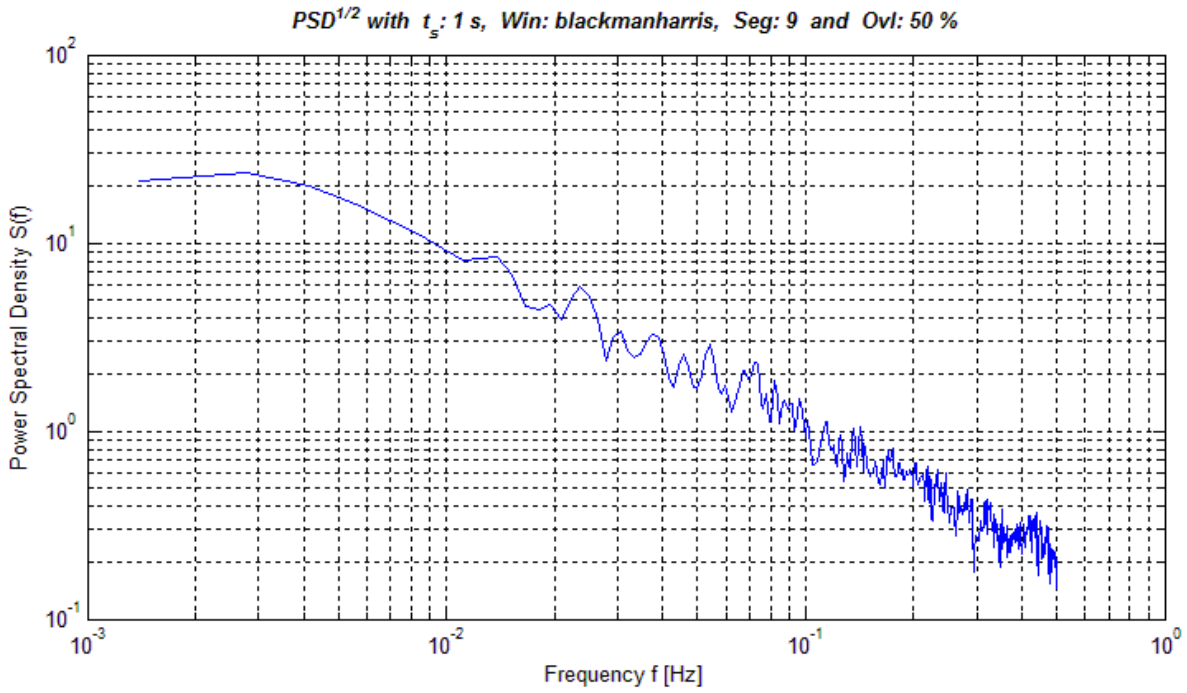


Tantal


PSD^1/2 with t_s: 1 s, Win: blackmanharris, Seg: 9 and Ovl: 50 %
Power Spectral Density S(f) of Ch1 [°C/√Hz]
Frequency f [Hz]
10^-3
10^-4
10^-5
10^-3
10^-2
10^-1
10^0

