Technische Universität Graz
**Institut für Elektronik – IFE**
Inffeldgasse 12
A–8010 GRAZ

# Design und Implementierung einer integrierten Abklingzeitmessung für opto-chemische Sensoren.

## Masterarbeit

Andreas Schuster

**Institut für Elektronik – IFE**
Leitung: Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Pribyl

in Zusammenarbeit mit
**Institut für Chemische Prozessentwicklung und -kontrolle**
Joanneum Research
Leitung: Ao.Univ.-Prof. Dr. Volker Ribitsch

Betreuer: *Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Pribyl*
Mitbetreuer: *Dipl.-Ing.(FH) Christian Konrad (JR)*

# Abstract

This work deals with direct measurement of decay-times from opto-chemical sensors. Amongst other things opto-chemical sensors include a sensitive membrane and some opto-electronic controlling and analysis device. The Institute of Chemical Process Development and Process Control at JOANNEUM RESEARCH already developed fluorescent dyes and opto-electronic control and phase-measurement devices.

A new system for direct measurement of decay-times is presented in this work. Therefore a new hardwareplatform and the corresponding software was designed and implemented. The system is evaluated with oxygen sensitive dyes which have decay-times as low as milliseconds. The big advantage of direct decay-time measurement is the lower excitation intensity compared to the needed intensity when using phase-measurement systems. For this reason bleaching effects of the dyes are reduced which results in longer lifetime of the opto-chemical sensors.

# Kurzfassung

Diese Arbeit beschäftigt sich mit der direkten Messung der Abklingzeit von opto-chemischen Sensoren. Opto-chemische Sensoren beinhalten unter anderem eine sensitive Membran sowie eine opto-elektronische Verarbeitungseinheit. Am Institut für Chemische Prozessentwicklung und -kontrolle des JOANNEUM RESEARCH Forschungsgesellschaft mbH werden diese sensitiven Membrane sowie dazugehörige opto-elektronische Phasenmesssysteme entwickelt.

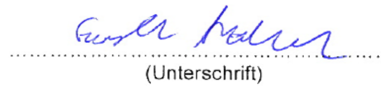
Ein neues Messsystem zur direkten Messung von Abklingzeiten wird in dieser Arbeite entwickelt und realisiert. Dazu wurde eine neue Hardwareplattform sowie die dazugehörige Software neu konzipiert und umgesetzt. Die Evaluierung dieses Systems erfolgt mit Sauerstoff sensitiven Membranen, welche Abklingzeiten bis in den einstelligen Millisekundenbereich aufweisen. Eine direkte Messung der Abklingzeiten bietet den wesentlichen Vorteil, dass die Anregungsintensität geringer ausfallen kann als bei vergleichbaren Phasenmesssystemen. Dadurch werden Ausbleichungseffekte verringert, wodurch sich die Lebensdauer opto-chemischer Sensoren verlängert.

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

## EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am 10.5.2010



(Unterschrift)

# Danksagung

Ich danke hiermit vor allem meiner Familie, welche diese Arbeit erst ermöglicht hat. Danke für den Ansporn und den Rückhalt.
Weiters danke ich Christian für alle Anregungen und Diskussionen. Den weiteren Mitarbeitern von JOANNEUM Research danke ich für die zahllosen Hilfestellungen.
Dank verpflichtet bin ich meinem Diplomarbeitsbetreuer an der TU Graz Dr. Wolfgang Pribyl für den großen Freiraum, den er mir bei der Gestaltung und praktischen Durchführung dieser Arbeit gewährt hat.

# Inhaltsverzeichnis

**1 Einleitung** **12**
1.1 Motivation . . . 12
1.2 Gliederung . . . 13
1.3 Aufgabenstellung und Ziele dieser Arbeit . . . 13

**2 Grundlagen zur Messung mit opto-chemischen Sensoren** **15**

**3 Direkte Messung von Abklingzeiten** **18**
3.1 Messaufbau für direkte Messung von Abklingzeiten . . . 18
3.1.1 Funktionsgenerator . . . 18
3.1.2 Optische Anregung . . . 18
3.1.3 Cube . . . 20
3.1.4 Optischer Filter . . . 20
3.1.5 Faserkabel FK1 . . . 22
3.1.6 Optischer Detektor . . . 22
3.1.7 Strom-Spannungswandler . . . 22
3.1.8 Oszilloskop . . . 23
3.1.9 Auswertung . . . 23
3.2 Ergebnisse . . . 24

**4 Hardwareplattform** **27**
4.1 Komponenten für die Hardwareplattform . . . 27
4.2 Zusätzliche externe Komponenten . . . 29
4.2.1 Treiber für optische Anregung . . . 29
4.3 Anforderungen . . . 30
4.4 SKØL-Board Hardwareplattform . . . 31
4.4.1 FPGA . . . 31
4.4.2 Analog-digital-Wandler . . . 32
4.4.3 Differentieller Eingangstreiber für ADC . . . 35
4.4.4 Peripherie, weitere Komponenten . . . 37
4.4.5 Board Übersicht, Jumper, IO's . . . 40
4.5 Evaluierung . . . 40
4.5.1 Testaufbau . . . 41
4.5.2 Rohdaten Aufnahme . . . 42
4.5.3 Berechnung NAD, SINAD, ENOB . . . 42
4.5.4 Ermittlung SNR . . . 44

4.5.5 Auswertung . . . . . 44

5 Konfiguration der Hardwareplattform 46
5.1 FPGA Konfiguration . . . . . 46
5.2 Softcore Mikrocontroller . . . . . 49
5.3 Interrupts, Trigger . . . . . 49
5.4 Speicher . . . . . 50
5.5 Einlesen der Daten vom ADC . . . . . 50
5.6 Clock Divider . . . . . 51

6 Verfahren zur direkten Abklingzeitmessung 55
6.1 Messung Grundgerüst . . . . . 55
6.2 Initiale Messung . . . . . 56
6.2.1 Wahl des Speicherfensters . . . . . 56
6.2.2 Ermittelung der ungefilterten Signale und Parameter . 58

7 Evaluierung der geeignetsten Parameter 62
7.1 Automatisches Evaluierungs und Bewertungssystem . . . . . 62
7.2 Filter und Algorithmen . . . . . 62
7.2.1 Stufe0: digitale Filter . . . . . 62
7.2.2 Stufe1: Ermittelung Start der Abklingfunktion . . . . . 63
7.2.3 Stufe2: Ermittelung TAU . . . . . 63
7.2.4 Zusammenfassung, Auswertung . . . . . 64
7.3 Ergebnisse . . . . . 65
7.3.1 Auszug aus den Algorithmen Stufe 1 . . . . . 66
7.3.2 Auszug aus den Algorithmen Stufe 2 . . . . . 66
7.4 Evaluierung der gefundenen Parameter . . . . . 69
7.5 Erhöhung der Trainingsdaten für RuDPP . . . . . 75

8 Zusammenfassung, Diskussion, Ausblick 78
8.1 Ausblick . . . . . 78

Literaturverzeichnis 80

Anhang 82
Fotos . . . . . 82
Datenblätter . . . . . 84
Schaltpläne . . . . . 103
FPGA Konfiguration . . . . . 116

Matlab Code . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 200

# Abbildungsverzeichnis


2.1 Pulsverfahren . . . 16
2.2 Phasenmessmethode . . . 17
3.1 Messaufbau für direkte Abklingzeitmessung . . . 19
3.2 Alle Spektren LED NSPB500 und PtTFPP Sensor und Emissionsfilter RG610 . . . 21
3.3 RuDPP mit übersteuertem und korrekt eingestelltem PMT Detektor . . . 23
3.4 Strom-Spannungswandlung . . . 24
3.5 PtTFPP Air, *Kurvenfit* mit ®Origin . . . 25
4.1 Hardware Komponenten . . . 28
4.2 LED Treiber . . . 29
4.3 Exponentieller Abfall . . . 31
4.4 AD9461 Schaltplan Auszug . . . 34
4.5 AD8138 Schaltplan . . . 36
4.6 AD8138 Frequenzabhängige Verstärkungskurve. ©Analog Devices . . . 37
4.7 MAX3232ECAE Schaltung . . . 38
4.8 XCF08P Schaltung . . . 39
4.9 Programmierung des FPGA mit Config PROM mittels *Master Serial Mode*. ©Xilinx . . . 39
4.10 Übersicht der Hardwareplattform Platine (Blick von oben) . . . 41
4.11 Sinus *gefitted* . . . 43
4.12 *Visual Analog* Konfiguration . . . 44
5.1 FPGA Module . . . 46
5.2 FPGA Komponenten (Mikrocontroller Peripherie Seite) . . . 47
5.3 FPGA Komponenten (Mikrocontroller Speicher Seite) . . . 48
5.4 Mittelwertbildung 64 Ablauf . . . 52
5.5 VHDL Modul Statemachine . . . 53
6.1 Mikrocontroller Ablauf Statemachine . . . 55
6.2 PtTFPP Messkurve, Mittelwert 64, Taktteiler 32, Umgebung Luft und N2 . . . 59
6.3 Ermittelte Parameter der ungefilterten Messkurve einer Beispielkurve . . . 61
7.1 PtTFPP (Taktteiler 32) N2, Messsignal mit und ohne Filterung - vergrößerte Abbildung . . . 71
7.2 Messreihe mit wechselnder Umgebung (Luft - N2 - Luft) . . . 72

7.3 Messreihe PtTFPP Taktteiler64 Luft und Standardabweichung 73
7.4 RuDPP - Vergleich TAU Messung und Standardabweichung . 77
8.1 Messaufbau, Ansicht von oben. . . . . . . . . . . . . . . . . . 82
8.2 Printed Circuit Board . . . . . . . . . . . . . . . . . . . . . . 83
8.3 Datenblatt - Transistor BC547 . . . . . . . . . . . . . . . . . . 84
8.4 Datenblatt - RS232 Teil 1 . . . . . . . . . . . . . . . . . . . . 85
8.5 Datenblatt - RS232 Teil 2 . . . . . . . . . . . . . . . . . . . . 86
8.6 Datenblatt - RS232 Teil 3 . . . . . . . . . . . . . . . . . . . . 87
8.7 Datenblatt - Xilinx Config PROM . . . . . . . . . . . . . . . . 88
8.8 Datenblatt - PMT Teil 1 . . . . . . . . . . . . . . . . . . . . . 89
8.9 Datenblatt - PMT Teil 2 . . . . . . . . . . . . . . . . . . . . . 90
8.10 Datenblatt - Spannungsregler LT1964 . . . . . . . . . . . . . . 91
8.11 Datenblatt - Oszillator . . . . . . . . . . . . . . . . . . . . . . 92
8.12 Datenblatt - Spannungsregler LM1086 . . . . . . . . . . . . . . 93
8.13 Datenblatt - Spannungsregler LT1964 . . . . . . . . . . . . . . 94
8.14 Datenblatt - Spannungsregler LM1084 . . . . . . . . . . . . . . 95
8.15 Datenblatt - Funktionsgenerator . . . . . . . . . . . . . . . . . 96
8.16 Datenblatt - LED . . . . . . . . . . . . . . . . . . . . . . . . . 97
8.17 Datenblatt - FPGA . . . . . . . . . . . . . . . . . . . . . . . . 98
8.18 Datenblatt - ADC Teil 1 . . . . . . . . . . . . . . . . . . . . . 99
8.19 Datenblatt - ADC Teil 2 . . . . . . . . . . . . . . . . . . . . . 100
8.20 Datenblatt - ADC Teil 3 . . . . . . . . . . . . . . . . . . . . . 101
8.21 Datenblatt - Spanungsregler TI . . . . . . . . . . . . . . . . . . 102
8.22 Schaltplan Top Level . . . . . . . . . . . . . . . . . . . . . . . 103
8.23 Schaltplan Differentieller Eingangstreiber . . . . . . . . . . . . 104
8.24 Schaltplan Strom-Spannungswandler, Led Treiber . . . . . . . 105
8.25 Schaltplan FPGA . . . . . . . . . . . . . . . . . . . . . . . . . 106
8.26 Schaltplan FRAM . . . . . . . . . . . . . . . . . . . . . . . . . 107
8.27 Schaltplan LED's, IO's . . . . . . . . . . . . . . . . . . . . . . 108
8.28 Schaltplan RS232 . . . . . . . . . . . . . . . . . . . . . . . . . 109
8.29 Schaltplan Spannungsversorgung Teil 1 . . . . . . . . . . . . . 110
8.30 Schaltplan Spannungsversorgung Teil 2 . . . . . . . . . . . . . 111
8.31 Schaltplan IO Erweiterungen . . . . . . . . . . . . . . . . . . . 112
8.32 Schaltplan Oszillator . . . . . . . . . . . . . . . . . . . . . . . 113
8.33 Schaltplan Erweiterungs ADC und DAC . . . . . . . . . . . . . 114
8.34 Schaltplan Config PROM . . . . . . . . . . . . . . . . . . . . . 115
8.35 FPGA Konfiguration . . . . . . . . . . . . . . . . . . . . . . . 116
8.36 Taktteiler Modul . . . . . . . . . . . . . . . . . . . . . . . . . 117

# Tabellenverzeichnis


3.1 Messaufbau - Einstellung Funktionsgenerator . . . . . . . . . . 18
3.2 Benötigte Anregungsspektren für opto-chemische Sensoren . . 20
3.3 Emissionsfilter für opto-chemische Sensoren . . . . . . . . . . 21
3.4 Hamamatsu PMT H6780-20 . . . . . . . . . . . . . . . . . . . . 22
3.5 PtTFPP Air, *Kurvenfit* Statistik . . . . . . . . . . . . . . . . 24
3.6 PtTFPP Air, *Kurvenfit* Ergebnisse Beispiel . . . . . . . . . . . 25
3.7 Abklingzeiten (Oszilloskop und Mathematik Software) . . . . . 26
4.1 ®Xilinx ™Spartan 3 XC3S1500-4FG676C Merkmale . . . . . 32
4.2 Jumper I1_ConfigModeHdr Konfigurationen . . . . . . . . . . 32
4.3 Jumper I1_Bank6VCC und I1_Bank7VCC Konfigurationen . 32
4.4 AD9461 Jumper Einstellung I4_ADC_SFDR . . . . . . . . . 33
4.5 AD9461 Jumper Einstellung I4_ADC_DFS . . . . . . . . . . 35
4.6 Widerstandsanpassung an Signalquelle für AD8138 . . . . . . 36
4.7 Analog Digital Wandlung, Verstärkung, Auflösung . . . . . . . 36
4.8 Übersicht Spannungsversorgung . . . . . . . . . . . . . . . . . 37
4.9 Beschreibung zu Abbildung 4.10 . . . . . . . . . . . . . . . . . 40
4.10 Evaluierung Übersicht (Mittelwert 1) . . . . . . . . . . . . . . 45
4.11 Evaluierung Übersicht (Mittelwert 64) . . . . . . . . . . . . . 45
5.1 TSK3000A Eigenschaften . . . . . . . . . . . . . . . . . . . . . 49
5.2 Speicheraufteilung Datenspeicher . . . . . . . . . . . . . . . . 51
5.3 ClockDivider Modul Daten . . . . . . . . . . . . . . . . . . . . 54
6.1 PtTFPP Wahl des Taktteilers bzw. Speicherfenster . . . . . . 57
6.2 RuDPP Wahl des Taktteilers bzw. Speicherfenster . . . . . . . 58
6.3 PdTFPP Wahl des Taktteilers bzw. Speicherfensters . . . . . . 59
7.1 Laufzeit einer Messkurven-Evaluierung (®Intel ™Core2 DUO 3GHz, 3GB Ram, Windows XP) . . . . . . . . . . . . . . . . . . 64
7.2 PdTFPP (Taktteiler 512) Berechnungsergebnis (bestes Ergebnis) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65
7.3 Ergebnisse der Evaluierung der besten Parameter - pro Umgebung jeweils 1 Messreihe als Trainigsdaten . . . . . . . . . . 65
7.4 PtTFPP Ergebnisse (Messreihe mit 50 Samples) . . . . . . . . 73
7.5 PdTFPP Ergebnisse (Messreihe mit 50 Samples) Taktteiler 512 74
7.6 PdTFPP Ergebnisse (Messreihe mit 50 Samples) Taktteiler 1024 74
7.7 RuDPP Ergebnisse (Messreihe mit 50 Samples) . . . . . . . . 74
7.8 Ergebnisse der Evaluierung der besten Parameter mit 5 Messreihen als Trainigsdaten für RuDPP . . . . . . . . . . . . . . . 75

7.9 RuDPP Ergebnisse, Vergleich bei Erhöhung der Trainingsdaten 76
8.1 Zusammenfassung der besten Ergebnisse . . . . . . . . . . . . . 78

# 1 Einleitung

Am Institut für Chemische Prozessentwicklung und Kontrolle des JOANNEUM werden Sensoren entwickelt, welche auf der Detektion der Änderung der Fluoreszenzabklingzeit von Farbstoffen beruhen. Diese Änderung wird durch Variation verschiedener Umgebungsbedingungen (Sauerstoffgehalt eines Gases, pH-Wert einer Lösung, usw.) hervorgerufen. Diese Sensoren bestehen aus einer sensitiven Schicht, welche auch den Farbstoff beinhaltet, einer geeigneten Optik um den Farbstoff anzuregen und um die Fluoreszenz detektieren zu können sowie einer elektronischen Auswerteeinheit [14]. Durch die immer stärker werdende Miniaturisierung der dazu notwendingen optoelektronischen und elektronischen Komponenten vermehren sich auch die Anwendungsfelder dieser Sensorsysteme bis hin zu z.B. invasiven Messungen von metabolischen Parametern bei Mensch und Tier [1] [5]. Erst durch geeignete Messsysteme, welche in der Lage sind immer schnellere Abklingzeiten bei kompaktem Aufbau zu bestimmen, steigern sich auch die Einsatzszenarien für opto-chemische Sensoren.

## 1.1 Motivation

Der vermehrte Einsatz opto-chemischer Sensoren zeigt deutlich die steigende Relevanz sich mit der Thematik elektro-optischer Messsysteme zu befassen. Für Phosphoreszenzfarbstoffe müssen Abklingzeiten im Millisekundenbereich bis zum Mikrosekundenbereich detektiert werden. Bei Fluoreszenzfarbstoffen liegen die Abklingzeiten im Nanosekunden- bis Pikosekundenbereich. Die Messsysteme hierfür lassen sich in Phasenmessysteme und Systeme zur direkten Messung von Abklingzeiten einteilen.
Seit ca. 15 Jahren werden am Institut für Chemische Prozessentwicklung und -kontrolle von JOANNEUM opto-chemische Sensoren entwickelt [14]. Diese Sensoren dienen zur Bestimmung verschiedenster Analyte. Die verfügbaren Messsysteme (Phasenmesssysteme) erlauben zur Zeit nur eine Messung bis ca. 10µs [2].
Das wesentliche Ziel dieser Masterarbeit ist die Entwicklung eines kompakten Messsystems, welches durch direkte Messung der Abklingzeit folgende Vorteile aufweist.
Die Anregungsintensität kann geringer als bei vergleichbaren Phasenmesssystemen ausfallen. Dadurch verlängert sich die Lebensdauer opto-chemischer Sensoren, da Ausbleichungseffekte geringer ausfallen.

Der bei der Phasenmessung benötigte Detektionsfilter kann bei der direkten Messung wegfallen, welches die Baugröße und die Kosten reduziert.

## 1.2 Gliederung

Diese Arbeit gliedert sich in 3 Hauptteile.

**Der erste Teil** liefert einen Überblick über die notwendigen theoretischen Grundlagen, gefolgt von praktischen Ausführungen. *Kapitel 1* beinhaltet die Motivation, die Gliederung sowie die Aufgabenstellung dieser Arbeit. *Kapitel 2* beschäftigt sich mit den Grundlagen zur Thematik opto-chemischer Sensoren. *Kapitel 3* widmet sich dem optischen Messaufbau sowie den Grundlagen zur direkten Messung der Abklingzeiten.

**Im zweiten Teil** werden die im Rahmen dieser Arbeit entwickelten Komponenten beschrieben. *Kapitel 4* spezifiziert die Hardwareplattform, welche als Kernkomponente einen rekonfigurierbaren Logikbaustein sowie eine Analog-Digital Wandlung bereitstellt. Die Konfiguration der Hardwareplattform wird in *Kapitel 5* erläutert. Das Verfahren zur direkten Messung von Abklingzeiten ist in *Kapitel 6* beschrieben.

**Der dritte und letzte Teil** zeigt in *Kapitel 7* die Anwendung und Evaluierung des Messsystems. *Kapitel 8* schließt diese Arbeit mit der Zusammenfassung und dem Ausblick.

## 1.3 Aufgabenstellung und Ziele dieser Arbeit

Es soll ein System zur direkten Bestimmung der Fluoreszenzabklingzeit, welches folgende Anforderungen erfüllen soll, entwickelt werden:

- Messung von Abklingzeiten im Bereich von 1ms bis 1µs.
- Messdatenausgabe: ca. 1 Wert pro Sekunde.
- Getreue Aufnahme der Kurve, um für Auswertungszwecke die Art der Abklingzeit bestimmen zu können (Monoexponentiell, Multiexponentiell).

Dabei sollen im ersten Schritt Referenzmessungen mittels eines hochauflösenden Oszilloskopes durchgeführt werden. Aus diesen Referenzmessungen wird mittels eines Mathematikprogrammes ein Referenzwert für die Abklingzeit der jeweiligen Sensoren berechnet.
Zur direkten Messung der Abklingzeit soll eine Hardwareplattform entwickelt werden, welche es schlussendlich erlaubt, die Abklingzeitmessung in einem integrierten Modul durchzuführen. Diese Hardwareplattform soll auch als Plattform für die zukünftige Portierung weiterer Anwendungen dienen (Phasenmessung [2], etc.). Auf der Hardwareplattform sollen als Hauptkomponenten ein FPGA (Field Programmable Gate Array) und ein Analog-digital-Wandler integriert sein. Zur Berechnung der Daten und Steuerung der Berechnung der Abklingzeiten soll ein Mikrocontroller (als Softcore) auf dem FPGA dienen.
Folgende Schnittstellen sollen auf der Hardwareplattform verfügbar sein:

- Steuerausgang (Trigger) für die optische Anregung
- Single-ended Spannungseingang mit folgenden Bereichen (+-400mV, +-4V)
- Digitales Interface zu Host (RS232 etc.)

Mittels der Hardwareplattform sollen Messkurven der Abklingzeiten in verschiedenen Auflösungen erfasst werden (initiale Messungen). Verschiedene Algorithmen zur direkten Berechnung der Abklingzeiten sollen mittels eines zu erstellendes MATLAB-Programms getestet werden. Dabei dienen die Referenzwerte der Abklingzeitmessungen (aus den Messungen mittels des hochauflösenden Oszilloskopes) als Zielvorgabe, und die Messkurven der initalen Messungen als Lernkurven. Die durch das MATLAB-Programm ermittelten geeignetsten Algorithmen und deren Parameter, sind danach in den Mikrocontroller zu implementieren und zu evaluieren.

# 2 Grundlagen zur Messung mit opto-chemischen Sensoren

Sensoren sind Geräte, welche es unter anderem erlauben, physikalische Parameter oder die chemischen Eigenschaften eines Stoffes zu erfassen. Zur Weiterverarbeitung dieser Größen dient meist ein elektrisches Signal am Ausgang des Sensors. [4]

**Lumineszenz** bezeichnet die Erzeugung von Licht. Darunter fallen eine Vielzahl von Anregungsarten, jedoch ausgenommen sind Methoden die über Emission oder Gasentladung zur Emission führen [13]. Der Begriff Lumineszenz dient dabei als übergeordneter Begriff für die Erzeugung von Licht, bei der eine Anregung eines Elektrons (durch Einstrahlen von Licht passender Frequenz, elektrischen oder chemischen Effekten) erfolgt. Zur Lumineszenz gehören Phosphoreszenz und Fluoreszenz. [12]

**Fluoreszenz** tritt auf, wenn ein angeregtes Atom unmittelbar ($\leq$ 10 ns) auf sein ursprüngliches Energieniveau zurückspringt und dabei ein Lichtquant emittiert. Wenn das Lichtquant dieselbe Frequenz hat wie die zuvor absorbierte, dann ist das Atom direkt auf sein ursprüngliches Energieniveau zurückgesprungen und man spricht von Resonanzfluoreszenz. Das Atom kann auch über mehrere andere dazwischenliegende Niveaus in seinen Ausgangszustand zurückkehren und dabei mehrere Lichtquanten verschiedener Frequenzen aussenden. Dies nennt man Fluoreszenz. [12]

**Phosphoreszenz** bezeichnet auch den Effekt, daß die absorbierte Strahlung wieder abgegeben wird. Dabei ist aber die Dauer zwischen der Anregung und der Abgabe länger als bei der Fluoreszenz. Dies kann bis zu mehrere Tage dauern. [12]

**Lebenszeit** ist definiert als die durchschnittliche Zeit in der ein Molekül in seinem angeregten Zustand verbleibt. Fluorphore Lebenszeiten beginnen typischerweise im Nanosekundenbereich und werden kleiner, phosphoreszierende Lebenszeiten hingegen sind wie vorhin beschrieben viel länger. Die Lebenszeit wird von Ausbleichungseffekten nicht beeinträchtigt. Im Vergleich zur auch angewendeten Intensitätsmessung bei Sensoren bringt dies einen

erheblichen Vorteil für die Messung der Lebenszeit. Diese Änderung der Intensität verfälscht auch nicht die Ergebnisse der Messungen . Die Lebenszeit gibt direkten Aufschluss über die Umgebungsbedingung (pH-Wert, O2, CO2 Konzentration etc.) der der Sensor ausgesetzt ist. [21]

**Opto-chemische Sensoren** erlauben das Messen von physikalischen Parametern oder chemischen Eigenschaften durch ihre Eigenschaft der Lumineszenz, in Abhängigkeit des Analyten (der zu messenden Größe). [21]
Ein frühes Beispiel hierfür, aus den 1930er Jahren, ist ein Verfahren von Kautsky und Hirsch, das auf der Änderung der Phosphoreszenz von Silica-Gel-adsorbiertem Tyrapaflavin in Abhängigkeit von Sauerstoff beruht. [21]

**Messverfahren:** Es gibt zwei verbreitete Methoden die Lebenszeit zu bestimmen, ein Pulsverfahren und die Phasenmessmethode. Beim Pulsverfahren (siehe Abbildung 2.1) wird der Sensor mit einem Lichtimpuls angeregt und der zeitabhängige Abfall der Intensität der Lumineszenz wird direkt gemessen. Aufgrund des exponentiellen Abfalles wird die Lebenszeit in TAU angegeben (siehe Gleichung (2.0.1)).

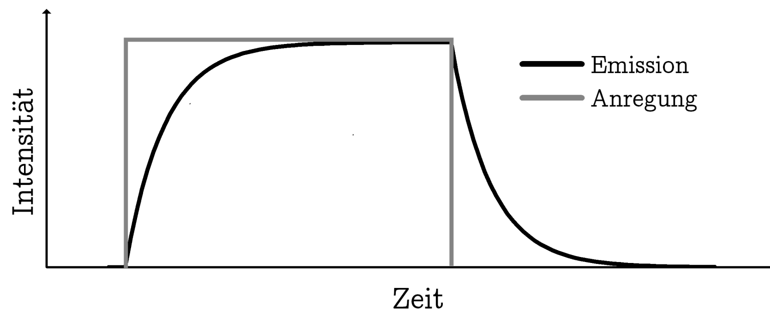



Abbildung 2.1: Pulsverfahren

$$y = e^{(\frac{-x}{\tau})} \tag{2.0.1}$$

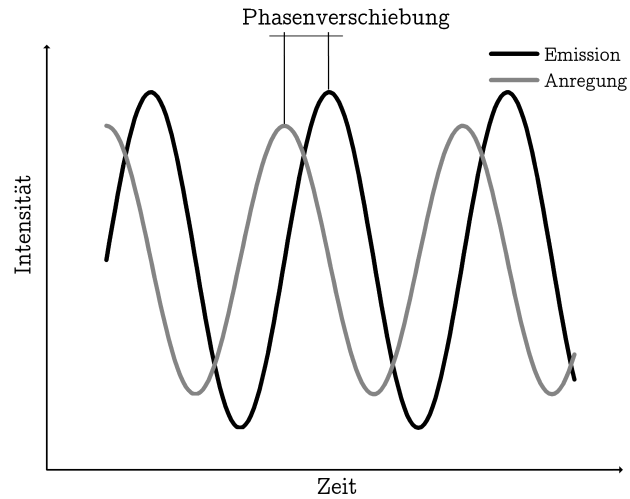



Abbildung 2.2: Phasenmessmethode

Um mit der Phasenmessmethode (Abbildung 2.2) messen zu können wird der Sensor mit sinusförmig moduliertem Licht angeregt. Danach wird die Phasenverschiebung von der angeregten Lumineszenz zur Anregung gemessen. Die Phasenverschiebung korreliert dabei mit der Lebenszeit. Für den Fall daß die Phasenverschiebung direkt proportional zur Lebenszeit ist, erhält man die Lebenszeit durch Gleichung (2.0.2). [21] [19] [11]

$$tan(Phasenverschiebung) = 2 \cdot \pi \cdot f \cdot \tau \tag{2.0.2}$$

# 3 Direkte Messung von Abklingzeiten

## 3.1 Messaufbau für direkte Messung von Abklingzeiten

Der verwendete Messaufbau ist in Abbildung 3.1 ersichtlich.
Der opto-chemische Sensor wird durch eine LED angeregt. Für die Ansteuerung mittels eines rechteckförmigen Signales wird ein Funktionsgenerator verwendet. In den folgenden Kapiteln werden die einzelnen Komponenten näher beschrieben.

### 3.1.1 Funktionsgenerator

Die LED's werden durch ein rechteckförmiges Signal angesteuert. Die verwendeten Einstellungen sind in Tabelle 3.1 ersichtlich. Der eingesetzte Funktionsgenerator ist ein HP 8116A.

| Sensor | LED | Spannung | Rechteckfrequenz |
|---:|---:|---:|---:|
| PdTFPP | NSPB500 | 3,5 V | 120 Hz |
| RUDPP | NSPB500 | 3,5 V | 4 kHz |
| PtTFPP | NSPB500 | 3,5 V | 4 kHz |

Tabelle 3.1: Messaufbau - Einstellung Funktionsgenerator

### 3.1.2 Optische Anregung

Die optische Anregung muss an den opto-chemischen Sensor angepasst werden. Für die verschiedenen Sensoren sind in Tabelle 3.2 die unterschiedlich notwendigen Bandbreiten der Spektren für die Anregung aufgeführt.
In Abbildung 3.2 sieht man das Spektrum der verwendeten NSPB500 LED als hellgraue Linie rund um 470nm. Die Änderung des Gesamtspektrums, durch einbringen eines PtTFPP Sensors, ist durch die dunkelgraue Kurve ersichtlich. Hier ist deutlich erkennbar, daß der hier verwendete PtTFPP Sensor bei ca 620nm eine markante Spitze aufweist. Dieser Bereich ist für die Messung interessant. Der verwendete Emissionsfilter RG610 filtert den Wellenlängen Bereich der LED aus (alles unter 610nm), sodaß nur die Spitze der Emission (schwarze Kurve) übrig bleibt.
Die verwendete Spannung zur Ansteuerung der LED's darf hierbei nicht zu

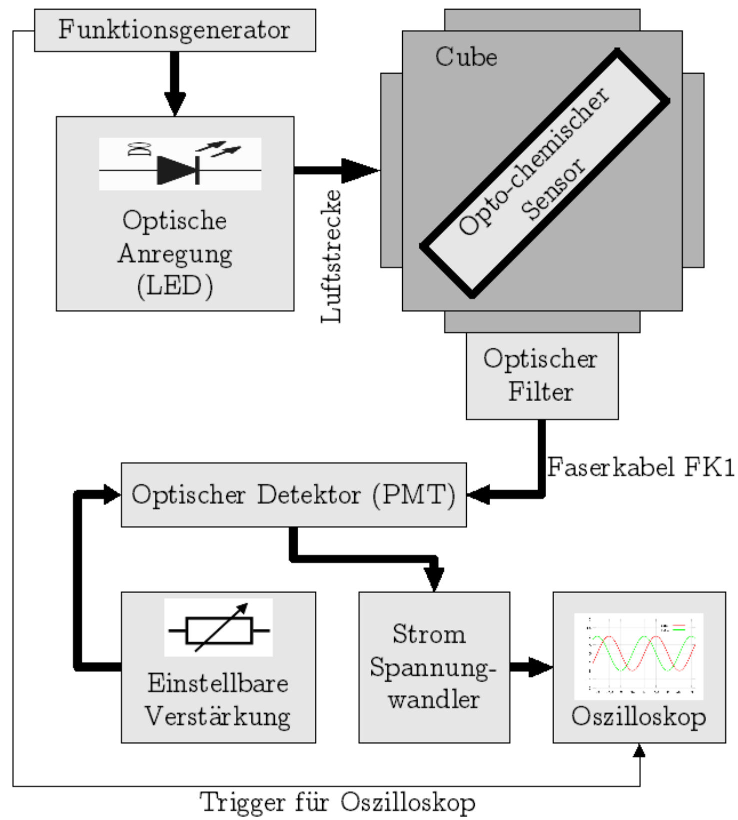



Abbildung 3.1: Messaufbau für direkte Abklingzeitmessung

hoch sein, da der verwendete optische Detektor ansonsten durch das emittierte Licht des opto-chemischen Sensors über seine Verwendungsgrenzen hinweg ausgesteuert wird und ein unbrauchbares Signal liefert. Ein Beispiel für das Signal eines übersteuerten optischen Detektors liefert Abbildung 3.3.
Die Frequenz des rechteckförmigen Signales, welches zur Anregung verwendet wird, muss klein genug sein, damit die Emission des opto-chemischen Sensors das stationäre Plateau erreicht.

| Sensor | Anregung [nm] | Emission Maximum [nm] | Led Led | Led Spektrum Maximum [nm] |
|---|---|---|---|---|
| PtTFPP | 392, 507, 540 | 650 | NSPB500 | 470 |
| PdTFPP | 407, 519, 552 | 670 | NSPB500 | 470 |
| RuDPP | 467, 441 | 597 | NSPB500 | 470 |

Tabelle 3.2: Benötigte Anregungsspektren für opto-chemische Sensoren

### 3.1.3 Cube

Es handelt sich hierbei um einen Würfel aus dem Bausteinsystem der Firma ®LINOS mit Öffnungen an allen 6 Seiten. In die Mitte des Würfels wird eine Halterung montiert, welche es erlaubt, einen auf einen Kunststoffträger gedruckten opto-chemischen Sensor aufzunehmen. Durch eine Seitenöffnung wird der Sensor mit dem Licht der LED bestrahlt. Auf einer anderen Seite werden ein optischer Filter sowie ein Anschluss für ein Faserkabel montiert. Eine rechtwinkelige Anordnung bietet sich an, da hier die LED nicht direkt in die für die Messung verwendete Austrittsöffnung scheint.

### 3.1.4 Optischer Filter

Der Einsatz eines optischen Filters ermöglicht die Messung dahingehend, das nur Spektralanteile, welche sich ausschliesslich dem emittierenden Sensor zuordnen lassen, für die Messung verwendet werden. Für einen PtTFPP Sensor und dessen Anregung wird dies in Abbildung 3.2 verdeutlicht. Die Abbildung zeigt das Spektrum der Anregung, des angeregten opto-chemischen Sensors sowie des gefilterten Ausgangssignales. Das Spektrum des angeregten opto-chemischen Sensors hat im Bereich ab 550nm starke Ausprägungen. Es soll

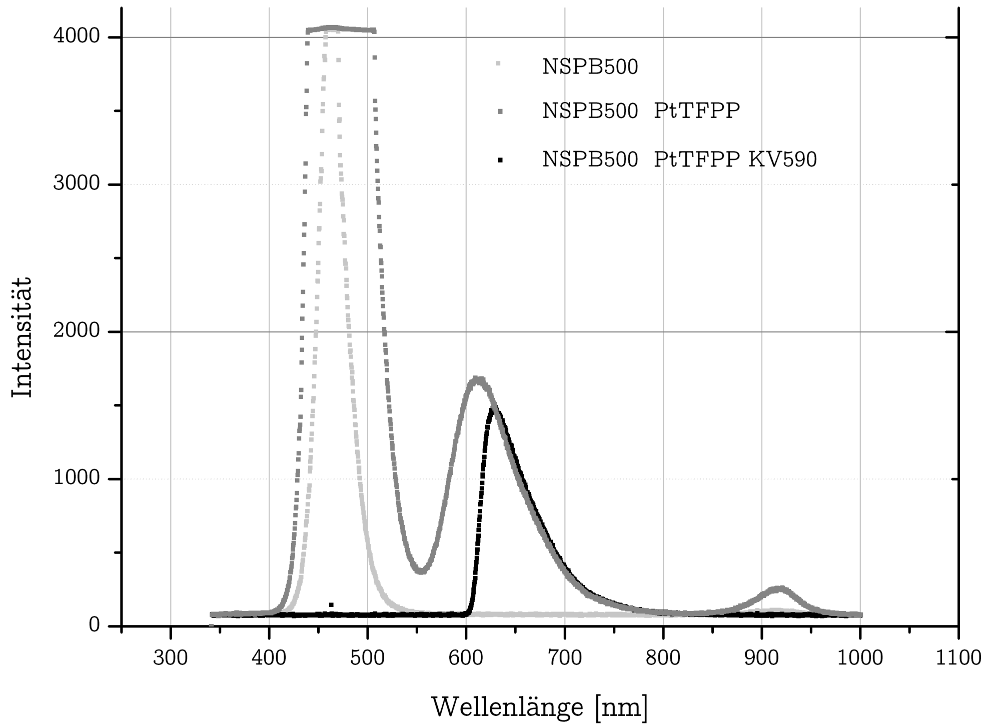



Abbildung 3.2: Alle Spektren LED NSPB500 und PtTFPP Sensor und Emissionsfilter RG610

Anmerkung: Unterschiedliche Integrationszeiten wurden hier verwendet. Der Peak bei ca. 920nm entstand durch Artefakte des Gitterspektrometers.

hier der Bereich, welcher sich der Anregung (LED) zuordnen lässt, ausgefiltert werden. Für PtTFPP wird hier ein RG610 Filter verwendet, welcher nur Wellenlängen über 610nm durchlässt und somit keine Anteile des Spektralbereiches der Anregung mehr in der Messung aufscheinen. Weitere verwendete Filter sind in Tabelle 3.3 aufgeführt.

| Sensor | Emissionsfilter |
|---|---|
| PdTFPP | RG610 |
| RUDPP | KV550 |
| PtTFPP | RG610 |

Tabelle 3.3: Emissionsfilter für opto-chemische Sensoren

### 3.1.5 Faserkabel FK1

Das verwendete Faserkabel hat einen Durchmesser von 1mm. Es hat einen Silica Kern und ist vom Typ: *0,48 NA Hard Polymer Clad Multimode Fiber*.

### 3.1.6 Optischer Detektor

Der Optische Detektor dient der Umwandlung von Licht in ein elektrisches Ausgangssignal. Er bietet die Möglichkeit einer einstellbaren Verstärkung, um je nach Signalstärke seinen Ausgabebereich voll ausnutzen zu können, bzw. um etwaige Übersteuerungen zu vermeiden. Abbildung 3.3 zeigt ein Triggersignal (hellgrau) und ein gemitteltes (Mittelwert 64) Signal (dunkelgrau) des Photodetektors bei korrekt eingestellter Verstärkung. Weiters ist ein Signal (schwarz) zu sehen, bei der der Photodetektor übersteuert wurde. Hier ist deutlich ersichtlich, das der Photodetektor bei zu hoher Verstärkung ein verfälschtes Signal liefert. Der Kurvenverlauf würde hier auf einen früheren Abfall des Signales schließen lassen, welcher aber nicht stattfindet.

Die Eckdaten des verwendeten optischen Detektors *PMT H6780-20* von Hamamatsu [9] sind in Tabelle 3.4 aufgeführt. Der Ausgang des verwendeten Detektors funktioniert wie eine Stromquelle, aus diesem Grund muss eine nachfolgende Wandlung in ein Spannungssignal durchgeführt werden.

| | |
|---:|---:|
| Wellenlängen Bereich | 300nm bis 900nm |
| Betriebsspannung | +11.5V bis +15.5V |
| Max. Eingangsstrom | 30mA |
| Max. Ausgangsstrom | 100µA |
| Max. Kontrollspannung | 1V |

Tabelle 3.4: Hamamatsu PMT H6780-20

### 3.1.7 Strom-Spannungswandler

Um das Stromsignal, welches der optische Detektor liefert, mit einem Oszilloskop messtechnisch erfassen zu können, muss eine Strom-Spannungswandlung durchgeführt werden. Eine einfache Schaltung dazu bedient sich lediglich eines Widerstandes und ist in Abbildung 3.4 ersichtlich [8].

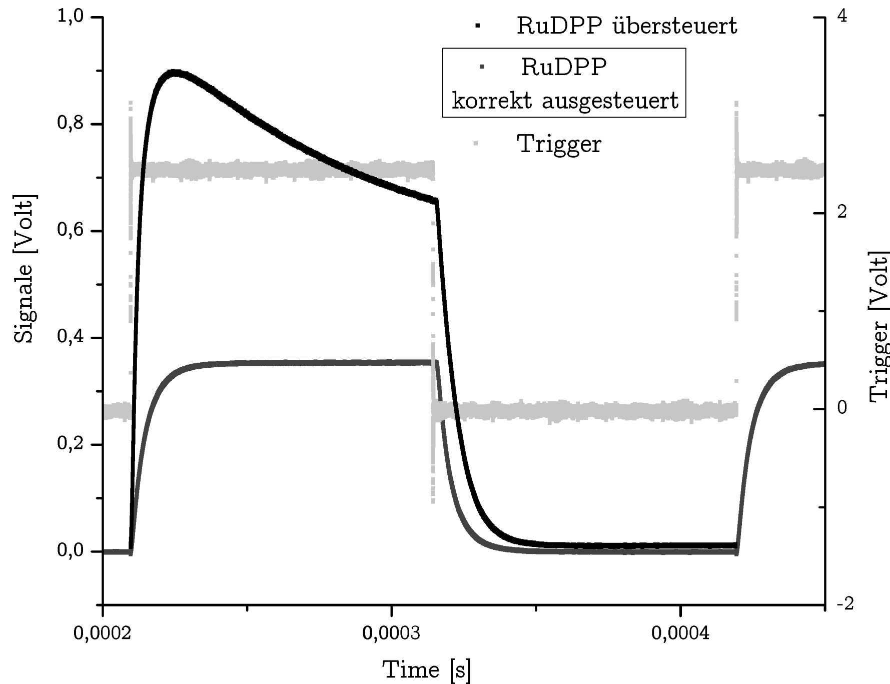



Abbildung 3.3: RuDPP mit übersteuertem und korrekt eingestelltem PMT Detektor

### 3.1.8 Oszilloskop

Zur Aufnahme der Signale wurde ein Oszilloskop des Typs Infiniium 54831D der Firma Agilent Technologies verwendet. Als Trigger diente das Signal des Funktionsgenerators. Bei der Messung wurde das Signal durch eingebaute Berechnungsfunktionen des digitalen Oszilloskops gemittelt (Mittelwert 64), um Rauschen zu unterdrücken.

### 3.1.9 Auswertung

Das Mathematikprogramm ®Origin wurde für die Auswertung der Messungen benutzt. Dabei wurde ein *Kurvenfitting* vorgenommen, welches die Kurve an eine vorgegebene Funktion anpasst. Abbildung 3.5 zeigt eine Messkurve,

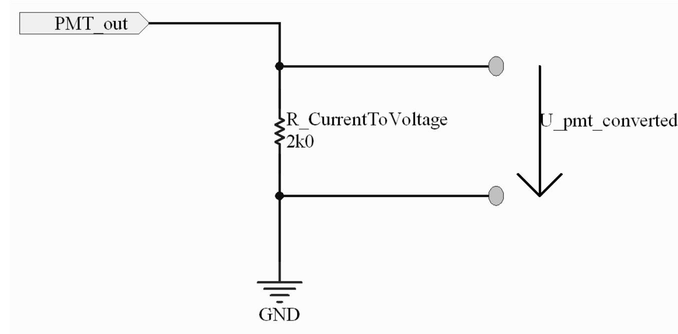



Abbildung 3.4: Strom-Spannungswandlung

sowie die dazugehörige *gefittete* Kurve. Die dazugehörigen Parameter sind in Tabelle 3.5 und die Ergebnisse des *Kurvenfittings* in Tabelle 3.6 ersichtlich. Als Grundformel, welche dem *Kurvenfitting* als Basis dient, wurde Gleichung (3.1.1) verwendet. Die daraus resultierenden Abklingzeiten dienen als Referenz für die Berechnung der optimalen Parameter in Kapitel 7.

$$y = A1 \cdot e^{(\frac{-x}{t1})} + y0 \tag{3.1.1}$$

| 200911226_PtTFPPinPSU_avg64_Air | |
|---:|---:|
| Anzahl der Punkte | 312100 |
| Freiheitsgrade | 312097 |
| Chi-Quadr Reduziert | 1.30E-06 |
| Fehler der Summe der Quadrate | 0.40511 |
| Kor. R-Quadrat | 0.99806 |
| Fit-Status | Erfolgreich(100) |

Tabelle 3.5: PtTFPP Air, *Kurvenfit* Statistik

## 3.2 Ergebnisse

Bei den für die Messung verwendeten opto-chemischen Sensoren handelt es sich um Sauerstoffsensoren. Bei ihnen ist eine Änderung des Sauerstoffgehalts durch eine Änderung der Abklingzeit messbar. Zur Messung mit PtTFPP und RuDPP wurden 2 Gase verwendet. Als erstes Gas wird Raumluft, mit

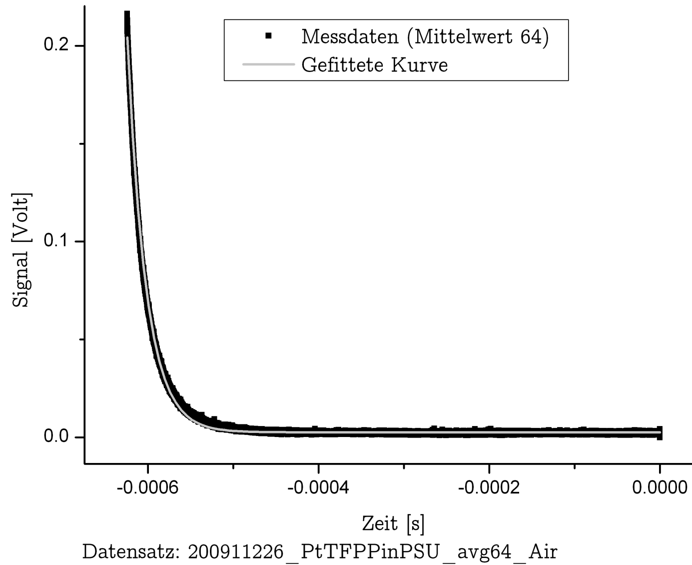



Abbildung 3.5: PtTFPP Air, *Kurvenfit* mit ®Origin

| 200911226_PtTFPPinPSU_avg64_Air | | |
|---|---|---|
| Formel | y = A1· exp(-x/t1) + y0 | |
| y0 | Wert | 0.00256 |
| | Standardfehler | 2.20E-06 |
| A1 | Wert | 9.46E-14 |
| | Standardfehler | 4.19E-16 |
| t1 | Wert | **2.20E-05** |
| | Standardfehler | 3.48E-09 |
| Statistik | Wert | 1.30E-06 |
| | Standardfehler | 0.99806 |

Tabelle 3.6: PtTFPP Air, *Kurvenfit* Ergebnisse Beispiel

einem Sauerstoffgehalt von 20,95%, verwendet. Das zweite Gas ist Stickstoff (N2) mit einem Sauerstoffgehalt von 0%. Der PdTFPP Sensor dient aufgrund seiner Empfindlichkeit Sauerstoff gegenüber als Spurensensor. Das heißt, er liefert schon bei kleinsten Mengen von Sauerstoff, große Änderungen der Ab-

klingzeit. Aus diesem Grund wurden für diese Messungen nur Stickstoff, Gas mit 0.1% Sauerstoff und Gas mit 5.0% Sauerstoff verwendet. Eine Messung mit dem Sauerstoffgehalt der Umgebungsluft wäre bei PdTFPP nicht sinnvoll. Die Abklingzeit hierbei wäre mit der Abklingzeit bei einem Gas mit 5.0% Sauerstoffgehalt nur minimal unterscheidbar.

| Sensor | Umgebung | Tau [s] |
|---:|---:|---:|
| PtTFPP | Air | 2.20E-05 |
| PtTFPP | N2 | 6.48E-05 |
| PdTFPP | 5% O2 | 1.54E-04 |
| PdTFPP | 0.1% O2 | 1.71E-04 |
| PdTFPP | N2 | 8.53E-04 |
| RuDPP | Air | 5.07E-06 |
| RuDPP | N2 | 6.10E-06 |

Tabelle 3.7: Abklingzeiten (Oszilloskop und Mathematik Software)

# 4 Hardwareplattform

Die Hardwareplattform soll alle benötigten Hardware-Komponenten in integrierter Form bereitstellen. Dieses Kapitel beinhaltet eine detaillierte Beschreibung aller Komponenten auf der Hardwareplattform und deren Beschaltung bzw. Zusammenschaltung. Es wird eine Übersicht über die per Hardware Jumper möglichen Einstellungen gegeben. Abschließend werden die Ergebnisse der Evaluierung präsentiert.

## 4.1 Komponenten für die Hardwareplattform

Folgende Kernkomponenten werden auf der Hardwareplattform integriert:

- Analog-digital-Wandler
- FPGA
  - Speicher
  - Recheneinheit
- Serielle Schnittstelle
- Trigger Ausgang für optische Anregung

Eine Übersicht über die internen und externen Komponenten gibt Abbildung 4.1.

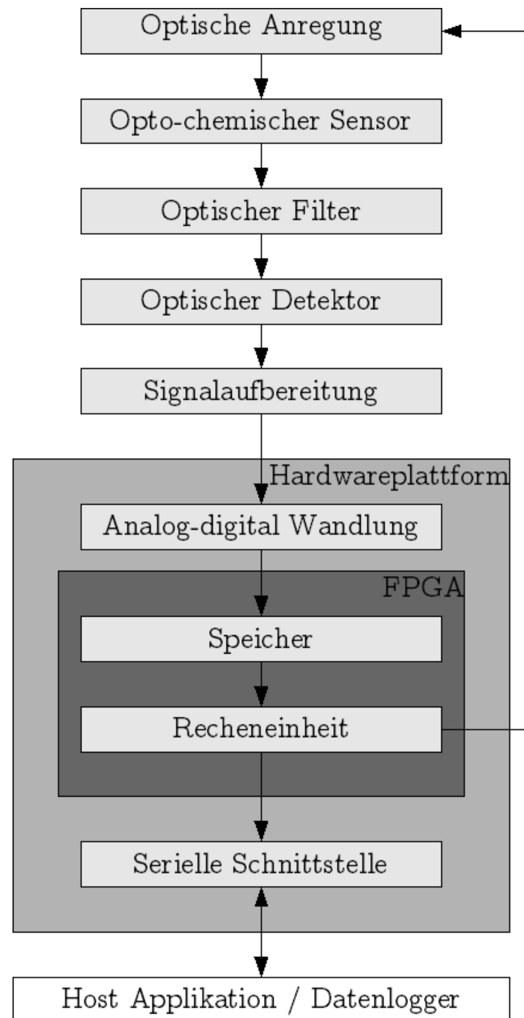



Abbildung 4.1: Hardware Komponenten

## 4.2 Zusätzliche externe Komponenten

### 4.2.1 Treiber für optische Anregung

Die optische Anregung, angepasst an den entsprechenden Wellenlängenbereich, welcher für den jeweiligen opto-chemischen Sensor notwendig ist, bekommt ein Triggersignal von der Hardwareplattform. Als Treiber für Leuchtmittel ist dieses Signal jedoch zu schwach und kann somit nur als Schaltsignal dienen. Die Verstärkung erfolgt durch eine Standard Transistor Schaltung [20]. Die Schaltung um LED's anzusteuern ist in Abbildung 4.2 ersichtlich [15]. Der Strom wird hier durch den 150 Ohm Widerstand begrenzt. Bei Verwendung einer NSPB500 LED ergeben sich folgende Messwerte an der LED:

- Strom $i_{LED} = 11.2$ mA
- Spannung $U_{LED} = 3.16$ V

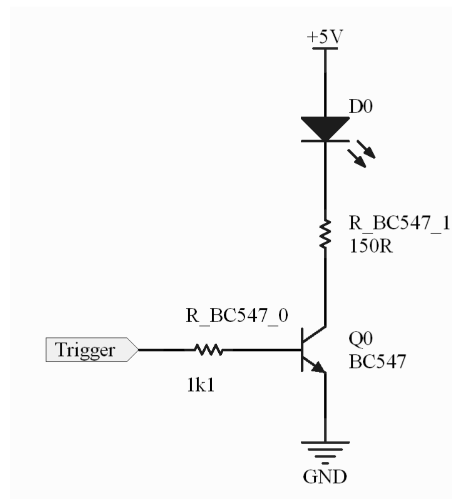



Abbildung 4.2: LED Treiber

## 4.3 Anforderungen

- Flexibilität
- Integriertes Gesamtsystem (Alle nötigen Komponenten integriert)
- Berechnung der Abklingzeit auf der Hardwareplattform mit einer Updaterate des Ergebnisses von ca. 1/sec.
- Messung von Abklingzeiten im Bereich von 1ms bis 1µs.

Die Anforderungen *Flexibilität und integriertes Gesamtsystem* werden hier durch ein System mit einem FPGA als zentrale Stelle erfüllt. Auf diesem kann ein Mikrocontroller (in einer Softcore Variante), Speicher sowie weitere Logik untergebracht werden. Ein Mikrocontroller kann hierbei auch die Aufgabe der *Berechnung der Abklingzeit* übernehmen.
Das System muss Abklingzeiten von 1ms bis zu 1µs messen können. Für die Erkennung von Abklingzeiten im Mikrosekunden-Bereich bedeutet das folgendes:
Der quasi stationäre Zustand für eine exponentiell abfallende Abklingzeit ist nach ca. 5 Tau erreicht. Dadurch muss ein Zeitfenster von mindestens 5 Tau, entspricht hier 5µs, erfasst werden.
Eine Abtastung mit 125MHz bei einer Fensterbreite von 1024 Datenpunkten kann einen Zeitraum von 8192ns speichern (siehe Gleichungen (4.3.1) bis (4.3.4)).

$$Fensterbreite = 1024 Datenpunkte \tag{4.3.1}$$

$$f_{Abtastung} = 125MHz \tag{4.3.2}$$

$$t_{Abtastintervall} = \frac{1}{f_{Abtastung}} = \frac{1}{125MHz} = 8ns \tag{4.3.3}$$

$$t_{Abtastfenster} = t_{Abtastintervall} \cdot Speichertiefe = 8ns \cdot 1024 = 8192ns \tag{4.3.4}$$

Bei einer Abklingzeit von Tau = -1µs (Formel (2.0.1)) und einer Abtastrate von 125MHz ist nach einer Messung Kurve aus Abbildung 4.3 im Speicher.

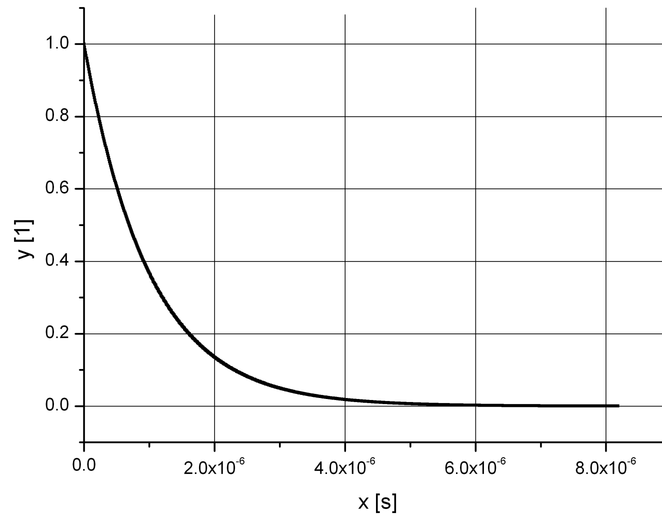



Abbildung 4.3: Exponentieller Abfall

## 4.4 SKØL-Board Hardwareplattform

### 4.4.1 FPGA

Der FPGA aus der ™Spartan 3 Reihe der Firma ®Xilinx ist ein konfigurierbarer Logikbaustein. Durch die Möglichkeit darauf vorhandene Zellen als Speicher zu nutzen, sowie einen Mikrocontroller in einer Softcore Variante darauf laufen zu lassen, ist er sehr gut für diese Anwendung geeignet. Die Merkmale des verwendeten Bausteins XC3S1500-4FG676C [22] sind in Tabelle 4.1 aufgeführt.

Die Konfiguration des FPGA wird je nach Jumper Einstellung nach anlegen der Versorgungsspannung von einem *Config PROM* automatisch geladen oder muss per JTAG Verbindung nach jedem Neustart neu programmiert werden. Die dafür notwendigen Jumper Einstellungen (I1_ConfigModeHdr) sind in Tabelle 4.2 beschrieben.

Einige zusätzliche IO's der FPGA Bank 6 und 7 lassen sich bezüglich ihrer Spannungspegel durch die Jumper I1_Bank6VCC und I1_Bank7VCC konfigurieren. Die dazu notwendigen Einstellungen sind in Tabelle 4.3 ersichtlich.

| | |
|---|---|
| 1.5 Millionen | System Gates |
| 29952 | Logic Cells |
| 208K | Distributed Ram Bits |
| 576K | Block Ram Bits |
| 487 | Maximum User I/O |
| 221 | Maximum Differential I/O Pairs |
| 326 MHz | System clock |
| 622 Mb/s | Data transfer rate per I/O |
| 17 | single-ended signal standards |
| 6 | differential signal standards |
| DDR | support |
| JTAG | compatible with IEEE 1149.1/1532 |

Tabelle 4.1: ®Xilinx ™Spartan 3 XC3S1500-4FG676C Merkmale

| I1_ConfigModeHdr | Programmierung |
|---|---|
| 12 | JTAG |
| 23 | Config PROM |

Tabelle 4.2: Jumper I1_ConfigModeHdr Konfigurationen

| Jumper | Spannung |
|---|---|
| 12 | 2.5 Volt |
| 23 | 3.3 Volt |

Tabelle 4.3: Jumper I1_Bank6VCC und I1_Bank7VCC Konfigurationen

### 4.4.2 Analog-digital-Wandler

Der ADC[1] AD9461 [7] von ®Analog Devices kann bis zu 130MSPS(Megasamples pro Sekunde), bei einer Datenbreite von 16 bit liefern. Seine digitalen Ausgänge können für den Betrieb im single-ended Modus oder für den Betrieb als differential-pair konfiguriert werden. Um Vorteile der differentiellen Übertragung nutzen zu können werden die digitalen Ausgänge des ADC für den

[1] Analog Digital Converter

Betrieb im differentiellen Modus konfiguriert. Weitere Merkmale des ADC sind

- Einstellbares Datenformat der digitalen Ausgänge (Offset Binary bzw. Zweierkomplement)
- Gepufferte analoge Eingänge
- Differentieller Eingang (2Volt peak-peak bis 4Volt peak-peak)
- Interne Referenz verfügbar
- Kontrollsignal für Out-of-range

Die relevanten Konfigurationseinstellungen sind im Schaltbild in Abbildung 4.4 ersichtlich.
Pin 3 (Output Mode) ist hierbei auf Masse. Dadurch werden die digitalen Ausgänge des ADC im differentiellen Modus betrieben.
Pin 7 (SENSE) stellt den ADC durch die Verbindung auf Masse darauf ein, die interne Referenzspannung zu verwenden.
Jumper *I4_ADC_SFDR* bietet die Möglichkeit eine Frequenzabhängige Rauschunterdrückung einzustellen (siehe Tabelle 4.4).