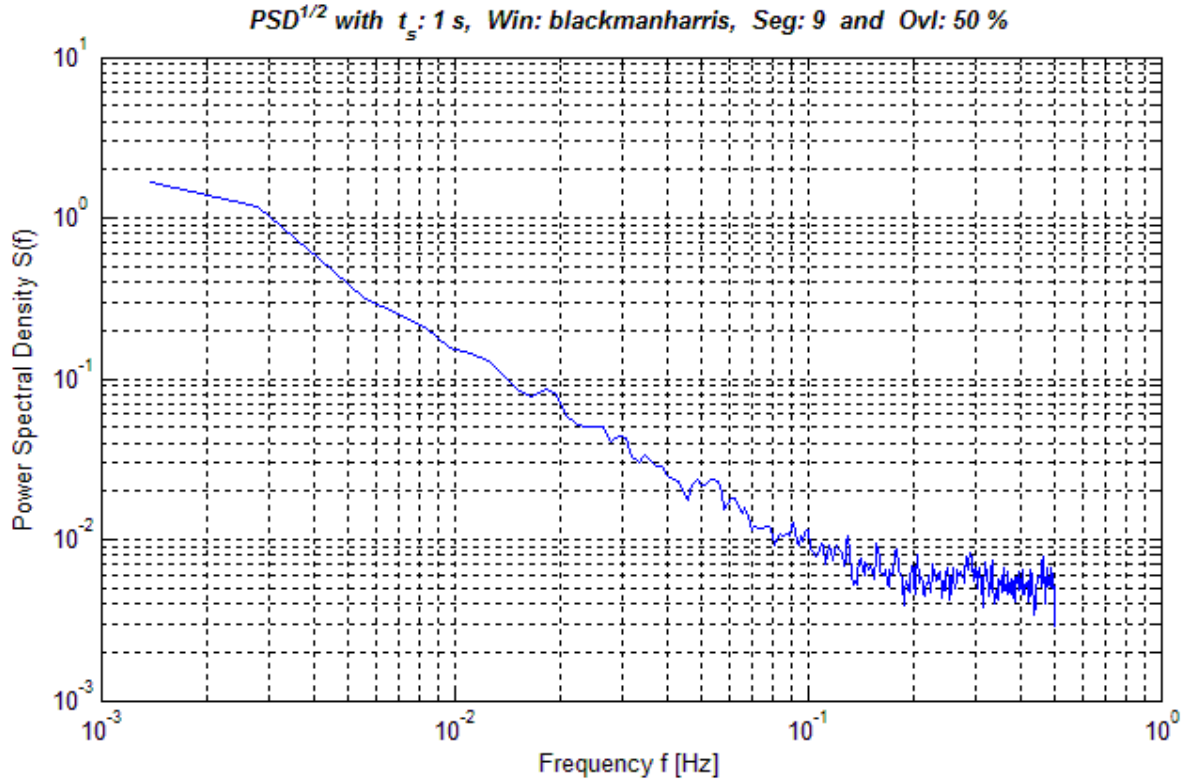
14.8.4 Messungen des Einfluss der Signalpfad-Kondensatoren

Ohne Signalpfad-Filterungs-Kondensatoren



14.8.5 Messungen des Einfluss der Referenzspannungs-Kondensatoren

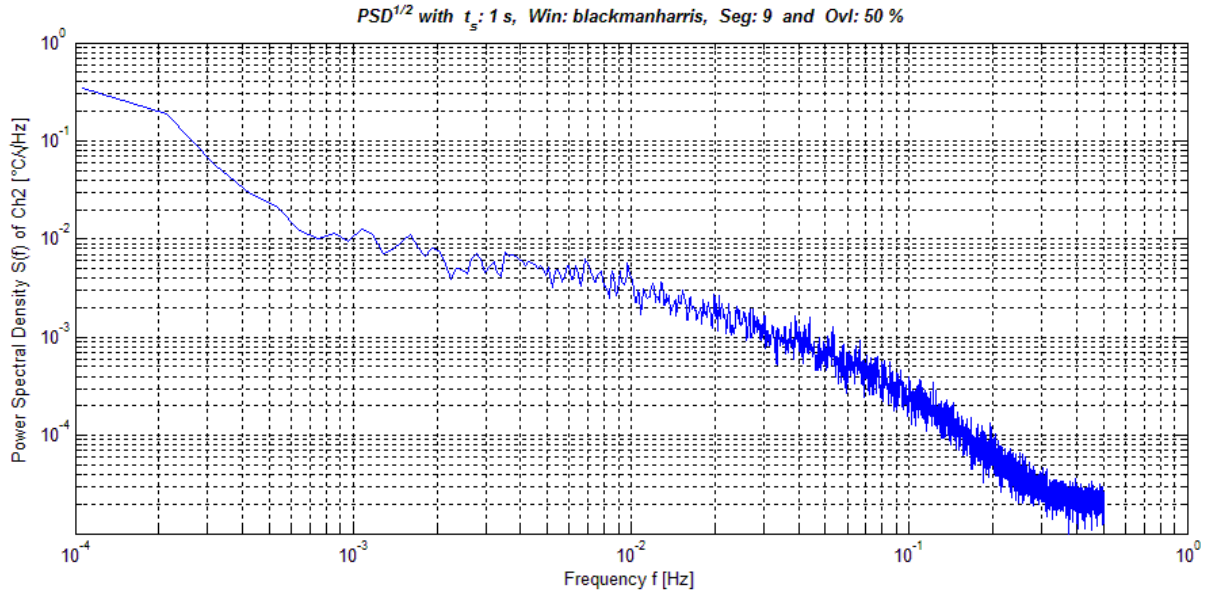
Ohne Filterungskondensatoren



14.8.6 Feldmessungen

Messung der Präzision des Kanals 2 und 4

Kanal 2



Kanal 4