Jumper *I4_ADC_DFS* dient der Auswahl des digitalen Datenformats für den Ausgang (siehe Tabelle 4.5).

| Jumper | Signalbandbreite |
|---:|---:|
| 12 | $<$ 40 MHz oder $>$ 215MHz (default) |
| 23 | $>$ 40 MHz oder $<$ 215MHz |

Tabelle 4.4: AD9461 Jumper Einstellung I4_ADC_SFDR

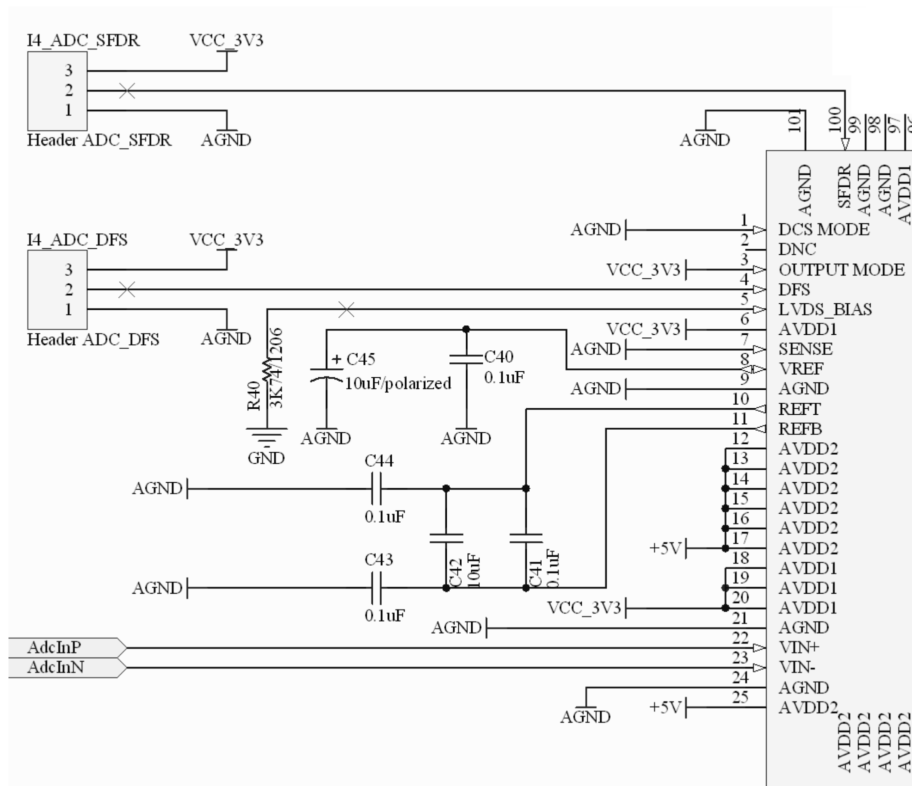



Abbildung 4.4: AD9461 Schaltplan Auszug

| Jumper | Datenformat |
|---|---|
| 12 | offset Binary (default) |
| 23 | twos Complement |

Tabelle 4.5: AD9461 Jumper Einstellung I4_ADC_DFS

### 4.4.3 Differentieller Eingangstreiber für ADC

Der ADC (siehe 4.4.2) benötigt ein differentielles Signal, der optische Detektor (siehe 3.1.6) liefert jedoch (nach der Strom-spannungs Wandlung) ein single-ended Signal. Aus diesem Grund muss eine single-ended zu differential Umwandlung durchgeführt werden.
Für diese Umwandlung wird der Baustein AD8138 [6] von ®Analog Devices verwendet. Dieser Baustein bietet eine konfigurierbare Verstärkung, welche verwendet wird, um den Eingangsbereich des ADC bestmöglich auszunutzen. Die Beschaltung des Bausteins ist in Abbildung 4.5 ersichtlich. Das Signal des Photodetektors wird in J3_SingleEndedInput eingespeist. Die beiden Signale AD8138Out_P und AD8138Out_N sind das differentielle Signalpaar für den ADC.

**Anpassung des Eingangswiderstandes an die Signalquelle** erfolgt durch die beiden Widerstände *R3_0* und *R3_6*. Für eine Signalquelle mit Innenwiderstand von 2kOhm ergeben sich folgende Werte, wie in den Gleichungen (4.4.1) bis (4.4.4) beschrieben. Tabelle 4.6 gibt eine Zusammenfassung für Quellen mit 50Ohm und 2kOhm Innenwiderstand.

$$R_{Source} = 2kOhm \tag{4.4.1}$$

$$R_{3_0} = R_{Source} = 2kOhm \tag{4.4.2}$$

$$R_{Source} \| R_{3_0} = 1kOhm \tag{4.4.3}$$

$$R_{3_6} = 499Ohm + (R_{Source} \| R_{3_0}) = 499Ohm + 1kOhm = 1499Ohm \tag{4.4.4}$$

**Einstellung der Verstärkung** erfolgt durch die Jumper *J3_PreAmpHdr0* und *J3_PreAmpHdr1*. Eine Zusammenfassung für die Einstellungen gibt Tabelle 4.7. Abbildung 4.6 zeigt die frequenzabhängige Verstärkungskurve des

| Quelle | R3_0 | R3_6 |
|---|---|---|
| 50Ohm | 49Ohm | 523Ohm |
| 2kOhm | 2kOhm | 1499Ohm |

Tabelle 4.6: Widerstandsanpassung an Signalquelle für AD8138

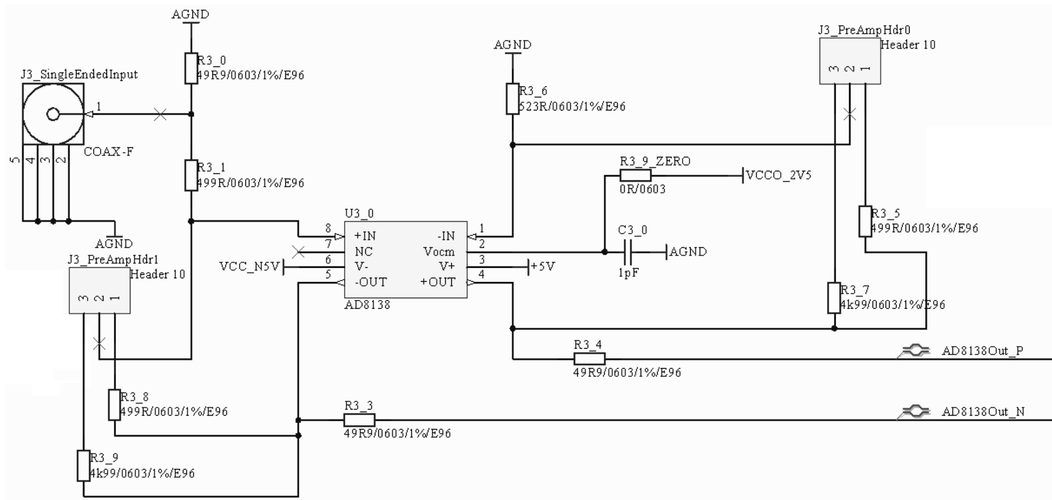



Abbildung 4.5: AD8138 Schaltplan

AD8138. Daraus resultierend ist zu beachten, daß die Verstärkung (Betrieb mit Verstärkung = 10) nur bis circa 10MHz konstant bleibt, jedoch im oberen Frequenzbereich leicht einbricht.

| | | Bereich 1 | Bereich 2 |
|---|---|---|---|
| Verstärkung | | 1 | 10 |
| Jumper Verbindung (beide) | | 12 | 23 |
| unteres Ende | [mV] | -4000 | -400 |
| oberes Ende | [mV] | 4000 | +400 |
| gesamter Bereich | [mV] | 8000 | 800 |
| Auflösung | [µV/bit] | 122.070 | 12.207 |
| Testsignal Amplitude 90% | [mV] | 7200 | 720 |

Tabelle 4.7: Analog Digital Wandlung, Verstärkung, Auflösung

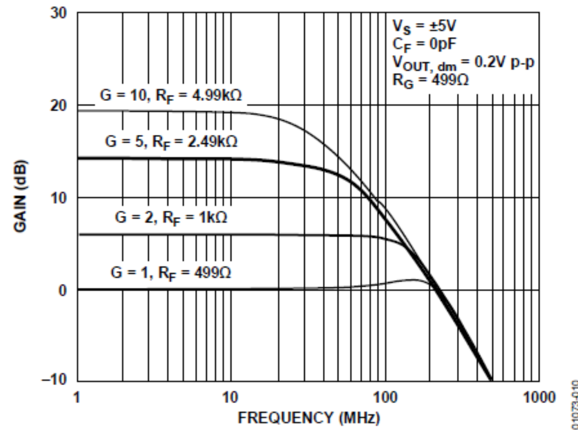



Abbildung 4.6: AD8138 Frequenzabhängige Verstärkungskurve. ©Analog Devices

### 4.4.4 Peripherie, weitere Komponenten

**Als RS232 Kommunikationstreiber** wird der Baustein *MAX3232ECAE* der Firma ®MAXIM verwendet. Die Schaltung ist in Abbildung 4.7 ersichtlich.

**Spannungsversorgung** Tabelle 4.8 beinhaltet eine Übersicht über die verwendeten Bauteile, sowie der jeweils versorgten Bausteine. Die Hardwareplattform wird hierbei mit +VCC_Board = +8 Volt und -VCC_Board = -8 Volt versorgt.

| Baustein | Spannung | zu versorgende Bauteile |
|---:|---:|---:|
| LM1086IS-5,0 | 5 Volt | AD9431, AD8138 |
| LM1086IS-3,3 | 3.3 Volt | AD9431, MAX3232, FPGA, Config PROM |
| LM1086CS-2,5 | 2.5 Volt | FPGA |
| LM1086IS-1,8 | 1.8 Volt | Config PROM |
| TPS73701DCQ | 1.2 Volt | FPGA |
| LT1964 -5V | -5 Volt | AD8138 |

Tabelle 4.8: Übersicht Spannungsversorgung

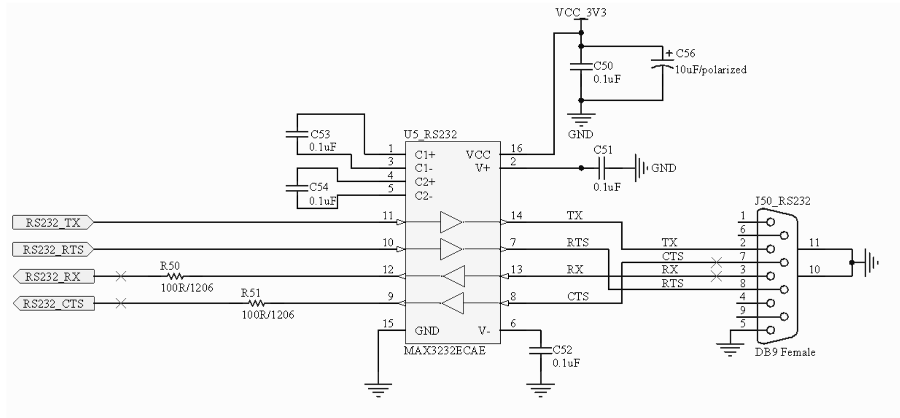



Abbildung 4.7: MAX3232ECAE Schaltung

**Die Speicherung der FPGA Konfiguration** muss extern erfolgen, da der verwendete FPGA (siehe 4.4.1) hierfür keine Möglichkeit bietet. Dafür wird ein eigener Baustein, der *Config PROM* verwendet. Das verwendete Bauteil ™XCF08P, der Firma ®Xilinx, bietet mit seinen 8Mb internen Speicher[23] ausreichend Platz für jede mögliche Konfiguration des FPGA's. Die Programmierung erfolgt per JTAG. In Abbildung 4.4.4 ist die notwendige Beschaltung ersichtlich.
Die Konfiguration des FPGA erfolgt im *Master Serial Mode*. Hierbei generiert der FPGA das Taktsignal, welches für die Konfiguration benötigt wird. Abbildung 4.9 gibt eine Übersicht über die Signale, die hierfür verwendet werden.

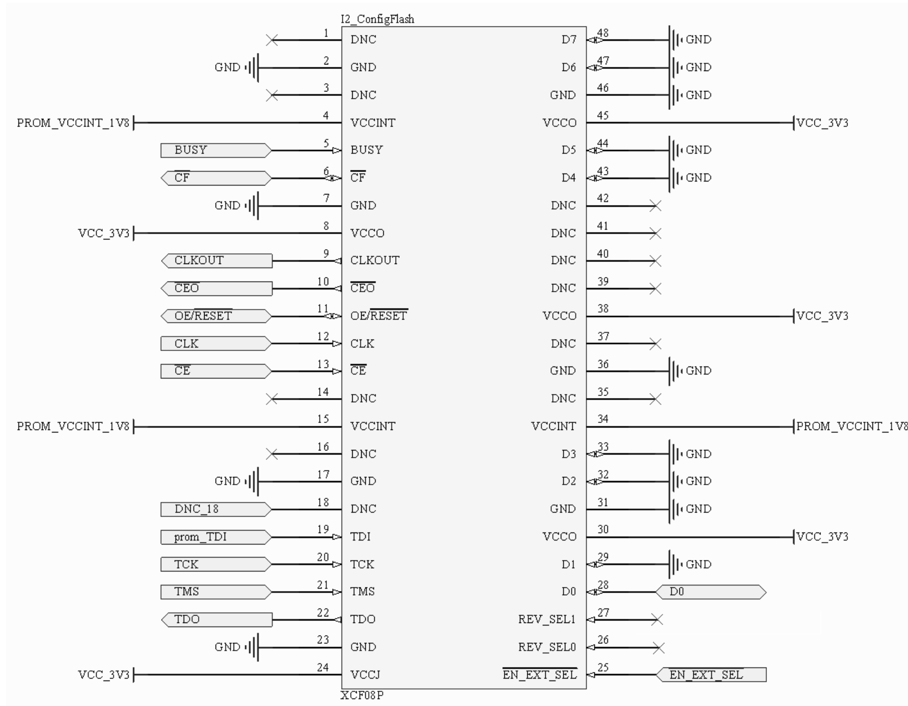



Abbildung 4.8: XCF08P Schaltung

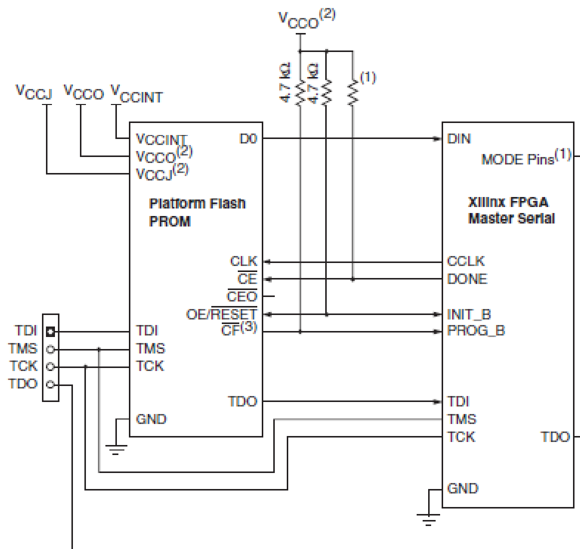



Abbildung 4.9: Programmierung des FPGA mit Config PROM mittels *Master Serial Mode*. ©Xilinx

### 4.4.5 Board Übersicht, Jumper, IO's

Eine Übersicht über alle Jumper, IO's usw. geben Abbildung 4.10 und Tabelle 4.9.

| | Name | Beschreibung |
|---|---|---|
| *1 | I4_ADC_DFS | ADC Datenformat (siehe 4.4.2) |
| *2 | I4_ADC_SFDR | ADC Rauschunterdrückung (siehe 4.4.2) |
| *3 | Analog Ground Testpoint | Testpunkt |
| *4 | J3_SingleEndedInput | Eingang für Photodetektor |
| *5 | J3_AdcInP | Direkter Eingang zu ADC |
| *6 | J3_AdcInN | Direkter Eingang zu ADC |
| ~1 | J3_AdcInNHdr | Direkter Eingang zu ADC |
| ~2 | J3_AdcInPHdr | Direkter Eingang zu ADC |
| ~3 | J3_PreAmpHdr0 | Verstärkungseinstellung (siehe 4.4.3) |
| ~4 | J3_PreAmpHdr1 | Verstärkungseinstellung (siehe 4.4.3) |
| #1 | J50_RS232 | Serielles Port |
| #2 | J5_Tag | JTAG |
| #3 | J6_clk0 | Zusätzlicher Clock Eingang |
| #4 | J6_clk1 | Zusätzlicher Clock Eingang |
| #5 | User Buttons | Diverse Taster |
| #6 | µC RESET Button | Reset für den Mikrocontroller |
| #7 | User Led's | Diverse Led's |
| +1 | I1_ConfigModeHdr | FPGA Konfiguration (siehe 4.4.1) |
| +2 | I1_BankxxVCC | FPGA Konfiguration (siehe 4.4.1) |
| +3 | J6_HDR_Trigger | Trigger Ausgang für optische Anregung |

Tabelle 4.9: Beschreibung zu Abbildung 4.10

## 4.5 Evaluierung

Tests zur Evaluierung der Hardwareplattform werden nach IEEE1057 [16] (auch [17]) durchgeführt. Es werden das Rauschverhalten (SNR, SINAD), sowie die effektive Bitanzahl (ENOB) der Plattform bestimmt.

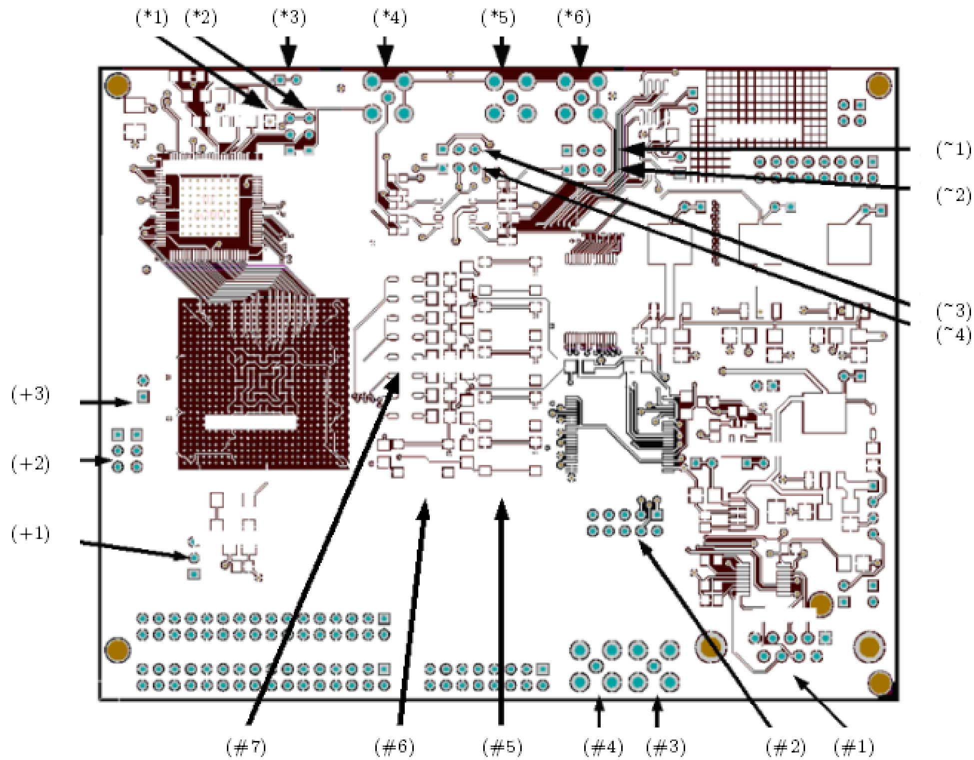



Abbildung 4.10: Übersicht der Hardwareplattform Platine (Blick von oben)

### 4.5.1 Testaufbau

Als Testsignal wird in [16] ein sinusförmiges Signal vorgeschlagen. Es soll dabei eine definierte Frequenz besitzen und mit seiner Amplitude bei 90% und einmal bei 100% des möglichen Eingangsbereichs liegen. Ausserdem sollen je nach Abtastrate und Speichertiefe mindestens 5 Perioden gespeichert werden. Bei einer Abtastrate von 125MSPS und einer Speichertiefe von 1024 Samples erhält man eine Speicherdauer von 8192ns (siehe Gleichungen (4.3.1) bis (4.3.4)). Um mindestens 5 Sinus Perioden aufzunehmen, benötigt man eine Frequenz von 650kHz am Funktionsgenerator (Agilent 81150A). Die Berechnung ist in den Gleichungen (4.5.1) und (4.5.2) ersichtlich.

$$t_{Sinus} = t_{Abtastfenster}/5 = 8192ns/5 = 1638.4ns \qquad (4.5.1)$$

$$f_{SinusMinimum} = 1/t_{Sinus} = 1/1638.4ns = 610.3515625kHz \qquad (4.5.2)$$

Das zu testende System wird hierbei direkt mit dem Testsignal beaufschlagt.

### 4.5.2 Rohdaten Aufnahme

Als Basis für die Berechnung der Parameter, dienen aufgenommene Kurven des zuvor beschriebenen Sinus Signales bei verschiedenen Amplituden. Diese aufgenommenen Messkurven werden mit einem Mathematik Programm (Origin) *gefittet*. Als Basis Gleichung für den *Fit* wird Gleichung (4.5.3) verwendet. Es werden einfache Kurven, sowie gemittelte (Mittelwert aus 64) Kurven aufgenommen. Ein *Kurvenfit* ist in Abbildung 4.11 zu sehen.

$$f(x) = y_0 + A \cdot sin(pi \cdot \frac{x - x_c}{w}) \tag{4.5.3}$$

$$\begin{aligned} y_0 &= \text{Offset} \\ x_c &= \text{Verschiebung auf x Achse} \\ w &= \text{Periodendauer} \end{aligned}$$

### 4.5.3 Berechnung NAD, SINAD, ENOB

**NAD** (Noise and Distortion) gibt Auskunft über die Summe der Auswirkungen des Rauschens und der totalen harmonischen Verzerrung (THD ... total harmonic distortion) [16]. Zur Berechnung des NAD benötigt man die Messkurve und die *gefittete* Kurve. Beim *Kurvenfitting* wird die Frequenz fix vorgegeben. Gleichung (4.5.4) beschreibt die Berechnung.

$$NAD = \left[ \frac{1}{M} \sum_{n=1}^{M} (x[n] - x'[n])^2 \right]^{\frac{1}{2}} \tag{4.5.4}$$

$$\begin{aligned} x[n] &= \text{Messkurve} \\ x'[n] &= \textit{gefittete} \text{ Kurve} \\ M &= \text{Anzahl Messpunkte} \end{aligned}$$

**SINAD** (Ratio of signal to noise and distortion) ist das Verhältnis des RMS Signalpegels zu NAD. Berechnung erfolgt mit Gleichung (4.5.5).

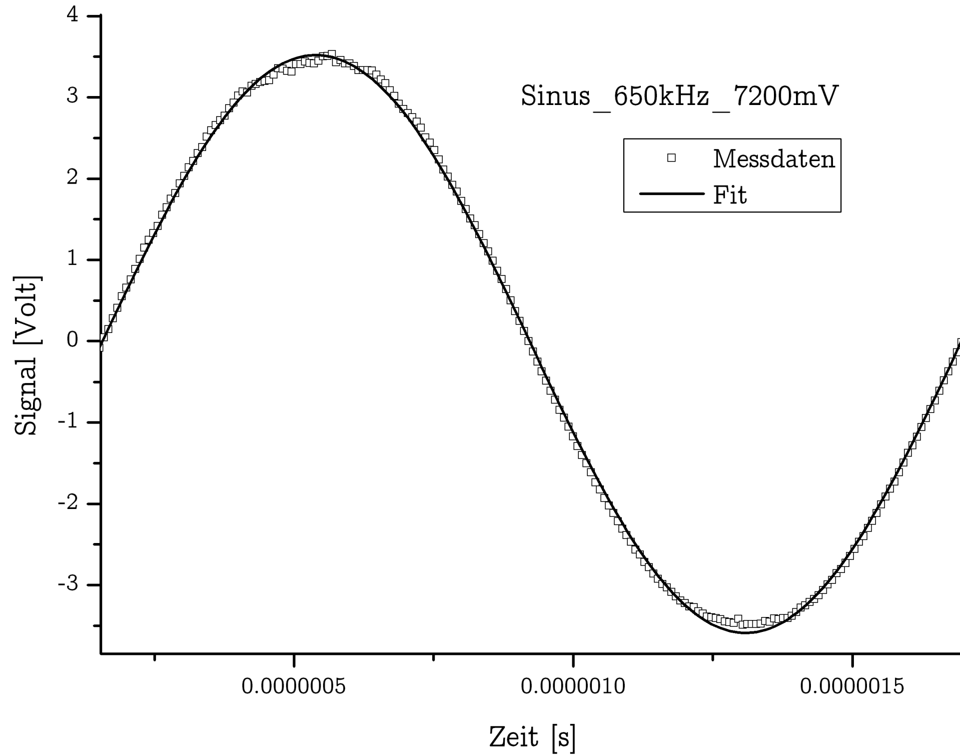



Abbildung 4.11: Sinus *gefitted*

$$SINAD = \frac{A_{rms}}{NAD} \tag{4.5.5}$$

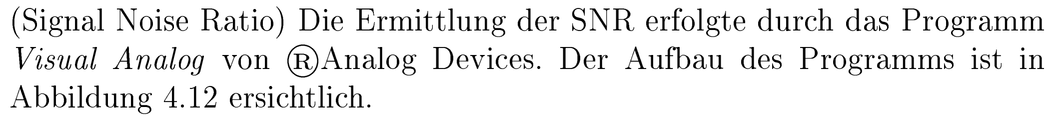
$A_{rms}$ =RMS Signal (Peak Amplitude)
des *gefitteten* Signals dividiert durch $\sqrt{2}$

**ENOB** (effective number of bits) Die Berechnung erfolgte nach Gleichung (4.5.6).

$$ENOB = log_2\left(\frac{FSR}{NAD \cdot \sqrt{12}}\right) \tag{4.5.6}$$

$FSR =$ (Full Scale Range) -

### 4.5.4 Ermittlung SNR



(Signal Noise Ratio) Die Ermittlung der SNR erfolgte durch das Programm *Visual Analog* von ®Analog Devices. Der Aufbau des Programms ist in Abbildung 4.12 ersichtlich.

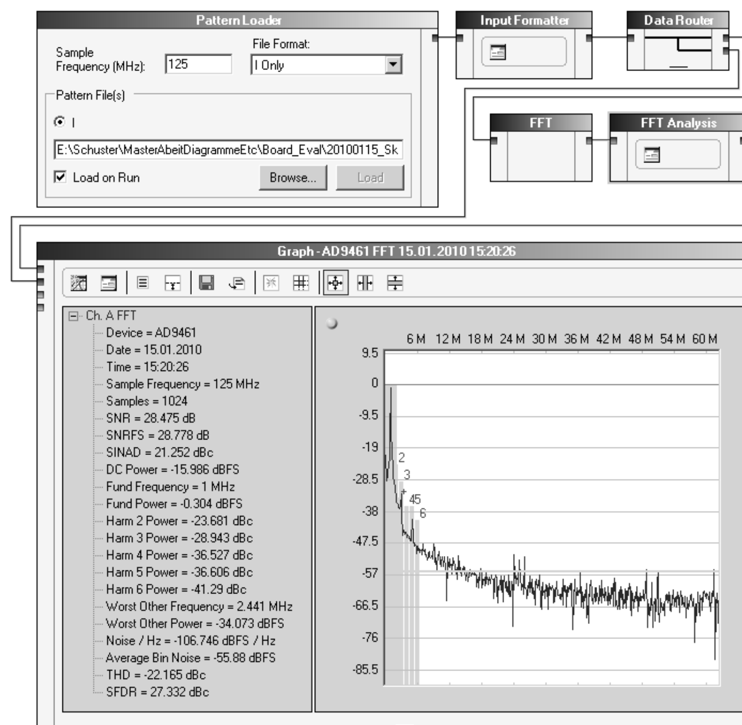



Abbildung 4.12: *Visual Analog* Konfiguration

### 4.5.5 Auswertung

Die berechneten Werte für Daten ohne Mittelwertbildung sind in Tabelle 4.10 ersichtlich, in Tabelle 4.11 sieht man die Auswertung für die gemittelten (Mittelwert 64) Daten. Hierbei ist deutlich zu sehen das sich der Signal Rausch Abstand bei Mittelung deutlich von rund 20dB auf rund 40dB vergrößert und somit eine Verbesserung der Signalqualität erreicht wurde.

| Verstärkung | 1 | 1 | 10 | 10 |
|---:|---:|---:|---:|---:|
| Sinus Amplitude [V] | 3.6 | 4 | 0.36 | 0.4 |
| *gefittete* Parameter: | | | | |
| $y_0$ | -0.03341 | -0.0336 | 0.01201 | 0.01128 |
| $y_0$ Standardabweichung | 0.00173 | 0.00289 | 0.000207 | 0.000275 |
| A | 3.55395 | 3.87021 | 0.36818 | 0.38334 |
| A Standardabweichung | 0.00246 | 0.00413 | 0.000294 | 0.000388 |
| NAD | 0.055125 | 0.092164 | 0.006596 | 0.008752 |
| SINAD | 45.159 | 29.435 | 40.756 | 31.881 |
| ENOB | 4.389 | 3.647 | 4.130 | 3.722 |
| SNR [dB] | 20.969 | 20.029 | 21.858 | 23.363 |
| Fehler: | | | | |
| Verstärkung [V] | -0.04605 | -0.12979 | 0.00818 | -0.01666 |
| Offset [V] | -0.03341 | -0.0336 | 0.01201 | 0.01128 |

Tabelle 4.10: Evaluierung Übersicht (Mittelwert 1)

| Verstärkung | 1 | 1 | 10 | 10 |
|---:|---:|---:|---:|---:|
| Sinus Amplitude [V] | 3.6 | 4 | 0.36 | 0.4 |
| *gefittete* Parameter: | | | | |
| $y_0$ | -0.0208 | -0.01952 | 0.00509 | 0.00495 |
| $y_0$ Standardabweichung | 0.00207 | 0.0027 | 0.000193 | 0.000256 |
| A | 3.73576 | 3.89309 | 0.37069 | 0.38606 |
| A Standardabweichung | 0.00292 | 0.00381 | 0.000273 | 0.000361 |
| NAD | 0.064034 | 0.083427 | 0.005968 | 0.007914 |
| SINAD | 41.02318 | 32.83121 | 44.52761 | 34.93849 |
| ENOB | 5.172543 | 4.790859 | 5.274238 | 4.867055 |
| SNR [dB] | 40.683 | 40.531 | 40.247 | 40.681 |
| Fehler: | | | | |
| Verstärkung [V] | 0.13576 | -0.10691 | 0.01069 | -0.01394 |
| Offset [V] | -0.0208 | -0.01952 | 0.00509 | 0.00495 |

Tabelle 4.11: Evaluierung Übersicht (Mittelwert 64)

# 5 Konfiguration der Hardwareplattform

Die für die im vorigen Kapitel beschriebene Hardwareplattform benötigte Software, Konfiguration des FPGA's sowie die Abläufe im Mikrocontroller werden hier beschrieben. Dies beinhaltet das Zusammenspiel der Komponenten durch gemeinsam benutzte Speicherbereiche, Interrupts und Trigger-Signale.

## 5.1 FPGA Konfiguration

Eine Übersicht der verwendeten Module im FPGA gibt Abbildung 5.1.

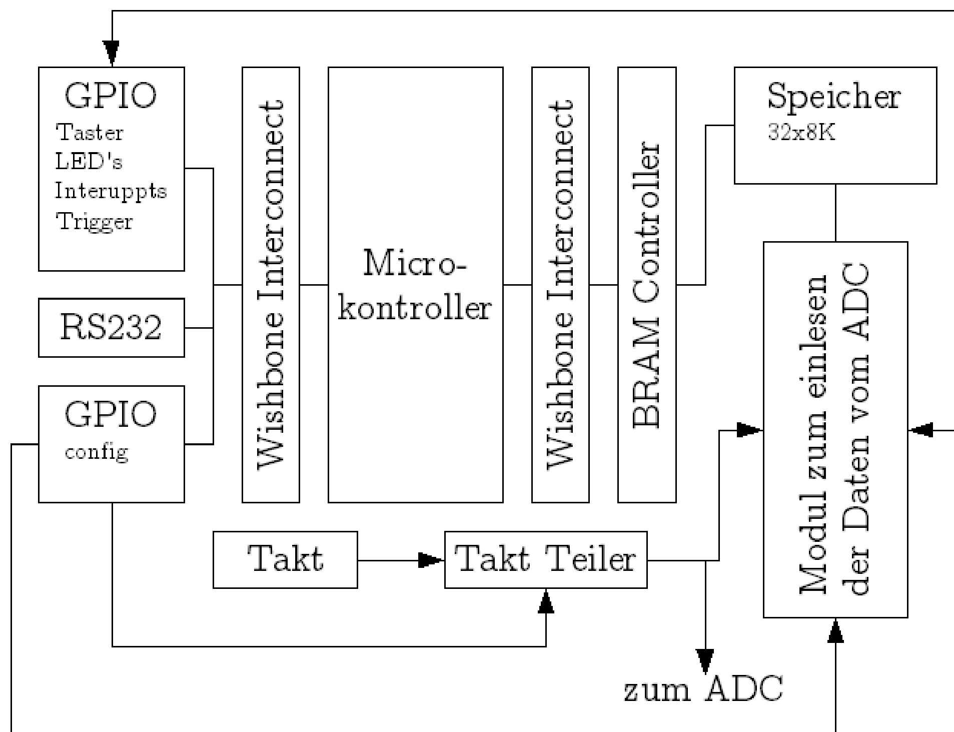



Abbildung 5.1: FPGA Module

Der Mikrocontroller bietet zur Ansteuerung von Peripherie und Speicher bereits 2 getrennte Anschlüsse. Abbildung 5.2 gibt einen detaillierteren Überblick über die Komponenten auf der *Peripherie-Seite*, und Abbildung 5.3 auf

der *Speicher-Seite* des Mikrocontrollers.

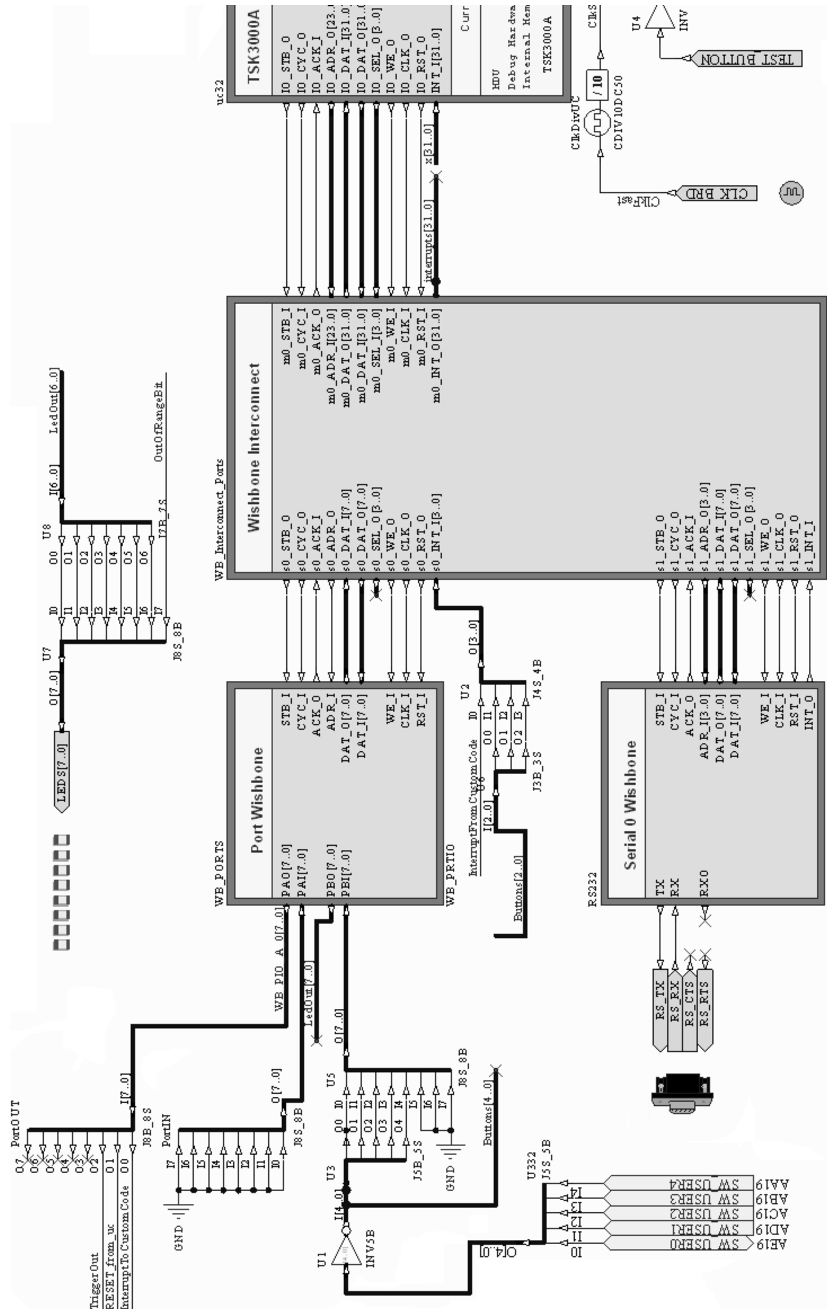



Abbildung 5.2: FPGA Komponenten (Mikrocontroller Peripherie Seite)

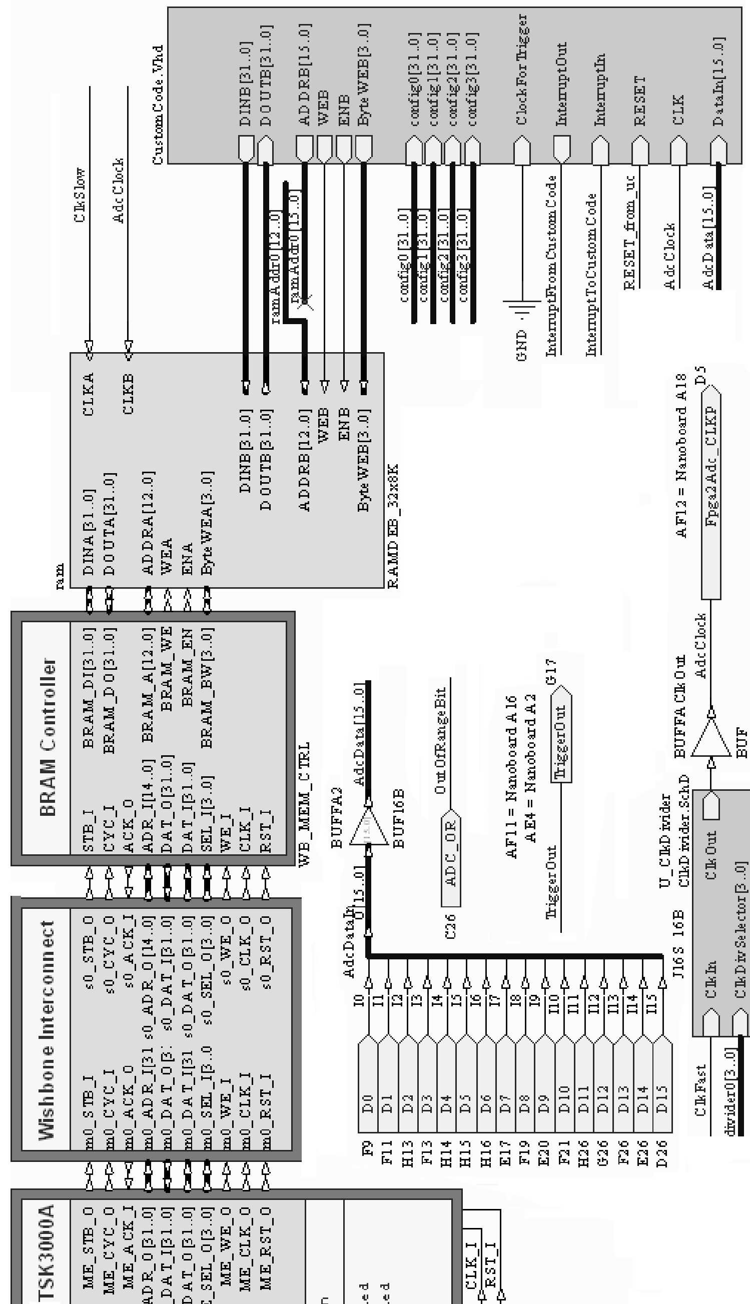



Abbildung 5.3: FPGA Komponenten (Mikrocontroller Speicher Seite)

## 5.2 Softcore Mikrocontroller

Der verwendete Mikrocontroller TSK3000A von ®Altium ist ein 32-bit Softcore Prozessor. Softcore Prozessoren existieren nicht als fertige Hardware, sie laufen ihrerseites wieder als Software auf einer anderen Plattform. Die Programmierung erfolgt in der Programmiersprache C. Dadurch ist eine Portierung der Software auf andere (auch reale Prozessoren wie zum Beispiel ARM, PowerPC, DSP56xxx oder Nios II) möglich. In diesem Fall läuft der TSK3000A auf einem FPGA. In Tabelle 5.1 sind die Eigenschaften des TSK3000A aufgeführt [3].

| | |
|---:|---:|
| Prozessor Typ | RISC |
| Instruktionsbreite | 32 bit |
| Wishbone | kompatibel |
| Pipeline | 5 stufig |
| Interrupts | 32 |
| Adressraum | 4GByte |
| Verfügbarkeit | Softcore |

Tabelle 5.1: TSK3000A Eigenschaften

Die Programmierung des TSK3000A ist in Kapitel 6 beschrieben. Zu den Hauptaufgaben des Mikrocontrollers gehören die Kommunikation mit einer etwaigen Host-Software oder einer anderen Auswerteeinheit, das Starten des VHDL Moduls, welches für das Einlesen der ADC-Daten zuständig ist sowie die Auswertung der gespeicherten Daten.

## 5.3 Interrupts, Trigger

Folgende Interrupts steuern den Mikrocontroller:

- *InterruptFromCustomCode*
- *RS232* Daten empfangen

Der Interrupt *InterruptFromCustomCode* signalisiert dem Mikrocontroller, das die gewünschten Daten in den Speicher eingelesen wurden (siehe 5.5). Interrupt *RS232* wird bei jedem neuen über die serielle Verbindung empfangenen Datenpaket ausgelöst.

**Trigger** Folgende Trigger-Signale finden im Design Verwendung:

- *TriggerOut:* Startet eine externe Leuchtquelle zur Anregung des Sensors
- *InterruptToCustomCode:* Durch ihn beginnt das *CustomCode-VHDL-Modul* mit dem Einlesen der ADC Daten.

## 5.4 Speicher

Als Speicher werden die intern im FPGA vorhandenen Block-RAM Elemente verwendet. In Summe sind 64KB[2] auf dem verwendeten FPGA verfügbar. Für den Mikrocontroller werden 32KB reserviert, die verbleibenden 32KB werden als Datenspeicher für ADC-Daten genutzt.

**Speicheraufteilung Datenspeicher** Der Datenspeicher mit 32KB ist wie in Tabelle 5.2 ersichtlich aufgeteilt. Durch diese Gliederung kann eine Mittelwertbildung aus 64 einzelnen Datenaufzeichnungen erfolgen. Diese Mittelwertbildung dient der Reduzierung des Rauschens [10].

Der Ablauf der Datenspeicherung sieht dabei wie in Abbildung 5.4 ersichtlich aus. Hierbei werden zuerst 8 Datenreihen aufgenommen, diese gemittelt und in den ersten *AverageBlock* Speicherbereich geschrieben. Eine solche Datenreihe, welche den Mittelwert aus 8 einzelnen Datenreihen bildet (8 Messkurven im *ScratchBlock*), wird nun *avg8* genannt. Dieser Vorgang wird 8 mal wiederholt, bis man 8 *avg8* Datenreihen im *AverageBlock* Speicherbereich hat. Nun werden diese 8 *avg8* Datenreihen gemittelt, wodurch man schlussendlich eine Mittelung von 64 Datenreihen berechnet hat.

## 5.5 Einlesen der Daten vom ADC

Der ADC liefert Daten (eingesetzter Oszillator auf der Hardwareplattform von 125MHz) mit einer Samplerate von bis zu 125MSPS. Da der Mikrocontroller zur direkten Verarbeitung von Daten mit dieser Geschwindigkeit nicht fähig ist, werden die Daten mit einem VHDL Modul in den Datenspeicher geladen. Dieser Datenspeicher bietet 2 Schnittstellen, welche unterschiedlich getaktet werden können (*Dual Ported Block Ram*). Die eine Seite ist über einen Speichercontroller und einen daran anschließenden Wishbone Controller mit dem Mikrocontroller verbunden, die andere Seite mit dem VHDL

[2] KB steht hier für 1024 Byte

| Bereich | Start Adresse (32 bit Adressraum) | Block (je 1024x16bit) |
|---|---|---|
| ScratchBlock | 0x0000 | 0 |
| | 0x0200 | 1 |
| | 0x0400 | 2 |
| | 0x0600 | 3 |
| | 0x0800 | 4 |
| | 0x0A00 | 5 |
| | 0x0C00 | 6 |
| | 0x0E00 | 7 |
| AverageBlock | 0x1000 | 8 |
| | 0x1200 | 9 |
| | 0x1400 | 10 |
| | 0x1600 | 11 |
| | 0x1800 | 12 |
| | 0x1A00 | 13 |
| | 0x1C00 | 14 |
| | 0x1E00 | 15 |

Tabelle 5.2: Speicheraufteilung Datenspeicher

Modul (*CustomCode*). Eine Übersicht dazu liefert Abbildung 5.3.
Das *CustomCode* Modul erhält folgende Informationen (über *config0,1,2,3* Signale) :

- **Startadresse:** Erste Speicheradresse
- **Endadresse:** Letzte Speicheradresse
- **Steuerkommando** Steuert die Funktionalität des Moduls

Die interne Funktion des Moduls ist in Abbildung 5.5 als Diagramm dargestellt.

## 5.6 Clock Divider

Um die verschiedenen möglichen Abklingzeiten im Bereich von 1ms bis 1µs messen zu können, werden verschiedene Auflösungen der Abtastung benötigt,

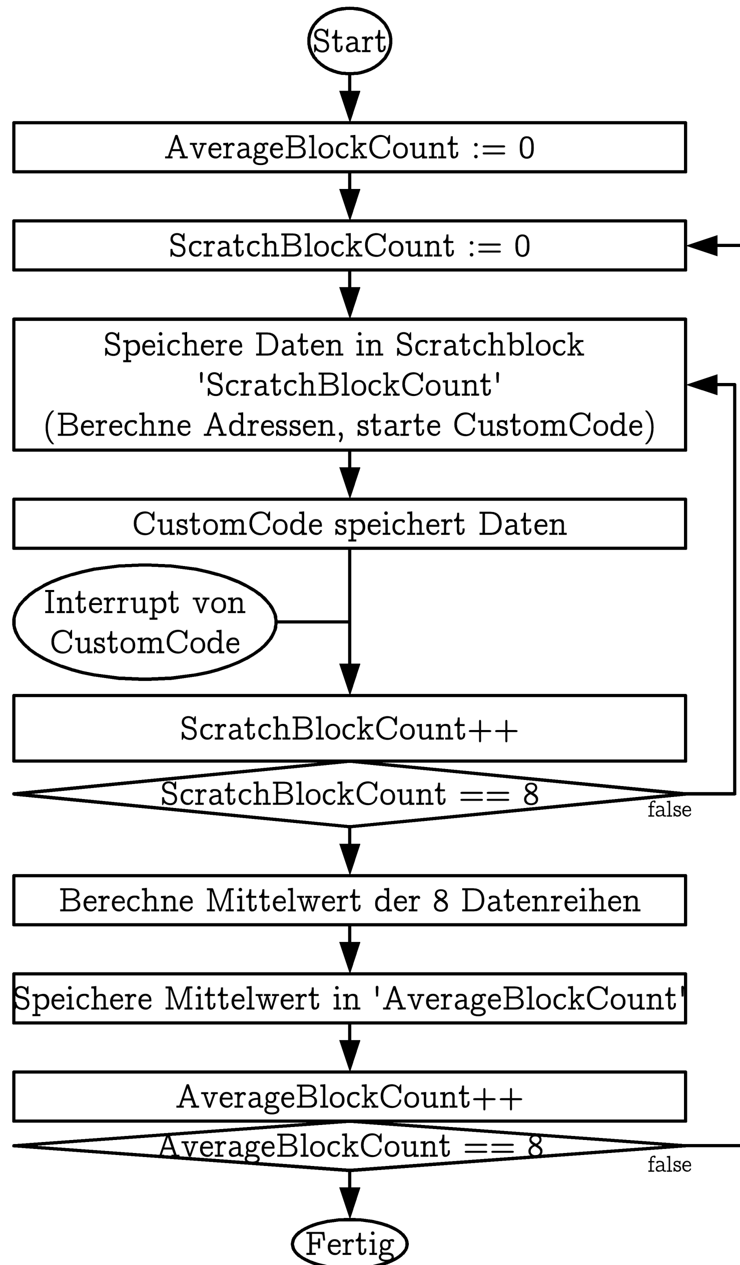



Abbildung 5.4: Mittelwertbildung 64 Ablauf

da nur ein begrenzter Speicher zur Verfügung steht. Diese variable Auflösung

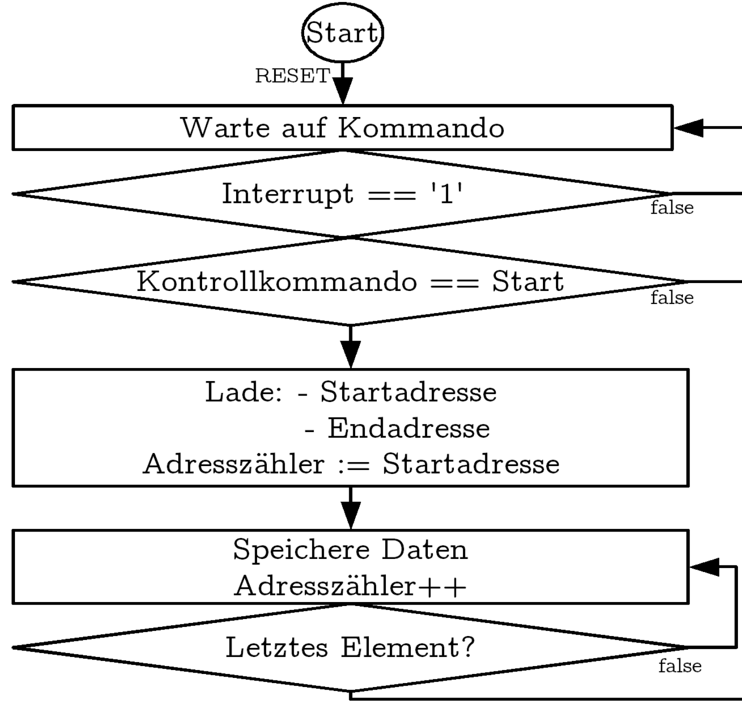



Abbildung 5.5: VHDL Modul Statemachine

werden durch ein *Clock-Divider* Modul erreicht. Der ADC liefert seine Daten immer mit der vollen Geschwindigkeit, das VHDL Modul arbeitet aber mit der durch das *Clock-Divider* Modul angepassten Taktfrequenz und übernimmt somit nur jedes *x'te* Sample. Die verschiedenen Einstellungen sind in Tabelle 5.3 ersichtlich.

| ClockDivider | ClockDivider | Frequenz |
|---|---|---|
| Index | [1] | [MHz] |
| 0 | 1 | 125 |
| 1 | 2 | 62.5 |
| 2 | 4 | 31.25 |
| 3 | 8 | 15.625 |
| 4 | 16 | 7.8125 |
| 5 | 32 | 3.90625 |
| 6 | 64 | 1.953125 |
| 7 | 128 | 0.9765625 |
| 8 | 256 | 0.48828125 |
| 9 | 512 | 0.244140625 |
| 10 | 1024 | 0.122070313 |

Tabelle 5.3: ClockDivider Modul Daten

# 6 Verfahren zur direkten Abklingzeitmessung

## 6.1 Messung Grundgerüst

Die Aufgaben des Mikrocontrollers sind: Kommunikation mit etwaiger Host-Software,das Starten des VHDL Moduls, welches für das Einlesen der ADC-Daten zuständig ist sowie die Auswertung der gespeicherten Daten. Die Messung startet sobald der Mikrocontroller den entsprechenden Befehl von einer externen Schnittstelle (Serielles Port) erhält. Sobald alle Daten im Speicher sind wird eine Vorverarbeitung durchgeführt und danach die Parameter der Kurve (Tau etc.) berechnet. Abbildung 6.1 illustriert diesen Ablauf.

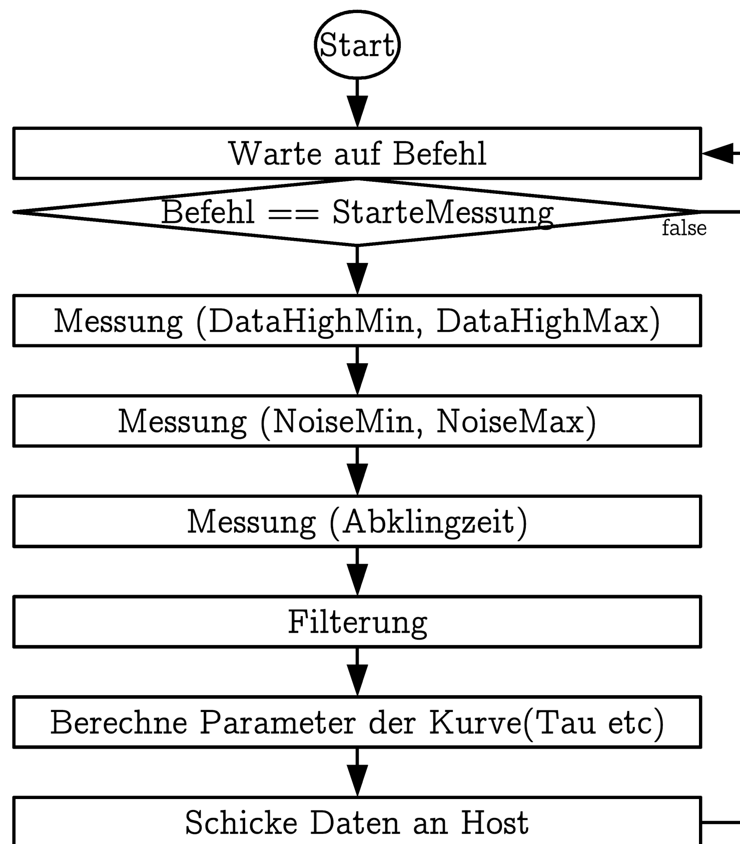



Abbildung 6.1: Mikrocontroller Ablauf Statemachine

## 6.2 Initiale Messung

Die Initiale Messung dient zur Erfassung von ungefilterten Messkurven, welche in einem späteren Schritt (siehe Kapitel 7) als Lernkurven verwendet werden. Diese Messung ist nur einmal zur Charakterisierung eines neuen opto-chemischen Sensors notwendig.

### 6.2.1 Wahl des Speicherfensters

Zur Wahl des Speicherfensers wird ein passender Takt-Teiler gewählt, bei dem das zugehörige Speicherfenster die Möglichkeit bietet alle Abklingzeiten des jeweiligen opto-chemischen Sensors, je nach Umgebungsbedingung, zu erfassen. Tabellen 6.1, 6.2 und 6.3 geben einen Überblick über die verwendeten Speicherfenster für die diversen Sensoren. Als Kriterium für die Auswahl dient hierbei der berechnet Wert *Tau/Speicherfenster*. Er gibt an wieviel von der Abklingzeit, gemessen in Tau, im Speicher zur Verfügung stehen wird. Um jeweils die gesamte Kurve betrachten zu können sollte dieser Faktor bei mindestens 5 liegen. Dies ist aber aufgrund der breiten Streuung dieses Parameters (siehe Tabelle 5.1 Taktteiler 1024) nicht immer möglich. Es wurden jeweils 2 erfolgversprechende Bereiche gewählt (fett markiert). Die Referenz TAU Werte wurden aus der Messung (Tabelle 3.7) in Kapitel 3 entnommen.

| | | | PtTFPP | |
|---|---|---|---|---|
| | | TAU: | 2.20E-05 [s] | 6.48E-05 [s] |
| DividerIndex | Taktteiler | Speicher [ns] | Air [Tau/Speicherfenster] | N2 |
| 0 | 1 | 8192 | 0.37 | 0.13 |
| 1 | 2 | 16384 | 0.75 | 0.25 |
| 2 | 4 | 32768 | 1.49 | 0.51 |
| 3 | 8 | 65536 | 2.98 | 1.01 |
| 4 | 16 | 131072 | 5.96 | 2.02 |
| 5 | 32 | 262144 | **11.92** | **4.04** |
| 6 | 64 | 524288 | **23.85** | **8.09** |
| 7 | 128 | 1048576 | 47.69 | 16.17 |
| 8 | 256 | 2097152 | 95.39 | 32.35 |
| 9 | 512 | 4194304 | 190.78 | 64.69 |
| 10 | 1024 | 8388608 | 381.55 | 129.39 |

Tabelle 6.1: PtTFPP Wahl des Taktteilers bzw. Speicherfenster

| | | | RuDPP | |
|---|---|---|---|---|
| | | TAU: | 5.07E-06 [s] | 6.1E-06 [s] |
| DividerIndex | Taktteiler | Speicher [ns] | Air [Tau/Speicherfenster] | N2 |
| 0 | 1 | 8192 | 1.62 | 1.34 |
| 1 | 2 | 16384 | 3.23 | 2.69 |
| 2 | 4 | 32768 | **6.47** | **5.37** |
| 3 | 8 | 65536 | **12.94** | **10.74** |
| 4 | 16 | 131072 | 25.87 | 21.49 |
| 5 | 32 | 262144 | 51.75 | 42.97 |
| 6 | 64 | 524288 | 103.50 | 85.94 |
| 7 | 128 | 1048576 | 207.00 | 171.88 |
| 8 | 256 | 2097152 | 413.99 | 343.76 |
| 9 | 512 | 4194304 | 827.98 | 687.53 |
| 10 | 1024 | 8388608 | 1655.96 | 1375.05 |

Tabelle 6.2: RuDPP Wahl des Taktteilers bzw. Speicherfenster

### 6.2.2 Ermittelung der ungefilterten Signale und Parameter

Gemittelte (Mittelwert 64) Abklingzeiten mit fest eingestelltem Taktteiler werden für die Weiterverarbeitung aufgenommen. Die Taktteiler wurden in Tabelle 6.1, Tabelle 6.2 und Tabelle 6.3 festgesetzt. Abbildung 6.2 zeigt ein Beispiel eines ungefilterten Signales für einen PtTFPP Sensor.

| | | | PdTFPP | | |
|---|---|---|---|---|---|
| | | TAU: | 1.54E-04 | 1.71E-04 | 8.53E-04 |
| | | | [s] | [s] | [s] |
| DividerIndex | Taktteiler | Speicher | 5% O2 | 0.1% O2 | N2 |
| | | [ns] | [Tau/Speicherfenster] | | |
| 0 | 1 | 8192 | 0.05 | 0.05 | 0.01 |
| 1 | 2 | 16384 | 0.11 | 0.10 | 0.02 |
| 2 | 4 | 32768 | 0.21 | 0.19 | 0.04 |
| 3 | 8 | 65536 | 0.43 | 0.38 | 0.08 |
| 4 | 16 | 131072 | 0.85 | 0.77 | 0.15 |
| 5 | 32 | 262144 | 1.70 | 1.53 | 0.31 |
| 6 | 64 | 524288 | 3.40 | 3.06 | 0.61 |
| 7 | 128 | 1048576 | 6.81 | 6.13 | 1.23 |
| 8 | 256 | 2097152 | 13.62 | 12.25 | 2.46 |
| 9 | 512 | 4194304 | **27.24** | **24.51** | **4.92** |
| 10 | 1024 | 8388608 | **54.48** | **49.01** | **9.83** |

Tabelle 6.3: PdTFPP Wahl des Taktteilers bzw. Speicherfensters

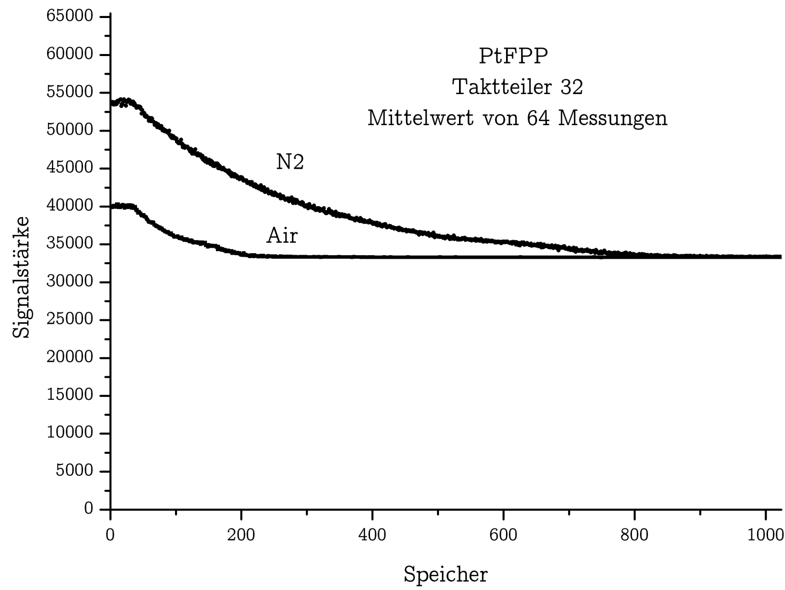



Abbildung 6.2: PtTFPP Messkurve, Mittelwert 64, Taktteiler 32, Umgebung Luft und N2

Zusätzlich zur Messkurve werden folgende Parameter ermittelt und gespeichert:

- Grundfrequenz
- Taktteiler
- DataHighMin (siehe Abbildung 6.3)
- DataHighMax (siehe Abbildung 6.3)
- NoiseMin (siehe Abbildung 6.3)
- NoiseMax (siehe Abbildung 6.3)

Die Grundfrequenz und der Taktteiler dienen zur Bestimmung der Zeitparameter. DataHighMin, DataHighMax, NoiseMin und NoiseMax werden von den Algorithmen der Stufe 1 und Stufe 2 (siehe Kapitel 7) verwendet.

**DataHighMin, DataHighMax** Um diese Parameter zu ermitteln wird der opto-chemische Sensor optisch angeregt und die Daten werden bei bestehender Anregung ermittelt. Als Grundlage für die Berechnung dieser Parameter wird ein gemitteltes (Mittelwert 64) Signal verwendet.

**NoiseMin, NoiseMax** Zur Messung dieser Parameter bleibt die optische Anregung deaktiviert und der Rauschteppich wird aufgenommen. Dies erfolgt auch mit einem gemittelten (Mittelwert 64) Signal.

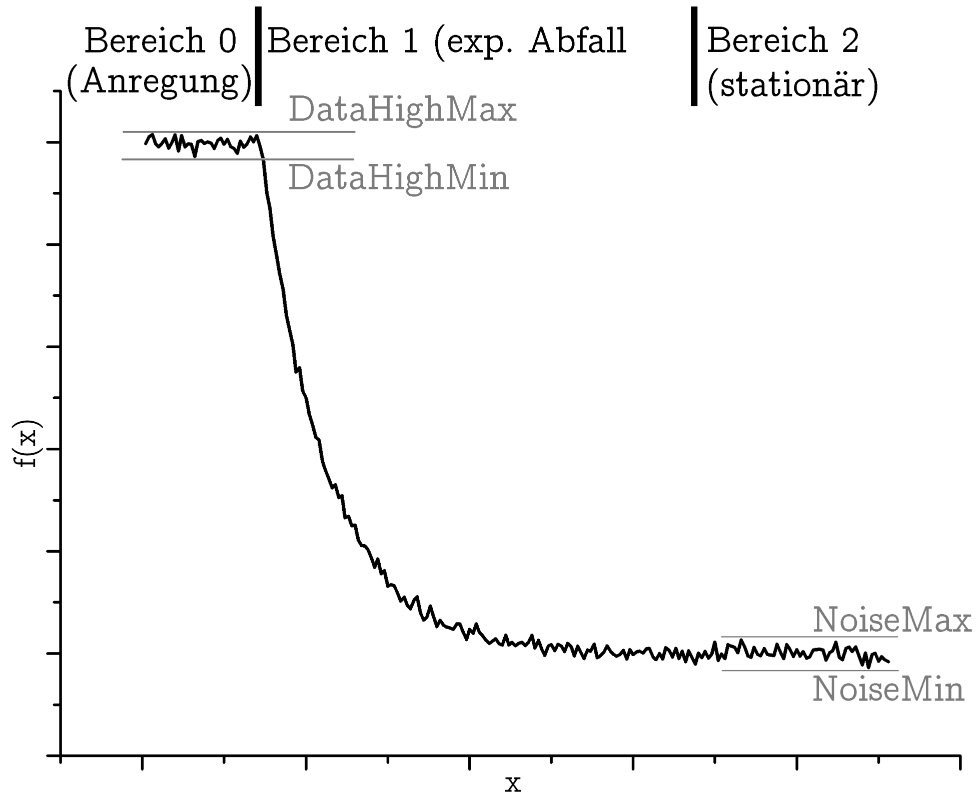



Abbildung 6.3: Ermittelte Parameter der ungefilterten Messkurve einer Beispielkurve

# 7 Evaluierung der geeignetsten Parameter

Ein automatisches Bewertungssystem wählt aus den verfügbaren Filtern und Berechnungsalgorithmen die beste Kombination aus, welche schlussendlich als Software in den Mikrocontroller implementiert werden soll. Die dazu verwendeten Filter und Algorithmen werden am Anfang dieses Kapitels beschrieben. Die Ergebnisse der Berechnung der Parameter sowie die Evaluierung ist im Schluss des Kapitels zu finden.

## 7.1 Automatisches Evaluierungs und Bewertungssystem

Ausgehend von folgenden Daten :

- Messkurve (ungefiltert, Mittelwert 64), Kapitel 6.2
- Grundfrequenz, Kapitel 6.2
- Taktteiler, Kapitel 6.2
- DataHighMin (siehe Abbildung 6.3), Kapitel 6.2
- DataHighMax (siehe Abbildung 6.3), Kapitel 6.2
- NoiseMin (siehe Abbildung 6.3), Kapitel 6.2
- NoiseMax (siehe Abbildung 6.3), Kapitel 6.2
- Referenz-TAU, Kapitel 3

wird die beste Kombination an Filtern, Berechnungsalgorithmen und deren Parametern ermittelt. Dazu werden alle Kombinationen in MATLAB durchgerechnet und die Ergebnisse verglichen.

## 7.2 Filter und Algorithmen

### 7.2.1 Stufe0: digitale Filter

Digitale Filter erlauben eine Rauschunterdrückung von zeit- und wertdiskreten Datenreihen [18]. Im automatischen Bewertungssystem stehen folgende Filter mit verschiedenen Parametern zur Verfügung:

- Tiefpass

- Tiefpass mit Hanning Fenster
- Tiefpass mit Hamming Fenster
- Tiefpass mit Blackman Fenster
- Tiefpass mit Bartlett Fenster
- Moving Average Filter
- Kein Filter

**Tiefpässe** Bei den Tiefpässen sind die Parameter Fensterbreite, Grundfrequenz und Grenzfrequenz fix eingestellt. Die Grundfrequenz liegt hier bei 125MHz. Die Fensterbreite variiert von 5 bis 61 Punkten (in 4er Schritten). Die Grenzfrequenz startet bei 500kHz und endet bei 10MHz (und steigert sich in 500kHz Schritten). Somit sind bei jedem der Tiefpässe 300 Varianten verfügbar.

**Moving Average** Hier wird ein laufender Mittelwert berechnet. Die Fensterbreite läuft von 5 bis 81 Punkten (in 4er Schritten). Somit sind hier 20 Variationen verfügbar.

**Kein Filter** Hier wird ein Filter mit der Filterfunktion [1] verwendet. Es findet somit keine Filterung statt.

### 7.2.2 Stufe1: Ermittelung Start der Abklingfunktion

Diese Stufe dient der Ermittelung der Position des Übergangs von Bereich 0 (Angeregter Zustand) in den Bereich 1 (exponentieller Abfall). Abbildung 6.3 illustriert diesen Übergang und die Bereiche.
In dieser Kategorie stehen 5 verschiedene Ansätze zur Verfügung. In Kapitel 7.3 erfolgt eine nähere Betrachtung der besten Varianten.

### 7.2.3 Stufe2: Ermittelung TAU

Für die Berechnung des TAU-Wertes stehen 11 verschiedene Verfahren zur Verfügung. Eine genauere Beschreibung der besten Methoden ist in Kapitel 7.3 zu finden.

### 7.2.4 Zusammenfassung, Auswertung

Es stehen im automatischen Evaluierungs- und Bewertungssystem folgende Methoden zur Verfügung:

- Filterung: 1521 Filter
  - $5 \cdot 300$ Tiefpässe = 1500
  - 20 Moving Average Filter
  - 1 Keine Filterung Filter
- 5 Methoden zum Auffinden des Kurvenabfalls
- 11 Methoden zum Berechnen von TAU

Dies ergibt in Summe 83655 Variationen die für jede Messkurve berechnet werden.
Ein detailliertes Beispiel für das Ergebnis eines Durchlaufs liefert Tabelle 7.2.

Die umfangreiche Auswertung ist auf einem handelsüblichen PC (®Intel ™Core2 DUO 3GHz, 3GB Ram, Windows XP) mit der hier vorgestellten Anzahl von Bearbeitungsstufen in MATLAB (Version R2009a) gerade noch durchführbar. Bei hinzufügen von mehr Filtern oder Algorithmen kann eine Verarbeitung von Matlab aufgrund von Speichermangel nicht mehr durchgeführt werden. Die Laufzeit einer solchen Evaluierung hängt stark von der Anzahl der Messkurven ab. Tabelle 7.1 liefert hierzu eine Übersicht.

| Lernkurven | Evaluierungsdauer [min] |
|---:|---:|
| 2 | 15 |
| 10 | 90 |

Tabelle 7.1: Laufzeit einer Messkurven-Evaluierung (®Intel ™Core2 DUO 3GHz, 3GB Ram, Windows XP)

Das Berechnungsergebnis für PdTFPP (Taktteiler 512) mit der besten Kombination der Algorithmen zeigt Tabelle 7.2. Hierbei ist das beste Ergebnis jenes, welches den geringsten Fehler (errSum) aufweist. Dieser Fehler ist die Summe aller Fehler über alle Lernkurven, welche dem Evaluierungsmodul zur Auswertung zur Verfügung stehen. Der Fehler pro Kurve ist der Fehler

(in Prozent), der Abweichung des ermittelten TAU Wertes, im Vergleich zum Referenz-TAU.

| | |
|---:|---:|
| Stufe 0 | movingAverage |
| Stufe 0 Fensterbreite | 69 |
| Stufe 1 | findDecayStart1 |
| Stufe 2 | Fit1e2xDecayStart |
| errSum | 4.167 |
| errSum/cnt | **1.389** |
| MinErr | -2.416 |
| MaxErr | 0.683 |
| 0.1% O2_fixCDiv512 | 0.683 |
| 5.0% O2_fixCDiv512 | -1.068 |
| 0.0% O2_fixCDiv512 | -2.416 |

Tabelle 7.2: PdTFPP (Taktteiler 512) Berechnungsergebnis (bestes Ergebnis)

## 7.3 Ergebnisse

Die besten Ergebnisse sind in Tabelle 7.3 verfügbar. Die Verfahren werden im Anschluss kurz beschrieben.

| Sensor | Takt-teiler | Stufe 0 | Stufe 1 | Stufe 2 |
|---:|---:|---:|---:|---:|
| PtTFPP | 32 | movingAverage_37 | findD.S.0 | Fit1eD.S. |
| PtTFPP | 64 | blackman0_29_1000000 | findD.S.0 | Fit1e_top_D.S. |
| PdTFPP | 512 | movingAverage_69 | findD.S.1 | Fit1e2xD.S. |
| PdTFPP | 1024 | movingAverage_73 | findD.S.1 | Fit2x1eD.S. |
| RuDPP | 4 | square_53_7000000 | findD.S.1 | Fit2x1eD.S. |
| RuDPP | 8 | bartlett0_37_8000000 | findD.S.1 | Fit2x1eD.S. |

Tabelle 7.3: Ergebnisse der Evaluierung der besten Parameter - pro Umgebung jeweils 1 Messreihe als Trainigsdaten
D.S. ... DecayStart

### 7.3.1 Auszug aus den Algorithmen Stufe 1

Die Algorithmen der Stufe 1 dienen zum Auffinden des Punktes, bei dem die Messkurve den Übergang vom Anregungsbereich in den Bereich des exponentiellen Abfalls vollzieht (siehe Abbildung 6.3).

**findDecayStart0** Diese Funktion sucht den ersten Wert, bei dem eine Kurve unter eine bestimmte Grenze fällt. Die Berechnung dieser Grenze ist in Gleichung (7.3.1) angegeben.

$$Grenze_{findDecayStart0} = dataMax - (NoiseMax - NoiseMin) \tag{7.3.1}$$
$$dataMax = \text{Maximum der Daten}$$

**findDecayStart1** Funktioniert wie *findDecayStart0*, hat aber eine andere Grenze. Die Berechnung dieser Grenze ist in in Gleichung (7.3.2) angegeben.

$$Grenze_{findDecayStart1} = dataMax - \frac{(NoiseMax - NoiseMin)}{2} \tag{7.3.2}$$
$$dataMax = \text{Maximum der Daten}$$

### 7.3.2 Auszug aus den Algorithmen Stufe 2

Ausgehend von dem in Stufe 1 berechneten Start der Kurve wird hier das TAU berechnet.

**Fit1eDecayStart** Hier wird der erste Punkt in der Kurve gesucht, welcher (beginnend bei Punkt aus Stufe 1) unter eine gewisse Grenze fällt.

```
% Fit1eDecayStart.m
expDecay = 1;

maxVal = data(decayStartPosition, 1);
expVal = exp(-expDecay) * (maxVal-min(data));
expVal = expVal + min(data);
tauPosition = 0;
for i = decayStartPosition:length(data)
    if data(i,1) < expVal
```

```
        tauPosition = i;
        break;
    end
end
```

**Fit1eTopDecayStart** Funktioniert wie Fit1eDecayStart, jedoch wird das Level um die Höhe des Rauschteppichs (*noiseDelta*) angehoben.

```
% Fit1eTopDecayStart.m
expDecay = 1;

maxVal = data(decayStartPosition, 1);
noiseDelta = abs(NoiseMax - NoiseMin);
expVal = exp(-expDecay) * (maxVal-min(data)+(noiseDelta));
expVal = expVal + min(data);
tauPosition = 0;
for i = decayStartPosition:length(data)
    if data(i,1) < expVal
        tauPosition = i;
        break;
    end
end
```

**Fit1e2xDecayStart** Funktioniert ähnlich wie Fit1eDecayStart. Jedoch werden hier 2 TAU Werte berechnet (jeweils mit einem anderen Startpunkt) und zum Schluss gemittelt.

```
% Fit1e2xDecayStart.m
expDecay = 1;

maxVal = data(decayStartPosition, 1);
expVal = exp(-expDecay) * (maxVal-min(data));
expVal = expVal + min(data);
tau = 0;
tau1postition = 1;
for i = decayStartPosition:length(data)
    if data(i,1) < expVal
```

```
            tau = i;
            tau1postition = i;
            break;
        end
    end

    expDecay = 1;

    maxVal = data(tau1postition, 1);
    expVal = exp(-expDecay) * (maxVal-min(data));
    expVal = expVal + min(data);
    tau2postition = 0;
    for i = decayStartPosition:length(data)
        if data(i,1) < expVal
            tau2postition = i;
            break;
        end
    end

    % Beide Tau Werte berechnen und mitteln
```

**Fit2x1eDecayStart** Hier werden 2 TAU Werte berechnet. Am Anfang wird ein einfacher exponentieller Abfall und danach ein zweifacher exponentieller Abfall berechnet. Der zweite TAU-Wert wird anschließend halbiert und mit dem ersten gemittelt.

```
    % Fit2x1eDecayStart.m
    expDecay = 1;

    maxVal = data(decayStartPosition, 1);
    expVal = exp(-expDecay) * (maxVal-min(data));
    expVal = expVal + min(data);
    tau1postition = 0;
    for i = decayStartPosition:length(data)
        if data(i,1) < expVal
            tau1postition = i;
            break;
```

```
    end
end

if tau1postition > 0
    deltaTimeSec = 1/(baseFrequencyKHZ*1000/ClkDivider) ;
    tau1postition = (tau1postition - decayStartPosition)*deltaTimeSec;
    tau1e = tau1postition / expDecay;
end


expDecay = 2;

maxVal = data(decayStartPosition, 1);
expVal = exp(-expDecay) * (maxVal-min(data));
expVal = expVal + min(data);
tau2postition = 0;
for i = decayStartPosition:length(data)
    if data(i,1) < expVal
        tau2postition = i;
        break;
    end
end

% Hier noch Tau 2 berechnen
% Tau 2 halbieren
% Tau 1 und Tau 2 mitteln
```

## 7.4 Evaluierung der gefundenen Parameter

Hier wurden für jede Messung die entsprechenden Filter und Algorithmen (siehe Tabelle 7.3) im Mikrocontroller in C implementiert und getestet. Abbildung 7.1 zeigt ein Beispiel für ein gefiltertes Signal (Moving Average Filter). Eine Messreihe eines PtTFPP Sensors, bei der der opto-chemische Sensor zuerst Umgebungsluft, dann N2 und zum Schluss wieder Umgebungsluft ausgesetzt war ist in Abbildung 7.2 zu sehen. Abbildung 7.3 zeigt die Daten einer Messreihe bei der ein PtTFPP Sensor nur der Umgebung Luft ausgesetzt war, inklusive der dazugehörigen Standardabweichung. Gleichung (7.4.1) gibt

die Berechnung der O2 Auflösung an. Dieser Wert gibt Auskunft über die Auflösung, in *Sauerstoff Prozent* gerechnet, die mit dem jeweiligen optochemischen Sensor und dem Messsystem erreicht werden kann. Man sieht in Tabelle 7.4, daß ein PtTFPP Sensor, bei dem TAU auf 33.9% (6.483E-5s für N2 auf 2.199E-05 für Luft) sinkt eine deutlich bessere Auflösung liefert (Tabelle 7.4) als ein RuDPP Sensor(Tabelle 7.7), bei dem TAU auf nur 83.0% absinkt (6.101E-6 für N2 auf 5.066E-6 für Luft). Hierbei wurden jeweils die besten Ergebnisse betrachtet. Dies ist durch den geringeren Bereich der Änderung des TAU-Wertes bei RuDPP im Vergleich zum PtTFPP Sensor begründbar.

$$t_{O2\,\%} = \frac{DeltaO2}{\frac{DeltaTAU}{2 \cdot Standardabweichung_{max}}} \tag{7.4.1}$$

$$t_{O2\,\%, PdTFPP1024} = \frac{5\%}{\frac{8{,}68E^{-4} - 1{,}46E^{-4}}{2 \cdot 4{,}68E^{-6}}} = 0,064789\, O2\,\% \quad (sieheTabelle 7.6) \tag{7.4.2}$$

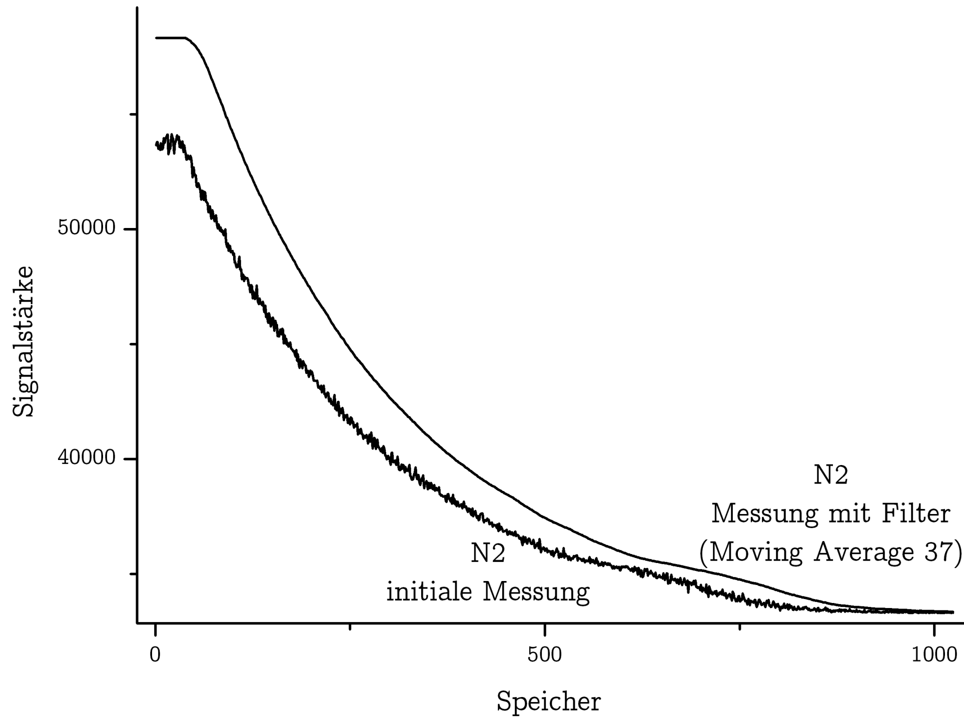



Abbildung 7.1: PtTFPP (Taktteiler 32) N2, Messsignal mit und ohne Filterung - vergrößerte Abbildung

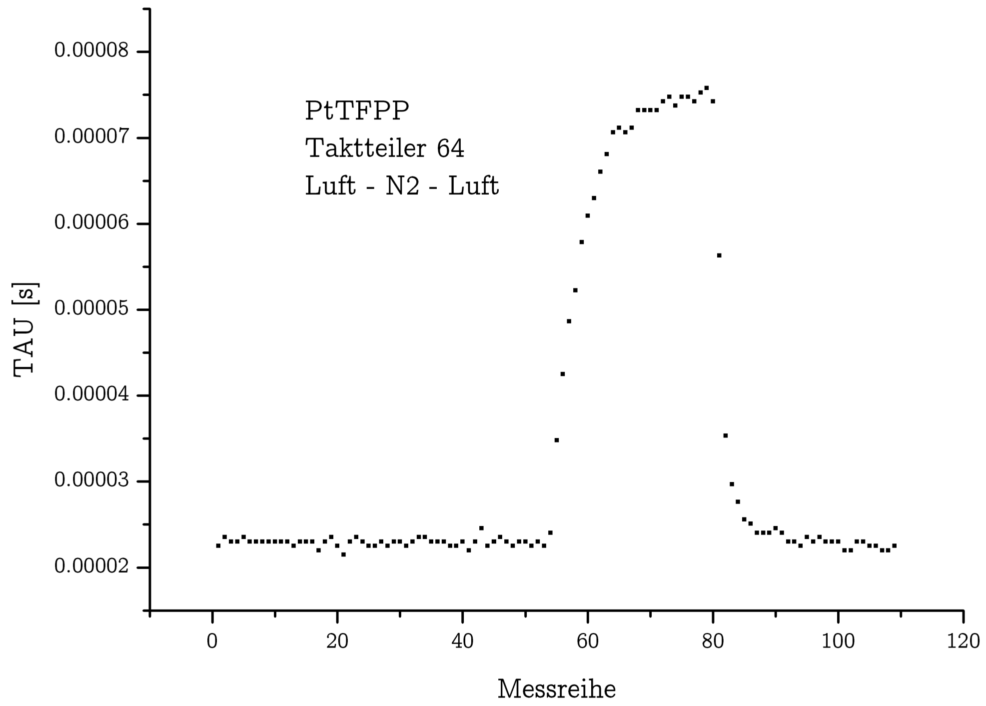



Abbildung 7.2: Messreihe mit wechselnder Umgebung (Luft - N2 - Luft)

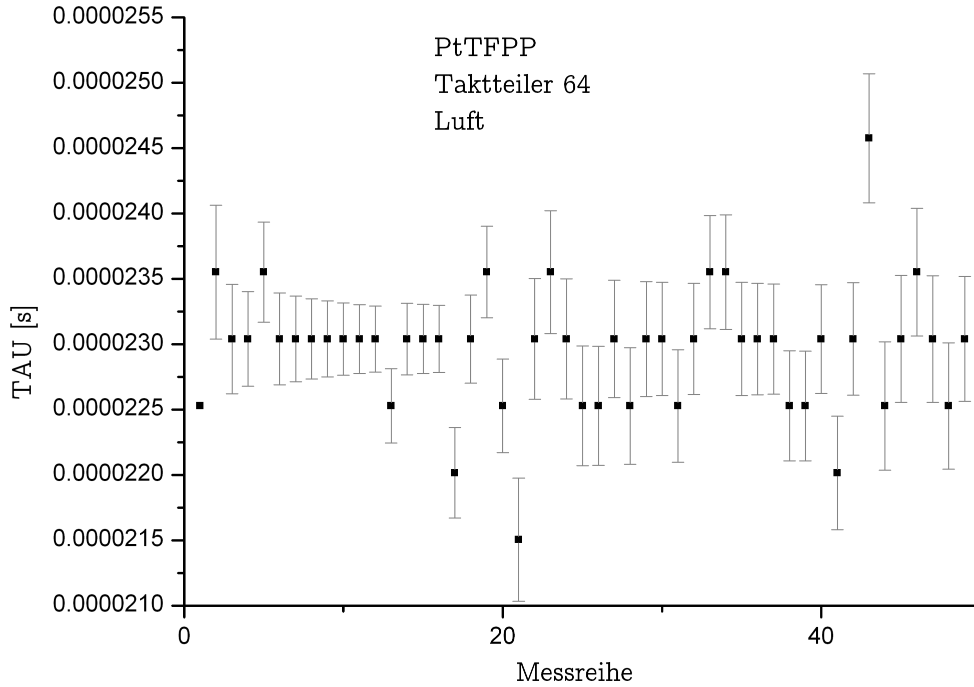



Abbildung 7.3: Messreihe PtTFPP Taktteiler64 Luft und Standardabweichung

Die Ergebnisse der Messung sind in Tabelle 7.4, 7.5, 7.6 und 7.7 aufgeführt.

| Sensor | | PtTFPP | | | |
|---|---|---|---|---|---|
| Taktteiler | | 32 | 32 | 64 | 64 |
| Trainingsdaten | | 1 | | | |
| Umgebung | | Luft | N2 | Luft | N2 |
| Referenz TAU | [s] | 2.199E-05 | 6.483E-05 | 2.199E-05 | 6.483E-05 |
| TAU Mittelwert | [s] | 2.283E-05 | 6.731E-05 | 2.295E-05 | 7.031E-05 |
| TAU Std.Abw. | [s] | 2.367E-07 | 4.051E-07 | 4.680E-07 | 6.792E-07 |
| O2 Auflösung | [O2%] | 0.382 | | 0.601 | |

Tabelle 7.4: PtTFPP Ergebnisse (Messreihe mit 50 Samples)

| Sensor | | PdTFPP | | |
|---|---|---|---|---|
| Taktteiler | | 512 | 512 | 512 |
| Trainingsdaten | | 1 | | |
| Umgebung | | 5% O2 | 0.1% O2 | N2 |
| Referenz TAU | [s] | 1.540E-04 | 1.711E-04 | 8.534E-04 |
| TAU Mittelwert | [s] | 1.442E-04 | 1.588E-04 | 8.743E-04 |
| TAU Std.Abw. | [s] | 7.167E-06 | 4.353E-06 | 3.918E-06 |
| O2 Auflösung | [O2%] | 0.0982 | | |

Tabelle 7.5: PdTFPP Ergebnisse (Messreihe mit 50 Samples) Taktteiler 512

| Sensor | | PdTFPP | | |
|---|---|---|---|---|
| Taktteiler | | 1024 | 1024 | 1024 |
| Trainingsdaten | | 1 | | |
| Umgebung | | 5% O2 | 0.1% O2 | N2 |
| Referenz TAU | [s] | 1.540E-04 | 1.711E-04 | 8.534E-04 |
| TAU Mittelwert | [s] | 1.456E-04 | 1.674E-04 | 8.678E-04 |
| TAU Std.Abw. | [s] | 4.282E-06 | 4.679E-06 | 2.767E-06 |
| O2 Auflösung | [O2%] | 0.0648 | | |

Tabelle 7.6: PdTFPP Ergebnisse (Messreihe mit 50 Samples) Taktteiler 1024

| Sensor | | RuDPP | | | |
|---|---|---|---|---|---|
| Taktteiler | | 4 | 4 | 8 | 8 |
| Trainingsdaten | | 1 | | | |
| Umgebung | | Luft | N2 | Luft | N2 |
| Referenz TAU | [s] | 5.066E-06 | 6.101E-06 | 5.066E-06 | 6.101E-06 |
| TAU Mittelwert | [s] | 4.985E-06 | 5.901E-06 | 5.510E-06 | 6.872E-06 |
| TAU Std.Abw. | [s] | 1.535E-07 | 8.695E-08 | 5.500E-07 | 8.510E-07 |
| O2 Auflösung | [O2%] | 7.029 | | 26.187 | |

Tabelle 7.7: RuDPP Ergebnisse (Messreihe mit 50 Samples)

## 7.5 Erhöhung der Trainingsdaten für RuDPP

Um ein stabileres Testsetup zu erhalten, werden für diese Ausbaustufe mehrerer Trainingsdaten (Messdaten - siehe Initiale Messung Kapitel 6.2) für jede Umgebung ermittelt. Eine Überprüfung dieser Annahme erfolgt anhand der schlechtesten Ergebnisse (RuDPP Sensor - Tabelle 7.7).
Für dieses Testsetup wurden 5 Messsignale für jede Umgebung aufgenommen. Die Ergebnisse der Evaluierung sind in Tabelle 7.8 ersichtlich.

| Sensor | Takt-teiler | Stufe 0 | Stufe 1 | Stufe 2 |
|---|---|---|---|---|
| RuDPP | 4 | square_57_500000 | findDecayStart1 | Fit2eDecayStart |
| RuDPP | 8 | square_45_4000000 | findDecayStart0 | Fit2x1eDecayStart |

Tabelle 7.8: Ergebnisse der Evaluierung der besten Parameter mit 5 Messreihen als Trainigsdaten für RuDPP

In Tabelle 7.9 ist eine deutliche Verbesserung der Standardabweichung sowie der Sauerstoffauflösung ersichtlich. Abbildung 7.4 verdeutlicht die Resultate. Hier ist eine markante Verbesserung der Standardabweichung zu erkennen.

| Sensor | | RuDPP | | | |
|---|---|---|---|---|---|
| Taktteiler | | 4 | 4 | 8 | 8 |
| Trainigsdaten | | 1 | | | |
| Umgebung | | Luft | N2 | Luft | N2 |
| Referenz TAU | [s] | 5.066E-06 | 6.101E-06 | 5.066E-06 | 6.101E-06 |
| TAU Mittelwert | [s] | 4.985E-06 | 5.901E-06 | 5.510E-06 | 6.872E-06 |
| TAU Std.Abw. | [s] | 1.535E-07 | 8.695E-08 | 5.500E-07 | 8.510E-07 |
| O2 Auflösung | [O2%] | 7.029 | | 26.187 | |
| Sensor | | RuDPP | | | |
| Taktteiler | | 4 | 4 | 8 | 8 |
| Trainigsdaten | | 5 | | | |
| Umgebung | | Luft | N2 | Luft | N2 |
| Referenz TAU | [s] | 5.066E-06 | 6.101E-06 | 5.066E-06 | 6.101E-06 |
| TAU Mittelwert | [s] | 5.022E-06 | 5.897E-06 | 5.108E-06 | 6.316E-06 |
| TAU Std.Abw. | [s] | 1.677E-08 | 1.654E-08 | 3.027E-08 | 2.767E-08 |
| O2 Auflösung | [O2%] | 0.803 | | 1.050 | |

Tabelle 7.9: RuDPP Ergebnisse, Vergleich bei Erhöhung der Trainingsdaten

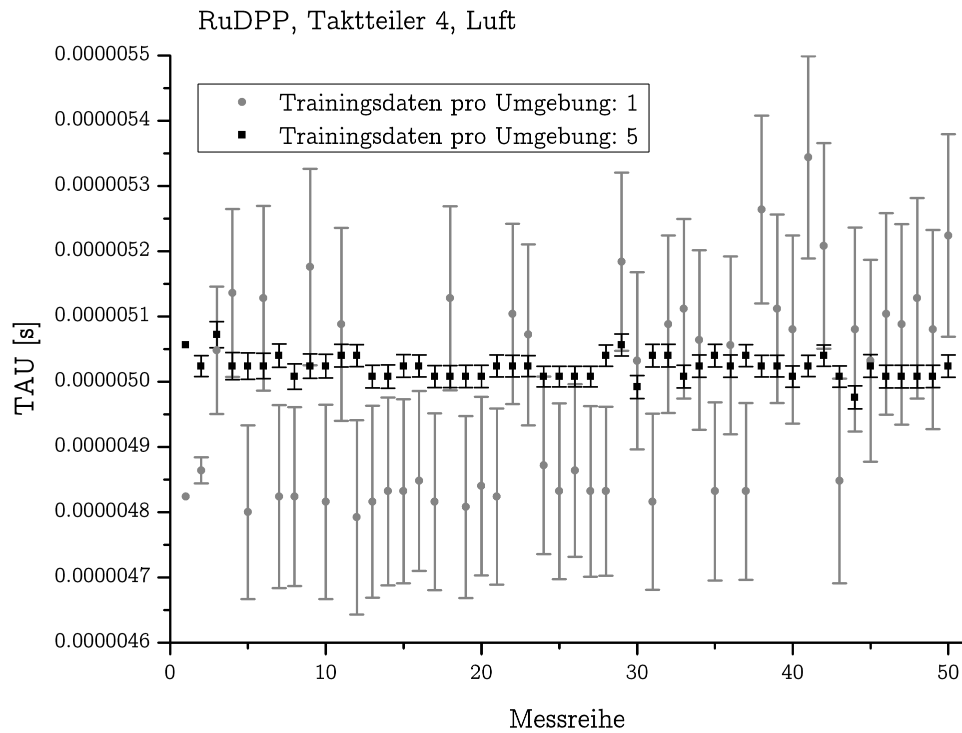



Abbildung 7.4: RuDPP - Vergleich TAU Messung und Standardabweichung

# 8 Zusammenfassung, Diskussion, Ausblick

Es wurde eine Hardwareplattform zuzüglich der dazugehörigen Software zur direkten Messung von Abklingzeiten erstellt. Diese Hardwareplattform ist durch ihren umfassenden Aufbau auch für andere Anwendungen wie zum Beispiel eine schnellere Implementierung der bisherigen Phasenmessung [2] geeignet. Die vorgestellte Software-Konfiguration erlaubt eine Sauerstoff-Messung im Bereich der Umgebungsluft mit einer Auflösung von 0.382 O2% (siehe PtTFPP Tabelle 8.1). Bei Spurensensoren (PdTFPP) lässt sich bei Umgebungsbedingungen im Bereich von 0% O2 bis 5% O2 eine Messung mit einer Auflösung von 0.0648 O2% durchführen.

| Sensor | Taktteiler | Lernkurven pro Umgebung | O2 Auflösung [O2%] | Messbereich [O2%] |
|---:|---:|---:|---:|---:|
| PtTFPP | 32 | 1 | 0.382 | 0% bis 20,95% |
| PdTFPP | 1024 | 1 | 0.0648 | 0% bis 5% |
| RuDPP | 4 | 1 | 7.029 | 0% bis 20,95% |
| RuDPP | 4 | 5 | 0.803 | 0% bis 20,95% |

Tabelle 8.1: Zusammenfassung der besten Ergebnisse

## 8.1 Ausblick

Durch die Erhöhung der Anzahl der Lernkurven konnte eine deutliche Steigerung der messbaren O2% Auflösung erreicht werden. Da der Aufwand der Erhöhung der Anzahl der Lernkurven nur in der initalen Phase steigt, wird eine Steigerung dieser Maßnahme als logischer nächster Schritt zur Verbesserung der messbaren Auflösung gesehen.

Weiters könnte man durch einen festen digitalen Filter, welcher schon bei den initalen Messungen aktiviert wird, die Anzahl der Lernkurven dramatisch erhöhen, da eine große Komponente in der speicherintensiven Berechnung der besten Parameter in der MATLAB Software (Kapitel 7) wegfallen würde. Die Auswirkungen der Erhöhung der Anzahl der Lernkurven im Vergleich zu einem Wegfallen einer Auswahl an digitalen Filtern ist zu untersuchen.

Eine Optimierung der Speicherverwaltung in der MATLAB Software aus Kapitel 7 würde sich vermutlich durch die Möglichkeit der Erhöhung der Anzahl von Algorithmen bzw. Lernkurven positiv auf die Messauflösung niederschlagen.

Die Möglichkeit schnellere Abklingzeiten als 1µs messen zu können, könnte durch das Wegfallen der Bedingung, eine getreue Abbildung der Kurve zu erhalten, möglich sein.

Eine Verbesserung der Qualität des Signales (SNR, etc. ) könnte durch eine optimiertes Layout der Platine bezüglich der hohen Frequenzen erreicht werden. Eine getrennte Spannungsversorgung für den Analog-digital Wandler könnte hierbei auch zu Verbesserungen führen.
Um die Auflösung im Zeitbereich zu erhöhen kann ein schnellerer Analog-digital Wandler, bei gleichem FPGA eingesetzt werden.

# Literatur

[1] Bizzarri Alessandro, Koehler Hans, Cajlakovic Merima, Pasic Alen, Schaupp Lukas, Klimant Ingo, and Ribitsch Volker. *Continuous oxygen monitoring in subcutaneous adipose tissue using microdialysis*, 2006.

[2] Roland Almer. Phasenmessung zur bestimmung der lumineszenzlebensdauer von opto-chemischen sensoren mit anregungsfrequenzen bis 12,5 mhz. Master's thesis, Technical University GRAZ, 2009.

[3] Altium. *CR0121 TSK300A 32 bit RISC Processor*, 2008.

[4] Nils Wiberg Arnold F. Holleman, Egon Wiberg. *Lehrbuch der anorganischen Chemie*. Gruyter, 1995.

[5] Ribitsch Volker Cajlakovic Merima, Bizzarri Alessandro. *Luminescence lifetime-based carbon dioxide optical sensor for clinical applications.*, 2006.

[6] Analog Devices. *AD8138 - Low Distortion Differential ADC Driver*, 2006.

[7] Analog Devices. *AD9461 - 16-Bit, 130 MSPS IF Sampling ADC*, 2006.

[8] Richard C. Dorf. *The electrical engineering handbook*. CRC Press, 1997.

[9] hamamatsu. *Photomultiplier Tube Modules*, 2007.

[10] Philip C. D. Hobbs. *Building Electro-Optical Systems*. WILEY, 2008.

[11] Mark Johnson. *Photodetection and Measurement*. McGraw-Hill, 2003.

[12] Uwe Kreibig Jürgen Hüttermann, Alfred Trautwein. *Physik für Mediziner, Biologen, Pharmazeuten*. Walter de Gruyter, 2004.

[13] Joseph R. Lakowicz. *Principles of fluorescence spectroscopy*. Springer, 2006.

[14] Cajlakovic Merima, Bizzarri Alessandro, Konrad Christian, and Voraberger Hannes. *Optochemical Sensors Based on Luminescence*, 2006.

[15] NXP. *BC547 NPN general purpose transistors*, 2004.

[16] Institute of Electrical and Inc. Electronics Engineers. *IEEE Std 1057-2007, IEEE Standard for Digitizing Waveform Recorders*, 2008.

[17] Institute of Electrical and Inc. Electronics Engineers. *IEEE Std 1241-2000, IEEE Standard for Terminology and Test Methods for Analog-to-Digital Converters*, 2008.

[18] Alan V. Oppenheim. *Zeitdiskrete Signalverarbeitung*. Pearson Studium, 2004.

[19] Carolin Trojer. Phase measurement referencing systems and background correction for opto-chemical sensors. Master's thesis, Technical University GRAZ, 2006.

[20] Christoph Schenk Ulrich Tietze. *Halbleiter-schaltungstechnik*. Springer, 2002.

[21] Otto S. Wolfbeis. *Fiber Optic Chemical Sensors and Biosensors*, volume 1. CRC Press, 1991.

[22] Xilinx. *DS099 - Spartan-3 FPGA Family, Complete Data Sheet*, 2003.

[23] Xilinx. *DS123 - Platform Flash In-System Programmable Configuration PROMs*, 2008.

# Anhang

## Fotos

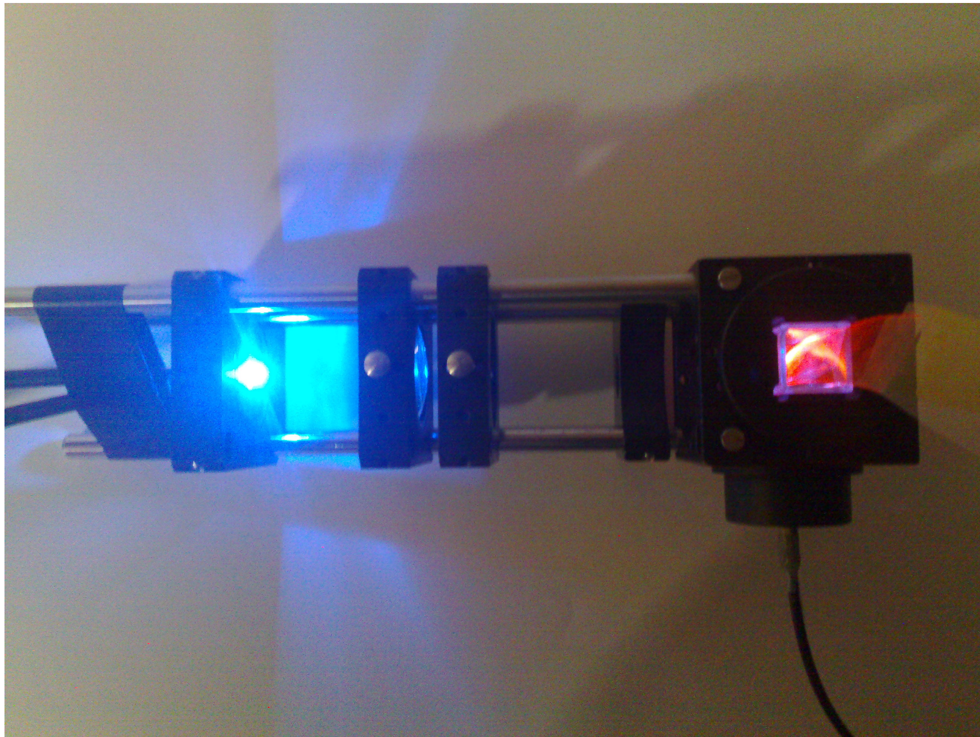


Abbildung 8.1: Messaufbau, Ansicht von oben.
Von links nach rechts: Anregung (LED mit sichtbarem Licht im blauen Spektralbereich), Linse, Linos Würfel mit Sensor in der Mitte ( Palladium - leuchtet orange im sichtbaren Spektralbereich). Der Emissionsfilter ist im Linos-Würfel verbaut. Am unteren Ende des Linos-Würfels geht ein Faserkabel weg (zum Photodetektor - nicht im Bild).

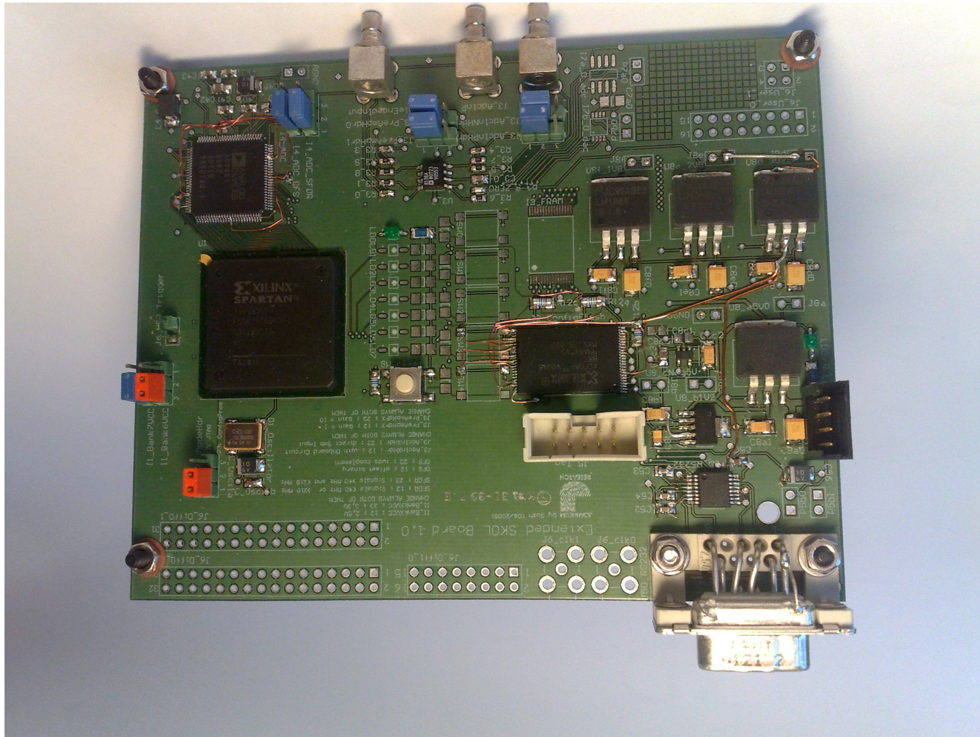



Abbildung 8.2: Printed Circuit Board
Für Details siehe Abbildung 4.10 und Tabelle 4.9

# Datenblätter

Philips Semiconductors — Product specification

## NPN general purpose transistors — BC546; BC547

**CHARACTERISTICS**

$T_{amb}$ = 25 °C unless otherwise specified.

| SYMBOL | PARAMETER | CONDITIONS | MIN. | TYP. | MAX. | UNIT |
|---|---|---|---|---|---|---|
| $I_{CBO}$ | collector-base cut-off current | $V_{CB}$ = 30 V; $I_E$ = 0 A | – | – | 15 | nA |
| | | $V_{CB}$ = 30 V; $I_E$ = 0 A; $T_j$ = 150 °C | – | – | 5 | μA |
| $I_{EBO}$ | emitter-base cut-off current | $V_{EB}$ = 5 V; $I_C$ = 0 A | – | – | 100 | nA |
| $h_{FE}$ | DC current gain | $V_{CE}$ = 5 V; $I_C$ = 10 μA; see Figs 2, 3 and 4 | | | | |
| | BC546A | | – | 90 | – | |
| | BC546B; BC547B | | – | 150 | – | |
| | BC547C | | – | 270 | – | |
| | DC current gain | $V_{CE}$ = 5 V; $I_C$ = 2 mA; see Figs 2, 3 and 4 | | | | |
| | BC546A | | 110 | 180 | 220 | |
| | BC546B; BC547B | | 200 | 290 | 450 | |
| | BC547C | | 420 | 520 | 800 | |
| | BC547 | | 110 | – | 800 | |
| $V_{CEsat}$ | collector-emitter saturation voltage | $I_C$ = 10 mA; $I_B$ = 0.5 mA | – | 90 | 250 | mV |
| | | $I_C$ = 100 mA; $I_B$ = 5 mA | – | 200 | 600 | mV |
| $V_{BEsat}$ | base-emitter saturation voltage | $I_C$ = 10 mA; $I_B$ = 0.5 mA; note 1 | – | 700 | – | mV |
| | | $I_C$ = 100 mA; $I_B$ = 5 mA; note 1 | – | 900 | – | mV |
| $V_{BE}$ | base-emitter voltage | $V_{CE}$ = 5 V; $I_C$ = 2 mA; note 2 | 580 | 660 | 700 | mV |
| | | $V_{CE}$ = 5 V; $I_C$ = 10 mA | – | – | 770 | mV |
| $C_c$ | collector capacitance | $V_{CB}$ = 10 V; $I_E$ = $i_e$ = 0 A; f = 1 MHz | – | 1.5 | – | pF |
| $C_e$ | emitter capacitance | $V_{EB}$ = 0.5 V; $I_C$ = $i_c$ = 0 A; f = 1 MHz | – | 11 | – | pF |
| $f_T$ | transition frequency | $V_{CE}$ = 5 V; $I_C$ = 10mA; f = 100 MHz | 100 | – | – | MHz |
| F | noise figure | $V_{CE}$ = 5 V; $I_C$ = 200 μA; $R_S$ = 2 kΩ; f = 1 kHz; B = 200 Hz | – | 2 | 10 | dB |

**Notes**

1. $V_{BEsat}$ decreases by about 1.7 mV/K with increasing temperature.
2. $V_{BE}$ decreases by about 2 mV/K with increasing temperature.

2004 Nov 25 — 4

Abbildung 8.3: Datenblatt - Transistor BC547

19-1298; Rev 9; 9/05

# ±15kV ESD-Protected, Down to 10nA, 3.0V to 5.5V, Up to 1Mbps, True RS-232 Transceivers

MAX3222E/MAX3232E/MAX3237E/MAX3241E†/MAX3246E

## General Description

The MAX3222E/MAX3232E/MAX3237E/MAX3241E/MAX3246E +3.0V-powered EIA/TIA-232 and V.28/V.24 communications interface devices feature low power consumption, high data-rate capabilities, and enhanced electrostatic-discharge (ESD) protection. The enhanced ESD structure protects all transmitter outputs and receiver inputs to ±15kV using IEC 1000-4-2 Air-Gap Discharge, ±8kV using IEC 1000-4-2 Contact Discharge (±9kV for MAX3246E), and ±15kV using the Human Body Model. The logic and receiver I/O pins of the MAX3237E are protected to the above standards, while the transmitter output pins are protected to ±15kV using the Human Body Model.

A proprietary low-dropout transmitter output stage delivers true RS-232 performance from a +3.0V to +5.5V power supply, using an internal dual charge pump. The charge pump requires only four small 0.1µF capacitors for operation from a +3.3V supply. Each device guarantees operation at data rates of 250kbps while maintaining RS-232 output levels. The MAX3237E guarantees operation at 250kbps in the normal operating mode and 1Mbps in the MegaBaud™ operating mode, while maintaining RS-232-compliant output levels.

The MAX3222E/MAX3232E have two receivers and two transmitters. The MAX3222E features a 1µA shutdown mode that reduces power consumption in battery-powered portable systems. The MAX3222E receivers remain active in shutdown mode, allowing monitoring of external devices while consuming only 1µA of supply current. The MAX3222E and MAX3232E are pin, package, and functionally compatible with the industry-standard MAX242 and MAX232, respectively.

The MAX3241E/MAX3246E are complete serial ports (three drivers/five receivers) designed for notebook and subnotebook computers. The MAX3237E (five drivers/three receivers) is ideal for peripheral applications that require fast data transfer. These devices feature a shutdown mode in which all receivers remain active, while consuming only 1µA (MAX3241E/MAX3246E) or 10nA (MAX3237E).

The MAX3222E, MAX3232E, and MAX3241E are available in space-saving SO, SSOP, TQFN and TSSOP packages. The MAX3237E is offered in an SSOP package. The MAX3246E is offered in the ultra-small 6 x 6 UCSP™ package.

## Applications

| | |
|---|---|
| Battery-Powered Equipment | Printers |
| Cell Phones | Smart Phones |
| Cell-Phone Data Cables | xDSL Modems |
| Notebook, Subnotebook, and Palmtop Computers | |

## Next-Generation Device Features

- **For Space-Constrained Applications**
  **MAX3228E/MAX3229E: ±15kV ESD-Protected, +2.5V to +5.5V, RS-232 Transceivers in UCSP**
- **For Low-Voltage or Data Cable Applications**
  **MAX3380E/MAX3381E: +2.35V to +5.5V, 1µA, 2Tx/2Rx, RS-232 Transceivers with ±15kV ESD-Protected I/O and Logic Pins**

## Ordering Information

| PART | TEMP RANGE | PIN-PACKAGE | PKG CODE |
|---|---|---|---|
| **MAX3222E**CTP | 0°C to +70°C | 20 Thin QFN-EP** (5mm x 5mm) | T2055-5 |
| MAX3222ECUP | 0°C to +70°C | 20 TSSOP | — |
| MAX3222ECAP | 0°C to +70°C | 20 SSOP | — |
| MAX3222ECWN | 0°C to +70°C | 18 Wide SO | — |
| MAX3222ECPN | 0°C to +70°C | 18 Plastic DIP | — |
| MAX3222EC/D | 0°C to +70°C | Dice* | — |
| MAX3222EETP | -40°C to +85°C | 20 Thin QFN-EP** (5mm x 5mm) | T2055-5 |
| MAX3222EEUP | -40°C to +85°C | 20 TSSOP | — |
| MAX3222EEAP | -40°C to +85°C | 20 SSOP | — |
| MAX3222EEWN | -40°C to +85°C | 18 Wide SO | — |
| MAX3222EEPN | -40°C to +85°C | 18 Plastic DIP | — |
| **MAX3232E**CAE | 0°C to +70°C | 16 SSOP | — |
| MAX3232ECWE | 0°C to +70°C | 16 Wide SO | — |
| MAX3232ECPE | 0°C to +70°C | 16 Plastic DIP | — |

*Dice are tested at $T_A$ = +25°C, DC parameters only.*

***EP = Exposed paddle.*

***Ordering Information continued at end of data sheet.***

***Pin Configurations appear at end of data sheet.***
***Selector Guide appears at end of data sheet.***
***Typical Operating Circuits appear at end of data sheet.***

*MegaBaud and UCSP are trademarks of Maxim Integrated Products, Inc.*

*†Covered by U.S. Patent numbers 4,636,930; 4,679,134; 4,777,577; 4,797,899; 4,809,152; 4,897,774; 4,999,761; and other patents pending.*

MAXIM ____ ***Maxim Integrated Products*** **1**

***For pricing, delivery, and ordering information, please contact Maxim/Dallas Direct! at 1-888-629-4642, or visit Maxim's website at www.maxim-ic.com.***

Abbildung 8.4: Datenblatt - RS232 Teil 1

## ±15kV ESD-Protected, Down to 10nA, 3.0V to 5.5V, Up to 1Mbps, True RS-232 Transceivers

### ABSOLUTE MAXIMUM RATINGS

$V_{CC}$ to GND ....................................................-0.3V to +6V
V+ to GND (Note 1) ..............................................-0.3V to +7V
V- to GND (Note 1) ..............................................+0.3V to -7V
V+ + |V-| (Note 1)..................................................+13V
Input Voltages
  T_IN, $\overline{EN}$, $\overline{SHDN}$, MBAUD to GND ......................-0.3V to +6V
  R_IN to GND ..........................................................±25V
Output Voltages
  T_OUT to GND.......................................................±13.2V
  R_OUT, R_OUTB (MAX3241E)................-0.3V to ($V_{CC}$ + 0.3V)
Short-Circuit Duration, T_OUT to GND......................Continuous
Continuous Power Dissipation ($T_A$ = +70°C)
  16-Pin SSOP (derate 7.14mW/°C above +70°C) ..........571mW
  16-Pin TSSOP (derate 9.4mW/°C above +70°C) .......754.7mW
  16-Pin TQFN (derate 20.8mW/°C above +70°C) .....1666.7mW
  16-Pin Wide SO (derate 9.52mW/°C above +70°C) .....762mW
  18-Pin Wide SO (derate 9.52mW/°C above +70°C) .....762mW
  18-Pin PDIP (derate 11.11mW/°C above +70°C)..........889mW
  20-Pin TQFN (derate 21.3mW/°C above +70°C) ........1702mW
  20-Pin TSSOP (derate 10.9mW/°C above +70°C) ........879mW
  20-Pin SSOP (derate 8.00mW/°C above +70°C) ..........640mW
  28-Pin SSOP (derate 9.52mW/°C above +70°C) ..........762mW
  28-Pin Wide SO (derate 12.50mW/°C above +70°C).............1W
  28-Pin TSSOP (derate 12.8mW/°C above +70°C) ......1026mW
  32-Lead Thin QFN (derate 33.3mW/°C above +70°C)..2666mW
  6 x 6 UCSP (derate 12.6mW/°C above +70°C)............1010mW
Operating Temperature Ranges
  MAX32_ _EC_ _ ..............................................0°C to +70°C
  MAX32_ _EE_ _..............................................-40°C to +85°C
Storage Temperature Range .............................-65°C to +150°C
Lead Temperature (soldering, 10s) ................................+300°C
Bump Reflow Temperature (Note 2)
  Infrared, 15s.........................................................+200°C
  Vapor Phase, 20s...................................................+215°C

**Note 1:** V+ and V- can have maximum magnitudes of 7V, but their absolute difference cannot exceed 13V.

**Note 2:** This device is constructed using a unique set of packaging techniques that impose a limit on the thermal profile the device can be exposed to during board-level solder attach and rework. This limit permits only the use of the solder profiles recommended in the industry-standard specification, JEDEC 020A, paragraph 7.6, Table 3 for IR/VPR and convection reflow. Preheating is required. Hand or wave soldering is not allowed.

*Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

### ELECTRICAL CHARACTERISTICS

($V_{CC}$ = +3V to +5.5V, C1–C4 = 0.1µF, $T_A$ = $T_{MIN}$ to $T_{MAX}$, unless otherwise noted. Typical values are at $T_A$ = +25°C.) (Notes 3, 4)

| PARAMETER | CONDITIONS | | MIN | TYP | MAX | UNITS |
|---|---|---|---|---|---|---|
| **DC CHARACTERISTICS** ($V_{CC}$ = +3.3V or +5V, $T_A$ = +25°C) | | | | | | |
| Supply Current | $\overline{SHDN}$ = $V_{CC}$, no load | MAX3222E, MAX3232E, MAX3241E, MAX3246E | | 0.3 | 1 | mA |
| | | MAX3237E | | 0.5 | 2.0 | |
| Shutdown Supply Current | $\overline{SHDN}$ = GND | | | 1 | 10 | µA |
| | $\overline{SHDN}$ = R_IN = GND, T_IN = GND or $V_{CC}$ (MAX3237E) | | | 10 | 300 | nA |
| **LOGIC INPUTS** | | | | | | |
| Input Logic Low | T_IN, $\overline{EN}$, $\overline{SHDN}$, MBAUD | | | | 0.8 | V |
| Input Logic High | T_IN, $\overline{EN}$, $\overline{SHDN}$, MBAUD | $V_{CC}$ = +3.3V | 2.0 | | | V |
| | | $V_{CC}$ = +5.0V | 2.4 | | | |
| Transmitter Input Hysteresis | | | | 0.5 | | V |
| Input Leakage Current | T_IN, $\overline{EN}$, $\overline{SHDN}$ | MAX3222E, MAX3232E, MAX3241E, MAX3246E | | ±0.01 | ±1 | µA |
| | T_IN, $\overline{SHDN}$, MBAUD | MAX3237E (Note 5) | | 9 | 18 | |
| **RECEIVER OUTPUTS** | | | | | | |
| Output Leakage Current | R_OUT (MAX3222E/MAX3237E/MAX3241E/MAX3246E), $\overline{EN}$ = $V_{CC}$, receivers disabled | | | ±0.05 | ±10 | µA |
| Output Voltage Low | $I_{OUT}$ = 1.6mA (MAX3222E/MAX3232E/MAX3241E/MAX3246E), $I_{OUT}$ = 1.0mA (MAX3237E) | | | | 0.4 | V |



Abbildung 8.5: Datenblatt - RS232 Teil 2

## ±15kV ESD-Protected, Down to 10nA, 3.0V to 5.5V, Up to 1Mbps, True RS-232 Transceivers



### ELECTRICAL CHARACTERISTICS (continued)

($V_{CC}$ = +3V to +5.5V, C1–C4 = 0.1µF, $T_A$ = $T_{MIN}$ to $T_{MAX}$, unless otherwise noted. Typical values are at $T_A$ = +25°C.) (Notes 3, 4)

| PARAMETER | CONDITIONS | | MIN | TYP | MAX | UNITS |
|---|---|---|---|---|---|---|
| Output Voltage High | IOUT = -1.0mA | | $V_{CC}$ - 0.6 | $V_{CC}$ - 0.1 | | V |
| **RECEIVER INPUTS** | | | | | | |
| Input Voltage Range | | | -25 | | +25 | V |
| Input Threshold Low | $T_A$ = +25°C | $V_{CC}$ = +3.3V | 0.6 | 1.1 | | V |
| | | $V_{CC}$ = +5.0V | 0.8 | 1.5 | | |
| Input Threshold High | $T_A$ = +25°C | $V_{CC}$ = +3.3V | | 1.5 | 2.4 | V |
| | | $V_{CC}$ = +5.0V | | 2.0 | 2.4 | |
| Input Hysteresis | | | | 0.5 | | V |
| Input Resistance | $T_A$ = +25°C | | 3 | 5 | 7 | kΩ |
| **TRANSMITTER OUTPUTS** | | | | | | |
| Output Voltage Swing | All transmitter outputs loaded with 3kΩ to ground | | ±5 | ±5.4 | | V |
| Output Resistance | $V_{CC}$ = 0, transmitter output = ±2V | | 300 | 50k | | Ω |
| Output Short-Circuit Current | | | | | ±60 | mA |
| Output Leakage Current | $V_{CC}$ = 0 or +3.0V to +5.5V, $V_{OUT}$ = ±12V, transmitters disabled (MAX3222E/MAX3232E/MAX3241E/MAX3246E) | | | | ±25 | µA |
| **MOUSE DRIVABILITY** (MAX3241E) | | | | | | |
| Transmitter Output Voltage | T1IN = T2IN = GND, T3IN = $V_{CC}$, T3OUT loaded with 3kΩ to GND, T1OUT and T2OUT loaded with 2.5mA each | | ±5 | | | V |
| **ESD PROTECTION** | | | | | | |
| R_IN, T_OUT | Human Body Model | | | ±15 | | kV |
| | IEC 1000-4-2 Air-Gap Discharge (except MAX3237E) | | | ±15 | | |
| | IEC 1000-4-2 Contact Discharge (except MAX3237E) | | | ±8 | | |
| | IEC 1000-4-2 Contact Discharge (MAX3246E only) | | | ±9 | | |
| T_IN, R_IN, R_OUT, $\overline{EN}$, $\overline{SHDN}$, MBAUD | MAX3237E | Human Body Model | | ±15 | | kV |
| | | IEC1000-4-2 Air-Gap Discharge | | ±15 | | |
| | | IEC1000-4-2 Contact Discharge | | ±8 | | |



Abbildung 8.6: Datenblatt - RS232 Teil 3

XILINX®

# Platform Flash In-System Programmable Configuration PROMs

DS123 (v2.16) November 14, 2008 **Product Specification**

## Features

- In-System Programmable PROMs for Configuration of Xilinx® FPGAs
- Low-Power Advanced CMOS NOR Flash Process
- Endurance of 20,000 Program/Erase Cycles
- Operation over Full Industrial Temperature Range (–40°C to +85°C)
- IEEE Standard 1149.1/1532 Boundary-Scan (JTAG) Support for Programming, Prototyping, and Testing
- JTAG Command Initiation of Standard FPGA Configuration
- Cascadable for Storing Longer or Multiple Bitstreams
- Dedicated Boundary-Scan (JTAG) I/O Power Supply ($V_{CCJ}$)
- I/O Pins Compatible with Voltage Levels Ranging From 1.5V to 3.3V
- Design Support Using the Xilinx ISE® Alliance and Foundation™ Software Packages
- XCF01S/XCF02S/XCF04S
  - 3.3V Supply Voltage
  - Serial FPGA Configuration Interface (up to 33 MHz)
  - Available in Small-Footprint VO20 and VOG20 Packages
- XCF08P/XCF16P/XCF32P
  - 1.8V Supply Voltage
  - Serial or Parallel FPGA Configuration Interface (up to 33 MHz)
  - Available in Small-Footprint VO48, VOG48, FS48, and FSG48 Packages
  - Design Revision Technology Enables Storing and Accessing Multiple Design Revisions for Configuration
  - Built-In Data Decompressor Compatible with Xilinx Advanced Compression Technology

## Description

Xilinx introduces the Platform Flash series of in-system programmable configuration PROMs. Available in 1 to 32 Mb densities, these PROMs provide an easy-to-use, cost-effective, and reprogrammable method for storing large Xilinx FPGA configuration bitstreams. The Platform Flash PROM series includes both the 3.3V XCFxxS PROM and the 1.8V XCFxxP PROM. The XCFxxS version includes 4 Mb, 2 Mb, and 1 Mb PROMs that support Master Serial and Slave Serial FPGA configuration modes (Figure 1, page 2). The XCFxxP version includes 32 Mb, 16 Mb, and 8 Mb PROMs that support Master Serial, Slave Serial, Master SelectMAP, and Slave SelectMAP FPGA configuration modes (Figure 2, page 2). A summary of the Platform Flash PROM family members and supported features is shown in Table 1.

*Table 1:* **Platform Flash PROM Features**

| Device | Density (Mb) | $V_{CCINT}$ (V) | $V_{CCO}$ Range (V) | $V_{CCJ}$ Range (V) | Packages | Program In-system via JTAG | Serial Config. | Parallel Config. | Design Revisioning | Compression |
|---|---|---|---|---|---|---|---|---|---|---|
| XCF01S | 1 | 3.3 | 1.8 – 3.3 | 2.5 – 3.3 | VO20/VOG20 | ✓ | ✓ | | | |
| XCF02S | 2 | 3.3 | 1.8 – 3.3 | 2.5 – 3.3 | VO20/VOG20 | ✓ | ✓ | | | |
| XCF04S | 4 | 3.3 | 1.8 – 3.3 | 2.5 – 3.3 | VO20/VOG20 | ✓ | ✓ | | | |
| XCF08P | 8 | 1.8 | 1.5 – 3.3 | 2.5 – 3.3 | VO48/VOG48 FS48/FSG48 | ✓ | ✓ | ✓ | ✓ | ✓ |
| XCF16P | 16 | 1.8 | 1.5 – 3.3 | 2.5 – 3.3 | VO48/VOG48 FS48/FSG48 | ✓ | ✓ | ✓ | ✓ | ✓ |
| XCF32P | 32 | 1.8 | 1.5 – 3.3 | 2.5 – 3.3 | VO48/VOG48 FS48/FSG48 | ✓ | ✓ | ✓ | ✓ | ✓ |

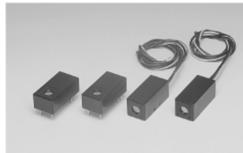
© 2003–2008 Xilinx, Inc. All rights reserved. XILINX, the Xilinx logo, Virtex, Spartan, ISE and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.



Abbildung 8.7: Datenblatt - Xilinx Config PROM

# Metal Package PMT

## Photosensor Modules H5773/H5783/H6779/H6780 Series



The H5773/H5783/H6779/H6780 series are photosensor modules housing a metal package PMT and high-voltage power supply circuit. The metal package PMTs have a metallic package with the same diameter as a TO-8 package used for semiconductor photodetectors, and deliver high gain, wide dynamic range and high-speed response while maintaining small dimensions identical to those of photodiodes. The internal high-voltage power supply circuit is also compact, making the module easy to use.
Considering the mounting methods, a cable output type and a pin output type are provided, and a total of 7 types are available according to the wavelength range to be measured. A P-type is also available with selected gain and dark count ideal for photon counting under extremely low light conditions.

### Product Variations

| Type No. \ Suffix | None | -01 | -02 | -03 | -04 | -06 | -20 | Output Type | Features |
|---|---|---|---|---|---|---|---|---|---|
| H5773 | yes | yes | yes | yes | yes | yes | yes | On-board | Low power consumption |
| H5783 | yes | yes | yes | yes | yes | yes | yes | Cable output | |
| H5773P | yes | no | no | no | no | no | no | On-board | For photon counting |
| H5783P | yes | no | no | no | no | no | no | Cable output | Low power consumption |
| H6779 | yes | yes | yes | yes | yes | yes | yes | On-board | Low ripple noise |
| H6780 | yes | yes | yes | yes | yes | yes | yes | Cable output | Fast settling time |

| Suffix | Spectral Response |
|---|---|
| None | 300 nm to 650 nm |
| -01 | 300 nm to 850 nm |
| -02 | 300 nm to 880 nm |
| -03 | 185 nm to 650 nm |
| -04 | 185 nm to 850 nm |
| -06 | 185 nm to 650 nm |
| -20 | 300 nm to 900 nm |

The suffix -06 type (synthetic silica window) has higher sensitivity than the -03 type below 300 nm in wavelength range.

### Specifications

| Parameter | | | | H5773 / H5783 / H6779 / H6780 Series | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|
| Suffix | | | | None | -03, -06 | -01, -04 | -02 | -20 | — |
| Input Voltage | | | | +11.5 to +15.5 | | | | | V |
| Max. Input Voltage | | | | +18 | | | | | V |
| Max. Input Current | | | | H5773 / H5783 Series: 9<br>H6779 / H6780 Series: 30 | | | | | mA |
| Max. Output Signal Current | | | | 100 | | | | | μA |
| Max. Control Voltage | | | | +1.0 (Input impedance 100 kΩ) | | | | | V |
| Recommended Control Voltage Adjustment Range | | | | +0.25 to +0.9 | | | | | V |
| Effective Area | | | | φ8 | | | | | mm |
| Sensitivity Adjustment Range | | | | 1: $10^4$ | | | | | — |
| Peak Sensitivity Wavelength | | | | 420 | 420 | 400 | 500 | 630 | nm |
| Cathode | Luminous Sensitivity | | Min. | 40 | 40 | 80 | 200 | 350 | μA/lm |
| | | | Typ. | 70 | 70 | 150 | 250 | 500 | |
| | Blue Sensitivity Index (CS 5-58) | | | 8 | 8 | — | — | — | — |
| | Red/White Ratio | | | — | — | 0.2 | 0.25 | 0.45 | — |
| | Radiant Sensitivity *1 | | | 62 | 62 | 60 | 58 | 78 | mA/W |
| Anode | Standard Type | Luminous Sensitivity | Min. | 10 | 10 | 15 | 25 | 35 | A/lm |
| | | | Typ. | 50 | 50 | 75 | 125 | 250 | |
| | | Radiant Sensitivity *1 *2 | | $4.3 \times 10^4$ | $4.3 \times 10^4$ | $3.0 \times 10^4$ | $2.9 \times 10^4$ | $3.9 \times 10^4$ | A/W |
| | | Dark Current *2 *3 | Typ. | 0.2 | 0.2 | 0.4 | 2 | 2 | nA |
| | | | Max. | 2 | 2 | 4 | 20 | 20 | |
| | P Type | Gain *2 | Min. | $7.5 \times 10^5$ | — | | | | — |
| | | | Typ. | $1 \times 10^6$ | — | | | | |
| | | Radiant Sensitivity *1 *2 | | $6.2 \times 10^4$ | — | | | | A/W |
| | | Dark Count *2 *3 | Typ. | 80 | — | | | | $s^{-1}$ |
| | | | Max. | 400 | — | | | | |
| Rise Time *2 | | | | 0.78 | | | | | ns |

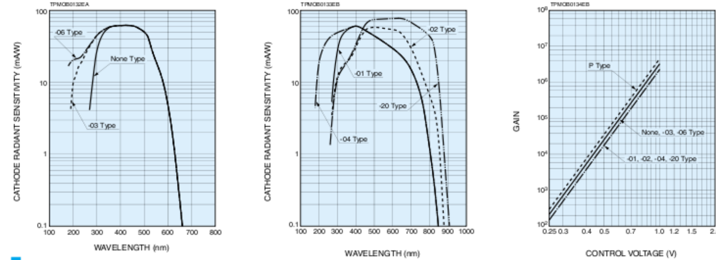
| Parameter | | H5773 Series | H5783 Series | H6779 Series | H6780 Series | Unit |
|---|---|---|---|---|---|---|
| Ripple Noise *2 *4 (peak to peak) | Max. | 1.2 | | 0.6 | | mV |
| Settling Time *5 | | 2 | | 0.2 | | s |
| Operating Ambient Temperature | | +5 to +50 | | +5 to +45 | | °C |
| Storage Temperature | | -20 to +50 | | | | °C |
| Weight | | 60 | 80 | 60 | 80 | g |

*1: Measured at the peak sensitivity wavelength *2: Control voltage = +0.8 V *3: After 30 minute storage in darkness
*4: Cable RG-174/U, Cable length 450 mm, Load resistance = 1 MΩ, Load capacitance = 22 pF
*5: The time required for the output to reach a stable level following a change in the control voltage from +1.0 V to +0.5 V.

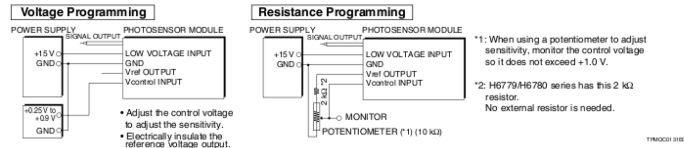
16

Abbildung 8.8: Datenblatt - PMT Teil 1

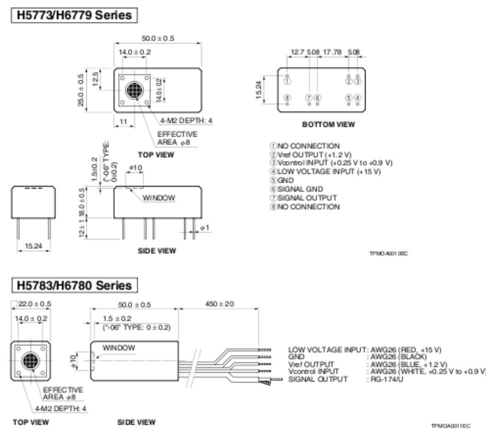
## Characteristics (Cathode radiant sensitivity, Gain)



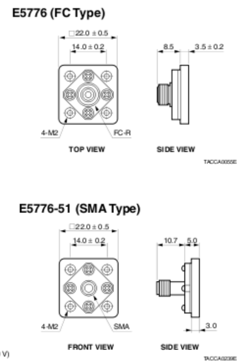

## Sensitivity Adjustment Method




## Dimensional Outlines (Unit: mm)




## Option (Optical Fiber Adapter) (Unit: mm)




17

Abbildung 8.9: Datenblatt - PMT Teil 2

## ELECTRICAL CHARACTERISTICS

**The q denotes the specifications which apply over the full operating temperature range, otherwise specifications are at $T_A = 25°C$.**

| PARAMETER | CONDITIONS | | MIN | TYP | MAX | UNITS |
|---|---|---|---|---|---|---|
| Dropout Voltage | $I_{LOAD} = -1mA$ | | | 0.1 | 0.15 | V |
| $V_{IN} = V_{OUT(NOMINAL)}$ | $I_{LOAD} = -1mA$ | q | | | 0.19 | V |
| (Notes 4, 5) | $I_{LOAD} = -10mA$ | | | 0.15 | 0.20 | V |
| | $I_{LOAD} = -10mA$ | q | | | 0.25 | V |
| | $I_{LOAD} = -100mA$ | | | 0.26 | 0.33 | V |
| | $I_{LOAD} = -100mA$ | q | | | 0.39 | V |
| | $I_{LOAD} = -200mA$ | | | 0.34 | 0.42 | V |
| | $I_{LOAD} = -200mA$ | q | | | 0.49 | V |
| GND Pin Current | $I_{LOAD} = 0mA$ | q | | 30 | 70 | μA |
| $V_{IN} = V_{OUT(NOMINAL)}$ | $I_{LOAD} = -1mA$ | q | | 85 | 180 | μA |
| (Notes 4, 6) | $I_{LOAD} = -10mA$ | q | | 300 | 600 | μA |
| | $I_{LOAD} = -100mA$ | q | | 1.3 | 3 | mA |
| | $I_{LOAD} = -200mA$ | q | | 2.5 | 6 | mA |
| Output Voltage Noise | $C_{OUT} = 10μF$, $C_{BYP} = 0.01μF$, $I_{LOAD} = -200mA$, BW = 10Hz to 100kHz | | | 30 | | $μV_{RMS}$ |
| ADJ Pin Bias Current | (Notes 2, 7) | | | 30 | 100 | nA |
| Minimum Input Voltage (Note 12) | LT1964-BYP | q | | −1.9 | −2.8 | V |
| $I_{LOAD} = -200mA$ | LT1964-SD | q | | −1.6 | −2.2 | V |
| Shutdown Threshold | $V_{OUT}$ = Off to On (Positive) | q | | 1.6 | 2.1 | V |
| | $V_{OUT}$ = Off to On (Negative) | q | | −1.9 | −2.8 | V |
| | $V_{OUT}$ = On to Off (Positive) | q | 0.25 | 0.8 | | V |
| | $V_{OUT}$ = On to Off (Negative) | q | −0.25 | −0.8 | | V |
| $\overline{SHDN}$ Pin Current (Note 8) | $V_{\overline{SHDN}} = 0V$ | | −1 | ±0.1 | 1 | μA |
| | $V_{\overline{SHDN}} = 15V$ | | | 6 | 15 | μA |
| | $V_{\overline{SHDN}} = -15V$ | | | −3 | −9 | μA |
| Quiescent Current in Shutdown | $V_{IN} = -6V$, $V_{\overline{SHDN}} = 0V$ | q | | 3 | 10 | μA |
| Ripple Rejection | $V_{IN} - V_{OUT} = -1.5V(Avg)$, $V_{RIPPLE} = 0.5V_{P-P}$, $f_{RIPPLE} = 120Hz$, $I_{LOAD} = -200mA$ | | 46 | 54 | | dB |
| Current Limit | $V_{IN} = -6V$, $V_{OUT} = 0V$ | | | 350 | | mA |
| | $V_{IN} = V_{OUT(NOMINAL)} - 1.5V$, $\Delta V_{OUT} = 0.1V$ | q | 220 | | | mA |
| Input Reverse Leakage Current | $V_{IN} = 20V$, $V_{OUT}$, $V_{ADJ}$, $V_{\overline{SHDN}}$ = Open Circuit | q | | | 1 | mA |

**Note 1:** Absolute Maximum Ratings are those values beyond which the life of a device may be impaired.

**Note 2:** The LT1964 (adjustable version) is tested and specified for these conditions with the ADJ pin connected to the OUT pin.

**Note 3:** Operating conditions are limited by maximum junction temperature. The regulated output voltage specification will not apply for all possible combinations of input voltage and output current. When operating at maximum input voltage, the output current range must be limited. When operating at maximum output current, the input voltage range must be limited.

**Note 4:** To satisfy requirements for minimum input voltage, the LT1964 (adjustable version) is tested and specified for these conditions with an external resistor divider (two 249k resistors) for an output voltage of −2.44V. The external resistor divider will add a 5μA DC load on the output.

**Note 5:** Dropout voltage is the minimum input to output voltage differential needed to maintain regulation at a specified output current. In dropout, the output voltage will be equal to: $(V_{IN} + V_{DROPOUT})$.

**Note 6:** GND pin current is tested with $V_{IN} = V_{OUT(NOMINAL)}$ and a current source load. This means the device is tested while operating in its dropout region. This is the worst-case GND pin current. The GND pin current will decrease slightly at higher input voltages.

**Note 7:** ADJ pin bias current flows out of the ADJ pin.

**Note 8:** Positive $\overline{SHDN}$ pin current flows into the $\overline{SHDN}$ pin. $\overline{SHDN}$ pin current is included in the GND pin current specification.

**Note 9:** For input-to-output differential voltages greater than 7V, a 50μA load is needed to maintain regulation.

**Note 10:** The LT1964E is guaranteed to meet performance specifications from 0°C to 125°C. Specifications over the −40°C to 125°C operating junction temperature range are assured by design, characterization and correlation with statistical process controls.

**Note 11:** A parasitic diode exists internally on the LT1964 between the OUT, ADJ and $\overline{SHDN}$ pins and the IN pin. The OUT, ADJ and $\overline{SHDN}$ pins cannot be pulled more than 0.5V more negative than the IN pin during fault conditions, and must remain at a voltage more positive than the IN pin during operation.

**Note 12:** For the LT1964-BYP, this specification accounts for the operating threshold of the $\overline{SHDN}$ pin, which is tied to the IN pin internally. For the LT1964-SD, the $\overline{SHDN}$ threshold must be met to ensure device operation.

**Note 13:** Actual thermal resistance ($\theta_{JA}$) junction to ambient will be a function of board layout. Junction-to-case thermal resistance ($\theta_{JC}$) measured at Pin 2 is 60°C/W. See the Thermal Considerations section in the Applications Information.

1964f

Abbildung 8.10: Datenblatt - Spannungsregler LT1964

# CFPS-72, -73 SMD CLOCK OSCILLATORS

CLOCK OSCILLATORS

**ISSUE 11; 1 NOVEMBER 2008 - RoHS 2002/95/EC**

**Description**

- Surface mount oscillators in a ceramic package with a hermetically sealed metal lid available in two voltages

**Fast Make Capability**

- Please see CFPP-72, 73 series Programmable Oscillators for nearest equivalent fast make parts

**Package Outline**

- 7 x 5mm SMD ceramic package

**Frequency Range**

- 1.25 to 160MHz

**Output Compatibility & Load**

- Tri-state HCMOS/TTL (5.0V) (CFPS-72)
- Tri-state HCMOS (3.3V) (CFPS-73)

| Maximum Capacitive Load | |
|---|---|
| 1.5MHz to 50MHz | 50pF max |
| >50MHz to 80MHz | 30pF max |
| >80MHz to 160MHz | 15pF max |

**Frequency Stabilities**

- ±20ppm, ±25ppm, ±50ppm, ±100ppm (inclusive of supply voltage and output load variations over the operating temperature range)

**Operating Temperature Range**

- 0 to 70°C (CFPS-72, -73)
- –40 to 85°C (CFPS-72I, -73I)

**Storage Temperature Range**

- –55 to 125°C

**Tri-state Operation**

- Logic '1' (2.2V min) to pad 1 enables oscillator output
- Logic '0' to pad 1 (0.8V max) disables oscillator output; when disabled the oscillator output goes to the high impedance state
- No connection to pad 1 enables oscillator output

**Marking Includes**

- Model Number + Frequency Stability Code + Frequency

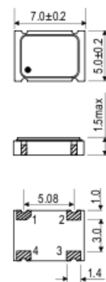
**Packaging**

- Bulk or Tape and Reel

**Minimum Order Information Required**

- Frequency + Model Number + Operating Temperature Code (if applicable) + Frequency Stability

**Outline (mm)**

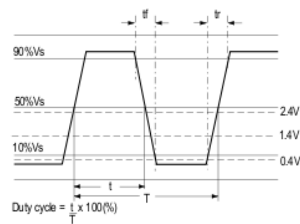



Pad Connections
1. N/C or Enable/Disable
2. GND
3. Output
4. +Vs

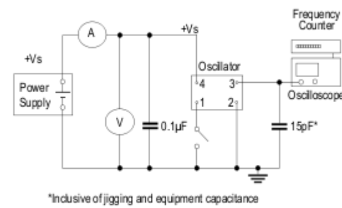
Solder pad layout

1.8 3.28 2.0 2.2

**Output Waveform**




**Test Circuit**




*Inclusive of jigging and equipment capacitance

Tel: +44 (0)1460 270200 122 www.iqdfrequencyproducts.com

Abbildung 8.11: Datenblatt - Oszillator

National Semiconductor

June 2005

# LM1086

# 1.5A Low Dropout Positive Regulators

LM1086 1.5A Low Dropout Positive Regulators

## General Description

The LM1086 is a series of low dropout positive voltage regulators with a maximum dropout of 1.5V at 1.5A of load current. It has the same pin-out as National Semiconductor's industry standard LM317.

The LM1086 is available in an adjustable version, which can set the output voltage with only two external resistors. It is also available in six fixed voltages: 1.8V, 2.5V, 2.85V, 3.3V, 3.45V and 5.0V. The fixed versions integrate the adjust resistors.

The LM1086 circuit includes a zener trimmed bandgap reference, current limiting and thermal shutdown.

The LM1086 series is available in TO-220, TO-263, and LLP packages. Refer to the LM1084 for the 5A version, and the LM1085 for the 3A version.

## Features

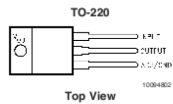
- Available in 1.8V, 2.5V, 2.85V, 3.3V, 3.45V, 5V and Adjustable Versions
- Current Limiting and Thermal Protection
- Output Current 1.5A
- Line Regulation 0.015% (typical)
- Load Regulation 0.1% (typical)

## Applications

- SCSI-2 Active Terminator
- High Efficiency Linear Regulators
- Battery Charger
- Post Regulation for Switching Supplies
- Constant Current Regulator
- Microprocessor Supply

## Connection Diagrams

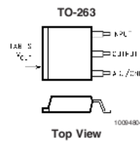
**TO-220**



10094802

**Top View**

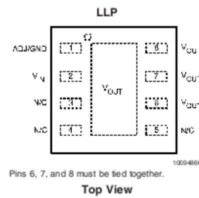
**TO-263**



10094804

**Top View**

**LLP**

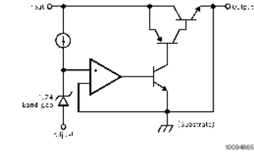


10094866

Pins 6, 7, and 8 must be tied together.

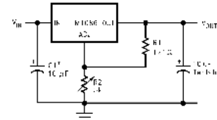
**Top View**

**Basic Functional Diagram, Adjustable Version**



10094865

**Application Circuit**



10094852

**1.2V to 15V Adjustable Regulator**

© 2005 National Semiconductor Corporation DS100948 www.national.com

Abbildung 8.12: Datenblatt - Spannungsregler LM1086

LT1964

## ABSOLUTE MAXIMUM RATINGS (Note 1)

IN Pin Voltage ........................................................ ±20V
OUT Pin Voltage (Note 11) .................................... ±20V
OUT to IN Differential Voltage (Note 11) ....... −0.5V, 20V
ADJ Pin Voltage
(with Respect to IN Pin) (Note 11) ........... −0.5V, 20V
BYP Pin Voltage
(with Respect to IN Pin) .................................. ±20V
$\overline{SHDN}$ Pin Voltage
(with Respect to IN Pin) (Note 11) ........... −0.5V, 35V
$\overline{SHDN}$ Pin Voltage
(with Respect to GND Pin) ........................ −20V, 15V
Output Short-Circuit Duration .......................... Indefinite
Operating Junction Temperature
Range (Note 10) ............................... −40°C to 125°C
Storage Temperature Range ................. −65°C to 150°C
Lead Temperature (Soldering, 10 sec) .................. 300°C

## PACKAGE/ORDER INFORMATION

| Package | ORDER PART NUMBER | S5 PART MARKING |
|---|---|---|
| TOP VIEW: GND 1, IN 2, $\overline{SHDN}$ 3, 4 ADJ, 5 OUT. S5 PACKAGE 5-LEAD PLASTIC SOT-23. $T_{JMAX}$ = 150°C, $\theta_{JA}$ ≈125°C/W to 250°C/W (NOTE 13) SEE THE APPLICATIONS INFORMATION SECTION | LT1964ES5-SD | LTVX |
| TOP VIEW: GND 1, IN 2, BYP 3, 4 ADJ, 5 OUT. S5 PACKAGE 5-LEAD PLASTIC SOT-23. $T_{JMAX}$ = 150°C, $\theta_{JA}$ ≈125°C/W to 250°C/W (NOTE 13) SEE THE APPLICATIONS INFORMATION SECTION | LT1964ES5-BYP | LTVY |
| TOP VIEW: GND 1, IN 2, BYP 3, 4 $\overline{SHDN}$, 5 OUT. S5 PACKAGE 5-LEAD PLASTIC SOT-23. $T_{JMAX}$ = 150°C, $\theta_{JA}$ ≈125°C/W to 250°C/W (NOTE 13) SEE THE APPLICATIONS INFORMATION SECTION | LT1964ES5-5 | LTVZ |

Consult LTC Marketing for parts specified with wider operating temperature ranges.

## ELECTRICAL CHARACTERISTICS

**The q denotes the specifications which apply over the full operating temperature range, otherwise specifications are at $T_A$ = 25°C.**

| PARAMETER | CONDITIONS | | | MIN | TYP | MAX | UNITS |
|---|---|---|---|---|---|---|---|
| Regulated Output Voltage (Notes 3, 9) | LT1964-5 | $V_{IN} = -5.5V$, $I_{LOAD} = -1mA$ | | −4.925 | −5 | −5.075 | V |
| | | $-20V < V_{IN} < -6V$, $-200mA < I_{LOAD} < -1mA$ | q | −4.850 | −5 | −5.150 | V |
| ADJ Pin Voltage (Notes 2, 3, 9) | LT1964 | $V_{IN} = -2V$, $I_{LOAD} = -1mA$ | | −1.202 | −1.22 | −1.238 | V |
| | | $-20V < V_{IN} < -2.8V$, $-200mA < I_{LOAD} < -1mA$ | q | −1.184 | −1.22 | −1.256 | V |
| Line Regulation | LT1964-5 | $\Delta V_{IN} = -5.5V$ to $-20V$, $I_{LOAD} = -1mA$ | q | | 15 | 50 | mV |
| | LT1964 (Note 2) | $\Delta V_{IN} = -2.8V$ to $-20V$, $I_{LOAD} = -1mA$ | q | | 1 | 12 | mV |
| Load Regulation | LT1964-5 | $V_{IN} = -6V$, $\Delta I_{LOAD} = -1mA$ to $-200mA$ | | | 15 | 35 | mV |
| | | $V_{IN} = -6V$, $\Delta I_{LOAD} = -1mA$ to $-200mA$ | q | | | 50 | mV |
| | LT1964 | $V_{IN} = -2.8V$, $\Delta I_{LOAD} = -1mA$ to $-200mA$ | | | 2 | 7 | mV |
| | | $V_{IN} = -2.8V$, $\Delta I_{LOAD} = -1mA$ to $-200mA$ | q | | | 15 | mV |

1964f

2

Abbildung 8.13: Datenblatt - Spannungsregler LT1964

August 2002



National Semiconductor

# LM1084
# 5A Low Dropout Positive Regulators

## General Description

The LM1084 is a series of low dropout voltage positive regulators with a maximum dropout of 1.5V at 5A of load current. It has the same pin-out as National Semiconductor's industry standard LM317.

The LM1084 is available in an adjustable version, which can set the output voltage with only two external resistors. It is also available in three fixed voltages: 3.3V, 5.0V and 12.0V. The fixed versions intergrate the adjust resistors.

The LM1084 circuit includes a zener trimmed bandgap reference, current limiting and thermal shutdown.

The LM1084 series is available in TO-220 and TO-263 packages. Refer to the LM1085 for the 3A version, and the LM1086 for the 1.5A version.

## Features

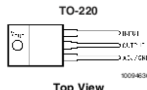
- Available in 3.3V, 5.0V, 12V and Adjustable Versions
- Current Limiting and Thermal Protection
- Output Current 5A
- Industrial Temperature Range –40˚C to 125˚C
- Line Regulation 0.015% (typical)
- Load Regulation 0.1% (typical)

## Applications

- Post Regulator for Switching DC/DC Conveter
- High Efficiency Linear Regulators
- Battery Charger

## Connection Diagrams

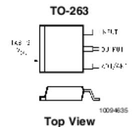
**TO-220**



10094636

**Top View**

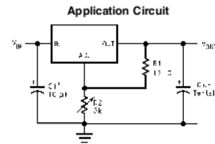
**TO-263**



10094635

**Top View**

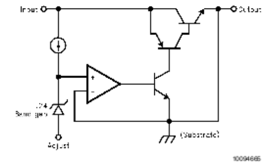
**Application Circuit**



10094652

**1.2V to 15V Adjustable Regulator**

**Basic Functional Diagram, Adjustable Version**



10094666

© 2002 National Semiconductor Corporation DS100946 www.national.com

Abbildung 8.14: Datenblatt - Spannungsregler LM1084

## The 81150A Pulse Function Arbitrary Noise Generator at a Glance

- 1 µHz – 120 MHz pulse generation with variable rise/fall time
- 1 µHz – 240 MHz sine waveform output
- 14-bit, 2 GSa/s arbitrary waveforms
- 512k samples deep arbitrary waveform memory per channel
- Pulse, sine, square, ramp, noise and arbitrary waveforms
- Noise, with an adjustable crest factor, and signal repetition time of 26 days
- FM, AM, PM, PWM, FSK modulation capabilities
- 1 or 2 channel, coupled and uncoupled
- Differential outputs
- Two selectable output amplifiers:
  - High bandwidth amplifier
    Amplitude: 50 mVpp to 5 Vpp; 50 Ω into 50 Ω
    100 mVpp to 10 Vpp; 50 Ω into open

    Voltage window: ± 5 V; 50 Ω into 50 Ω
    ± 10 V; 50 Ω into open
    ± 9 V; 5 Ω into 50 Ω
  - High voltage amplifier
    Amplitude: 100 mVpp to 10 Vpp; 50 Ω into 50 Ω, 200 m pp to 20 Vpp;
    5 Ω into 50 Ω, or 50 Ω into open

    Voltage window: ± 10 V; 50 Ω into 50 Ω
    ± 20 V; 5 Ω into 50 Ω or 50 Ω into open
- Glitch free change of timing, parameters
  (delay, frequency, transition time, width, duty cycle)
- Programming language compatible with Agilent 81101A, 81104A, 81105A
- ISO 17025 and Z540 calibration
- LXI class C compliant

2 Agilent 81150A Pulse Function Arbitrary Noise Generator
Data Sheet

Abbildung 8.15: Datenblatt - Funktionsgenerator

Nichia STS-DA1-0424
<Cat.No.081104>

1.SPECIFICATIONS

(1) Absolute Maximum Ratings (Ta=25°C)

| Item | Symbol | Absolute Maximum Rating | Unit |
|---|---|---|---|
| Forward Current | $I_F$ | 30 | mA |
| Pulse Forward Current | $I_{FP}$ | 100 | mA |
| Reverse Voltage | $V_R$ | 5 | V |
| Power Dissipation | $P_D$ | 120 | mW |
| Operating Temperature | $T_{opr}$ | -30 ~ + 85 | °C |
| Storage Temperature | $T_{stg}$ | -40 ~ +100 | °C |
| Soldering Temperature | $T_{sld}$ | 265°C for 10sec. | |

$I_{FP}$ Conditions : Pulse Width ≤ 10msec. and Duty ≤ 1/10

(2) Initial Electrical/Optical Characteristics (Ta=25°C)

| Item | | Symbol | Condition | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| Forward Voltage | | $V_F$ | $I_F$=20[mA] | - | 3.6 | 4.0 | V |
| Reverse Current | | $I_R$ | $V_R$= 5[V] | - | - | 50 | μA |
| Luminous Intensity (new)*1 | Rank U | $I_V$ | $I_F$=20[mA] | 4880 | 5840 | 6960 | mcd |
| | Rank T | $I_V$ | $I_F$=20[mA] | 3480 | 4000 | 4880 | mcd |
| | Rank S | $I_V$ | $I_F$=20[mA] | 2440 | 2920 | 3480 | mcd |
| | Rank R | $I_V$ | $I_F$=20[mA] | 1740 | 2000 | 2440 | mcd |
| Luminous Intensity (old)*2 | Rank U | $I_V$ | $I_F$=20[mA] | 5760 | 6920 | 8240 | mcd |
| | Rank T | $I_V$ | $I_F$=20[mA] | 4120 | 4800 | 5760 | mcd |
| | Rank S | $I_V$ | $I_F$=20[mA] | 2880 | 3460 | 4120 | mcd |
| | Rank R | $I_V$ | $I_F$=20[mA] | 2060 | 2400 | 2880 | mcd |

* Luminous Intensity Measurement allowance is ± 10%.

*1 Change previously listed luminous intensity values (see *2) to luminous intensity values traceable to the current national standards on and after January 1, 2009.
(In accordance with CIE 127:2007)

Color Rank ($I_F$=20mA,Ta=25°C)

| | Rank W | | | |
|---|---|---|---|---|
| x | 0.11 | 0.11 | 0.15 | 0.15 |
| y | 0.04 | 0.10 | 0.10 | 0.04 |

* Color Coordinates Measurement allowance is ± 0.01.

* Basically, a shipment shall consist of the LEDs of a combination of the above ranks.
The percentage of each rank in the shipment shall be determined by Nichia.

2.INITIAL OPTICAL/ELECTRICAL CHARACTERISTICS

Please refer to “CHARACTERISTICS” on the following pages.

-1-

Abbildung 8.16: Datenblatt - LED

XILINX®

# Spartan-3 1.2V FPGA Family: Introduction and Ordering Information

DS099-1 (v1.1) April 24, 2003

**Advance Product Specification**

## Introduction

The 1.2V Spartan™-3 family of Field-Programmable Gate Arrays is specifically designed to meet the needs of high volume, cost-sensitive consumer electronic applications. The eight-member family offers densities ranging from 50,000 to five million system gates, as shown in Table 1.

The Spartan-3 family builds on the success of the earlier Spartan-IIE family by increasing the amount of logic resources, the capacity of internal RAM, the total number of I/Os, and the overall level of performance as well as by improving clock management functions. Numerous enhancements derive from state-of-the-art Virtex™-II technology. These Spartan-3 enhancements, combined with advanced process technology, deliver more functionality and bandwidth per dollar than was previously possible, setting new standards in the programmable logic industry.

Because of their exceptionally low cost, Spartan-3 FPGAs are ideally suited to a wide range of consumer electronics applications, including broadband access, home networking, display/projection and digital television equipment.

The Spartan-3 family is a superior alternative to mask programmed ASICs. FPGAs avoid the high initial cost, the lengthy development cycles, and the inherent inflexibility of conventional ASICs. Also, FPGA programmability permits design upgrades in the field with no hardware replacement necessary, an impossibility with ASICs.

## Features

- Revolutionary 90-nanometer process technology
- Very low cost, high-performance logic solution for high-volume, consumer-oriented applications
  - Densities as high as 74,880 logic cells
  - 326 MHz system clock rate
  - Three separate power supplies for the core (1.2V), I/Os (1.2V to 3.3V), and special functions (2.5V)
- SelectIO™ signaling
  - Up to 784 I/O pins
  - 622 Mb/s data transfer rate per I/O
  - Seventeen single-ended signal standards
  - Six differential signal standards including LVDS
  - Termination by Digitally Controlled Impedance
  - Signal swing ranging from 1.14V to 3.45V
  - Double Data Rate (DDR) support
- Logic resources
  - Abundant, flexible logic cells with registers
  - Wide multiplexers
  - Fast look-ahead carry logic
  - Dedicated 18 x 18 multipliers
  - JTAG logic compatible with IEEE 1149.1/1532 standards
- SelectRAM™ hierarchical memory
  - Up to 1,872 Kbits of total block RAM
  - Up to 520 Kbits of total distributed RAM
- Digital Clock Manager (up to four DCMs)
  - Clock skew elimination
  - Frequency synthesis
  - High resolution phase shifting
- Eight global clock lines and abundant routing
- Fully supported by Xilinx ISE development system
  - Synthesis, mapping, placement and routing

*Table 1:* **Summary of Spartan-3 FPGA Attributes**

| Device | System Gates | Logic Cells | CLB Array (One CLB = Four Slices) | | | Distributed RAM (bits[1]) | Block RAM (bits[1]) | Dedicated Multipliers | DCMs | Maximum User I/O | Maximum Differential I/O Pairs |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Rows | Columns | Total CLBs | | | | | | |
| XC3S50 | 50K | 1,728 | 16 | 12 | 192 | 12K | 72K | 4 | 2 | 124 | 56 |
| XC3S200 | 200K | 4,320 | 24 | 20 | 480 | 30K | 216K | 12 | 4 | 173 | 76 |
| XC3S400 | 400K | 8,064 | 32 | 28 | 896 | 56K | 288K | 16 | 4 | 264 | 116 |
| XC3S1000 | 1M | 17,280 | 48 | 40 | 1,920 | 120K | 432K | 24 | 4 | 391 | 175 |
| | | | | | | | | | | | |
| XC3S2000 | 2M | 46,080 | 80 | 64 | 5,120 | 320K | 720K | 40 | 4 | 565 | 270 |
| XC3S4000 | 4M | 62,208 | 96 | 72 | 6,912 | 432K | 1,728K | 96 | 4 | 712 | 312 |
| XC3S5000 | 5M | 74,880 | 104 | 80 | 8,320 | 520K | 1,872K | 104 | 4 | 784 | 344 |

**Notes:**
1. By convention, one Kb is equivalent to 1,024 bits.

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and disclaimers are as listed at http://www.xilinx.com/legal.htm.
All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

DS099-1 (v1.1) April 24, 2003
**Advance Product Specification**
www.xilinx.com
1-800-255-7778
1

Abbildung 8.17: Datenblatt - FPGA

AD9461

**AC SPECIFICATIONS**

AVDD1 = 3.3 V, AVDD2 = 5.0 V, DRVDD = 3.3 V, LVDS mode, specified minimum sample rate, 3.4 V p-p differential input, internal trimmed reference (1.7 V mode), $A_{IN}$ = −1.0 dBFS, DCS on, SFDR = AGND, unless otherwise noted.

**Table 2.**

| Parameter | Temp | AD9461BSVZ Min | Typ | Max | Unit |
|---|---|---|---|---|---|
| SIGNAL-TO-NOISE RATIO (SNR) | | | | | |
| $f_{IN}$ = 10 MHz | 25°C | 76.3 | 77.7 | | dB |
| | Full | 76.0 | | | dB |
| $f_{IN}$ = 170 MHz[1] | 25°C | 74.2 | 76.0 | | dB |
| | Full | 73.8 | | | dB |
| $f_{IN}$ = 225 MHz | 25°C | | 74.4 | | dB |
| $f_{IN}$ = 225 MHz @125 MSPS | 25°C | | 75.3 | | dB |
| SIGNAL-TO-NOISE AND DISTORTION (SINAD) | | | | | |
| $f_{IN}$ = 10 MHz | 25°C | 74.0 | 76.7 | | dB |
| | Full | 74.0 | | | dB |
| $f_{IN}$ = 170 MHz[1] | 25°C | 71.9 | 75.1 | | dB |
| | Full | 68.3 | | | dB |
| $f_{IN}$ = 225 MHz | 25°C | | 73.5 | | dB |
| $f_{IN}$ = 225 MHz @125 MSPS | 25°C | | 74.6 | | dB |
| EFFECTIVE NUMBER OF BITS (ENOB) | | | | | |
| $f_{IN}$ = 10 MHz | 25°C | | 12.5 | | Bits |
| $f_{IN}$ = 170 MHz[1] | 25°C | | 12.2 | | Bits |
| $f_{IN}$ = 225 MHz | 25°C | | 11.9 | | Bits |
| SPURIOUS-FREE DYNAMIC RANGE (SFDR, SECOND OR THIRD HARMONIC) | | | | | |
| $f_{IN}$ = 10 MHz | 25°C | 82 | 90 | | dBc |
| | Full | 80 | | | dBc |
| $f_{IN}$ = 170 MHz[1] | 25°C | 77 | 84 | | dBc |
| | Full | 71 | | | dBc |
| $f_{IN}$ = 225 MHz | 25°C | | 82 | | dBc |
| $f_{IN}$ = 225 MHz @125 MSPS | 25°C | | 86 | | dBc |
| WORST SPUR EXCLUDING SECOND OR THIRD HARMONICS | | | | | |
| $f_{IN}$ = 10 MHz | 25°C | 88 | 96 | | dBc |
| | Full | 86 | | | dBc |
| $f_{IN}$ = 170 MHz[1] | 25°C | 89 | 95 | | dBc |
| | Full | 85 | | | dBc |
| $f_{IN}$ = 225 MHz | 25°C | | 91 | | dBc |
| $f_{IN}$ = 225 MHz @ 125 MSPS | 25°C | | 93 | | dBc |
| TWO-TONE SFDR | | | | | |
| $f_{IN}$ = 169.6 MHz @ −7 dBFS, 170.6 MHz @ −7 dBFS | 25°C | | 89 | | dBFS |
| ANALOG BANDWIDTH | Full | | 615 | | MHz |

[1] SFDR = high (AVDD1). See the Operational Mode Selection section.

Rev. 0 | Page 4 of 28

Abbildung 8.18: Datenblatt - ADC Teil 1

## SPECIFICATIONS

### DC SPECIFICATIONS

AVDD1 = 3.3 V, AVDD2 = 5.0 V, DRVDD = 3.3 V, LVDS mode, specified minimum sampling rate, 3.4 V p-p differential input, internal trimmed reference (1.0 V mode), $A_{IN}$ = −1.0 dBFS, DCS on, SFDR = AGND, unless otherwise noted.

**Table 1.**

| | | AD9461BSVZ | | | |
|---|---|---|---|---|---|
| **Parameter** | **Temp** | **Min** | **Typ** | **Max** | **Unit** |
| RESOLUTION | Full | | 16 | | Bits |
| ACCURACY | | | | | |
| No Missing Codes | Full | | Guaranteed | | |
| Offset Error | Full | −4.2 | ±0.1 | +4.2 | mV |
| Gain Error | 25°C | −3 | ±0.5 | +3 | % FSR |
| | Full | −3.4 | | +3.4 | % FSR |
| Differential Nonlinearity (DNL)[1] | 25°C | −1.0 | ±0.6 | +1.0 | LSB |
| | Full | −1.0 | | +1.3 | LSB |
| Integral Nonlinearity (INL)[1] | 25°C | −7 | ±5.0 | +7 | LSB |
| VOLTAGE REFERENCE | | | | | |
| Output Voltage VREF = 1.7 V | Full | | +1.7 | | V |
| Load Regulation @ 1.0 mA | Full | | ±2 | | mV |
| Reference Input Current (External VREF = 1.7 V) | Full | | 350 | | µA |
| INPUT REFERRED NOISE | 25°C | | 2.6 | | LSB rms |
| ANALOG INPUT | | | | | |
| Input Span | | | | | |
| VREF = 1.7 V | Full | | 3.4 | | V p-p |
| VREF = 1.0 V | Full | | 2.0 | | V p-p |
| Internal Input Common-Mode Voltage | Full | | 3.5 | | V |
| External Input Common-Mode Voltage | Full | 3.2 | | 3.9 | V |
| Input Resistance[2] | Full | | 1 | | kΩ |
| Input Capacitance[2] | Full | | 6 | | pF |
| POWER SUPPLIES | | | | | |
| Supply Voltage | | | | | |
| AVDD1 | Full | 3.14 | 3.3 | 3.46 | V |
| AVDD2 | Full | 4.75 | 5.0 | 5.25 | V |
| DRVDD—LVDS Outputs | Full | 3.0 | | 3.6 | V |
| DRVDD—CMOS Outputs | Full | 3.0 | 3.3 | 3.6 | V |
| Supply Current[1] | | | | | |
| AVDD1 | Full | | 405 | 426 | mA |
| AVDD2[1, 3] | Full | | 131 | 143 | mA |
| $I_{DRVDD}$[1]—LVDS Outputs | Full | | 72 | 81 | mA |
| $I_{DRVDD}$[1]—CMOS Outputs | Full | | 14 | | mA |
| PSRR | | | | | |
| Offset | Full | | 1 | | mV/V |
| Gain | Full | | 0.2 | | %/V |
| POWER CONSUMPTION | | | | | |
| LVDS Outputs | Full | | 2.2 | 2.4 | W |
| CMOS Outputs (DC Input) | Full | | 2.0 | | W |

[1] Measured at the maximum clock rate, $f_{IN}$ = 15 MHz, full-scale sine wave, with a 100 Ω differential termination on each pair of output bits for LVDS output mode and approximately 5 pF loading on each output bit for CMOS output mode.
[2] Input capacitance or resistance refers to the effective impedance between one differential input pin and AGND. Refer to Figure 6 for the equivalent analog input structure.
[3] For SFDR = AVDD1, $I_{AVDD2}$ decreases by ~8 mA, decreasing power dissipation.

Rev. 0 | Page 3 of 28

Abbildung 8.19: Datenblatt - ADC Teil 2

## DIGITAL SPECIFICATIONS

AVDD1 = 3.3 V, AVDD2 = 5.0 V, DRVDD = 3.3 V, $R_{LVDS_BIAS}$ = 3.74 kΩ, unless otherwise noted.

**Table 3.**

| Parameter | Temp | AD9461BSVZ | | | Unit |
|---|---|---|---|---|---|
| | | Min | Typ | Max | |
| CMOS LOGIC INPUTS (DFS, DCS MODE, OUTPUT MODE) | | | | | |
| High Level Input Voltage | Full | 2.0 | | | V |
| Low Level Input Voltage | Full | | | 0.8 | V |
| High Level Input Current | Full | | | 200 | µA |
| Low Level Input Current | Full | −10 | | +10 | µA |
| Input Capacitance | Full | | 2 | | pF |
| DIGITAL OUTPUT BITS—CMOS MODE (D0 to D15, OTR)[1] | | | | | |
| High Level Output Voltage | Full | 3.25 | | | V |
| Low Level Output Voltage | Full | | | 0.2 | V |
| DIGITAL OUTPUT BITS—LVDS MODE (D0 to D15, OTR) | | | | | |
| $V_{OD}$ Differential Output Voltage[2] | Full | 247 | | 545 | mV |
| $V_{OS}$ Output Offset Voltage | Full | 1.125 | | 1.375 | V |
| CLOCK INPUTS (CLK+, CLK−) | | | | | |
| Differential Input Voltage | Full | 0.2 | | | V |
| Common-Mode Voltage | Full | 1.3 | 1.5 | 1.6 | V |
| Input Resistance | Full | 1.1 | 1.4 | 1.7 | kΩ |
| Input Capacitance | Full | | 2 | | pF |

[1] Output voltage levels measured with 5 pF load on each output.
[2] LVDS $R_{TERM}$ = 100 Ω.

## SWITCHING SPECIFICATIONS

AVDD1 = 3.3 V, AVDD2 = 5.0 V, DRVDD = 3.3 V, unless otherwise noted.

**Table 4.**

| Parameter | Temp | AD9461BSVZ | | | Unit |
|---|---|---|---|---|---|
| | | Min | Typ | Max | |
| CLOCK INPUT PARAMETERS | | | | | |
| Maximum Conversion Rate | Full | 130 | | | MSPS |
| Minimum Conversion Rate | Full | | | 1 | MSPS |
| CLK Period | Full | 7.7 | | | ns |
| CLK Pulse Width High[1] ($t_{CLKH}$) | Full | 3.1 | | | ns |
| CLK Pulse Width Low[1] ($t_{CLKL}$) | Full | 3.1 | | | ns |
| DATA OUTPUT PARAMETERS | | | | | |
| Output Propagation Delay—CMOS ($t_{PD}$)[2] (Dx, DCO+) | Full | | 3.35 | | ns |
| Output Propagation Delay—LVDS ($t_{PD}$)[3] (Dx+), ($t_{CPD}$)[3] (DCO+) | Full | 2.3 | 3.6 | 4.8 | ns |
| Pipeline Delay (Latency) | Full | | 13 | | Cycles |
| Aperture Uncertainty (Jitter, $t_J$) | Full | | 60 | | fsec rms |

[1] With duty cycle stabilizer (DCS) enabled.
[2] Output propagation delay is measured from clock 50% transition to data 50% transition with 5 pF load.
[3] LVDS $R_{TERM}$ = 100 Ω. Measured from the 50% point of the rising edge of CLK+ to the 50% point of the data transition.

Rev. 0 | Page 5 of 28

Abbildung 8.20: Datenblatt - ADC Teil 3

TEXAS INSTRUMENTS

**TPS737xx**

www.ti.com SBVS067I–JANUARY 2006–REVISED MARCH 2009

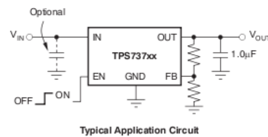
# 1A Low-Dropout Regulator with Reverse Current Protection

## FEATURES

- Stable with 1.0μF or Larger Ceramic Output Capacitor
- Input Voltage Range: 2.2V to 5.5V
- Ultra-Low Dropout Voltage: 130mV typ at 1A
- Excellent Load Transient Response—Even With Only 1.0μF Output Capacitor
- NMOS Topology Delivers Low Reverse Leakage Current
- 1.0% Initial Accuracy
- 3% Overall Accuracy Over Line, Load, and Temperature
- Less Than 20nA typical $I_Q$ in Shutdown Mode
- Thermal Shutdown and Current Limit for Fault Protection
- Available in Multiple Output Voltage Versions
  - Adjustable Output: 1.20V to 5.5V
  - Custom Outputs Available Using Factory Package-Level Programming

## APPLICATIONS

- Point of Load Regulation for DSPs, FPGAs, ASICs, and Microprocessors
- Post-Regulation for Switching Supplies
- Portable/Battery-Powered Equipment



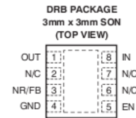

Typical Application Circuit

## DESCRIPTION

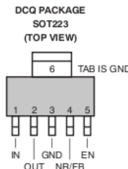
The TPS737xx family of linear low-dropout (LDO) voltage regulators uses an NMOS pass element in a voltage-follower configuration. This topology is relatively insensitive to output capacitor value and ESR, allowing a wide variety of load configurations. Load transient response is excellent, even with a small 1.0μF ceramic output capacitor. The NMOS topology also allows very low dropout.

The TPS737xx family uses an advanced BiCMOS process to yield high precision while delivering very low dropout voltages and low ground pin current. Current consumption, when not enabled, is under 20nA and ideal for portable applications. These devices are protected by thermal shutdown and foldback current limit.

DRB PACKAGE
3mm x 3mm SON
(TOP VIEW)




DCQ PACKAGE
SOT223
(TOP VIEW)




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.

Copyright © 2006–2009, Texas Instruments Incorporated

Abbildung 8.21: Datenblatt - Spanungsregler TI

# Schaltpläne

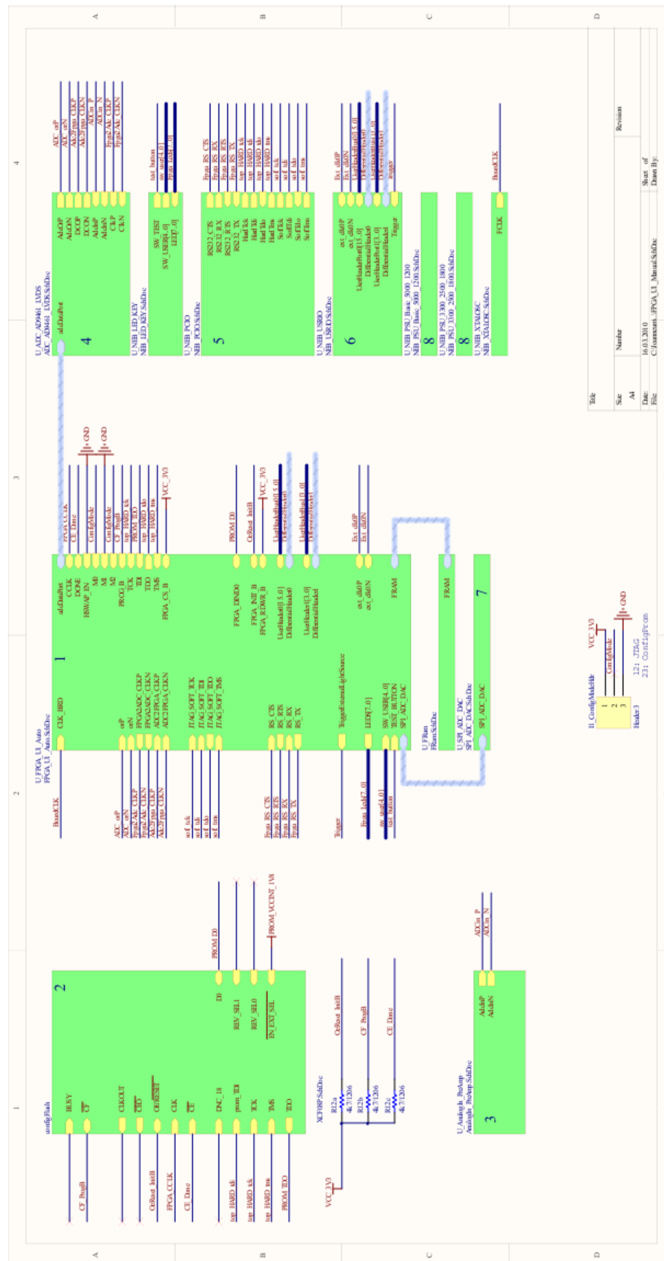


Abbildung 8.22: Schaltplan Top Level

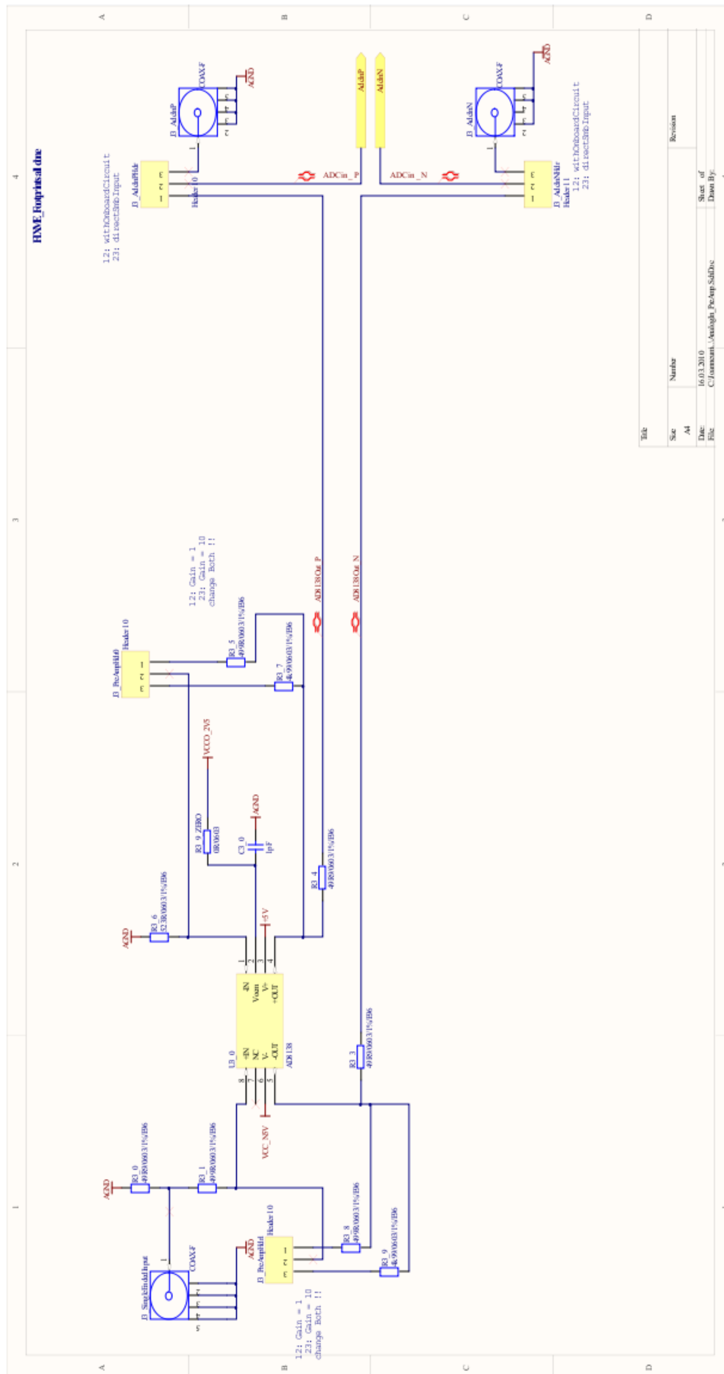


Abbildung 8.23: Schaltplan Differentieller Eingangstreiber

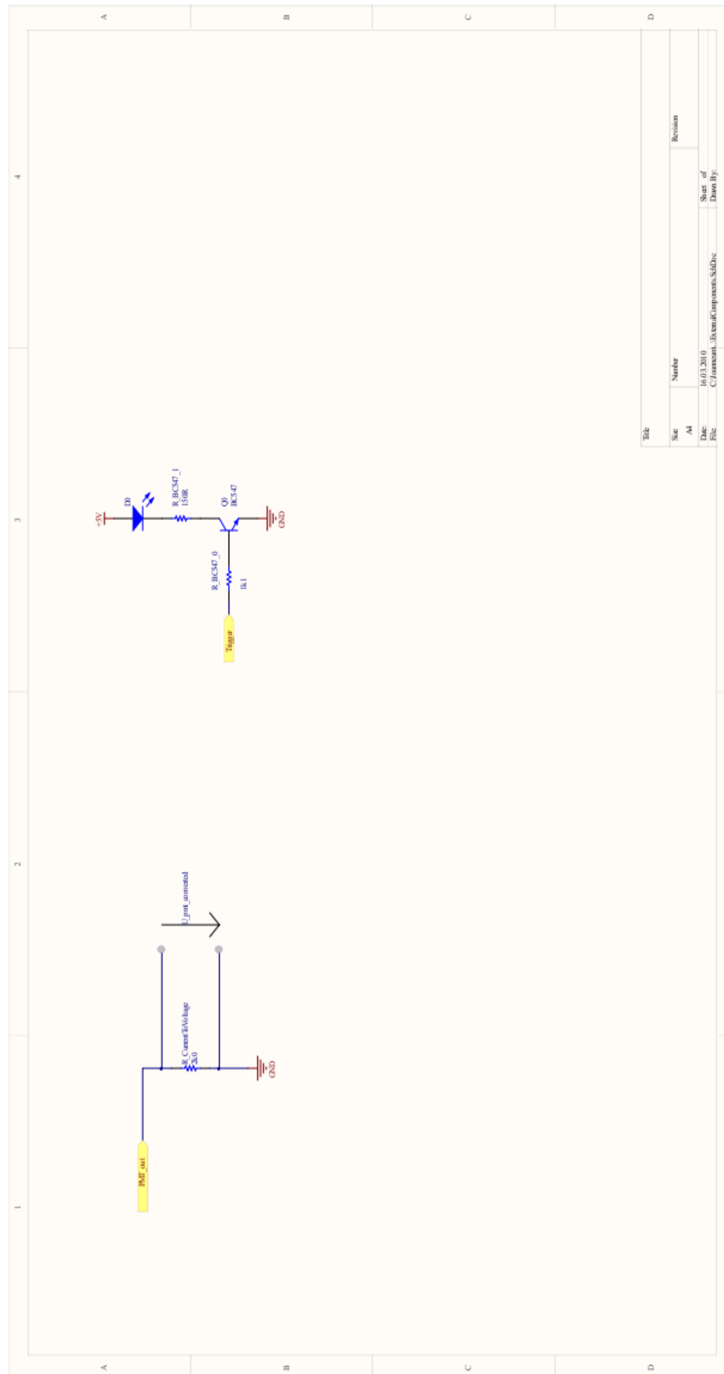



Abbildung 8.24: Schaltplan Strom-Spannungswandler, Led Treiber

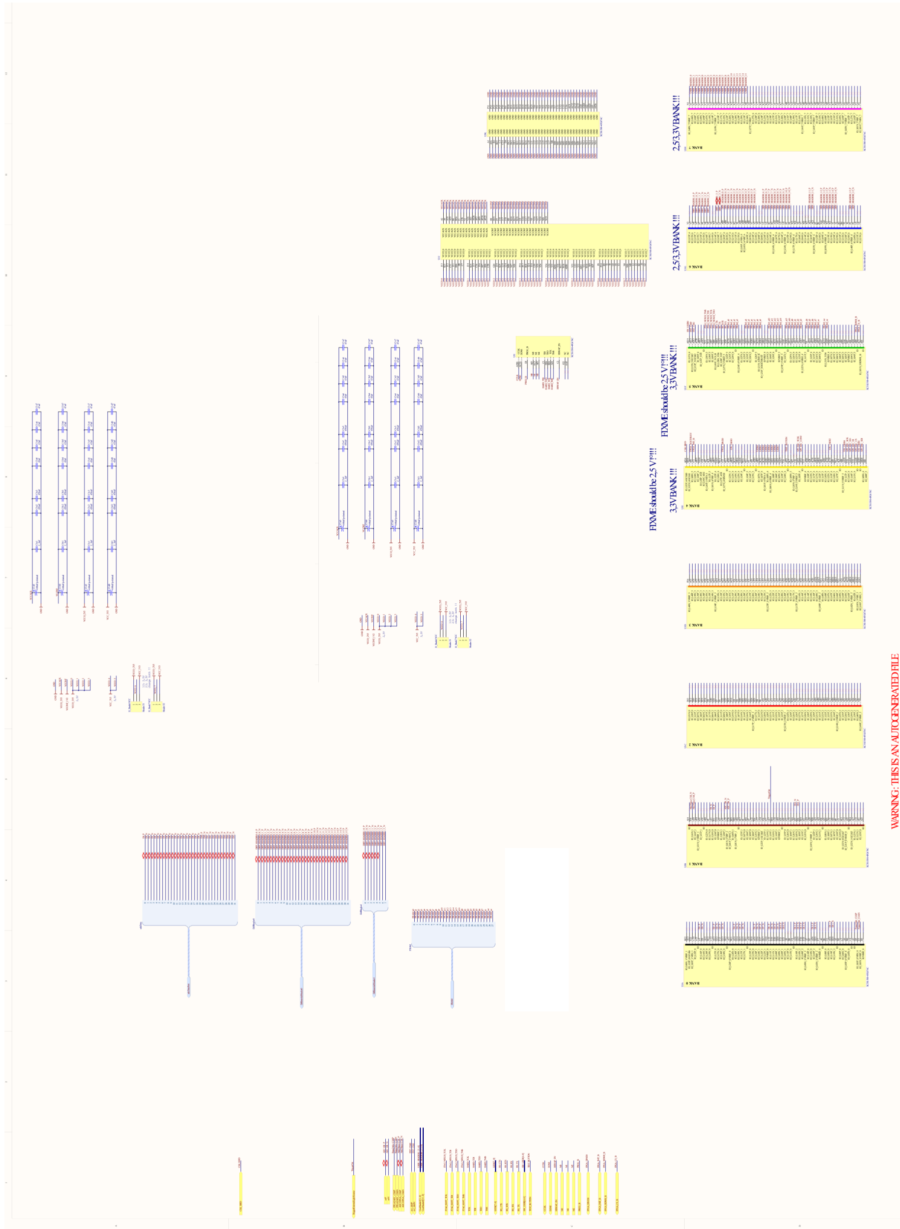


Abbildung 8.25: Schaltplan FPGA

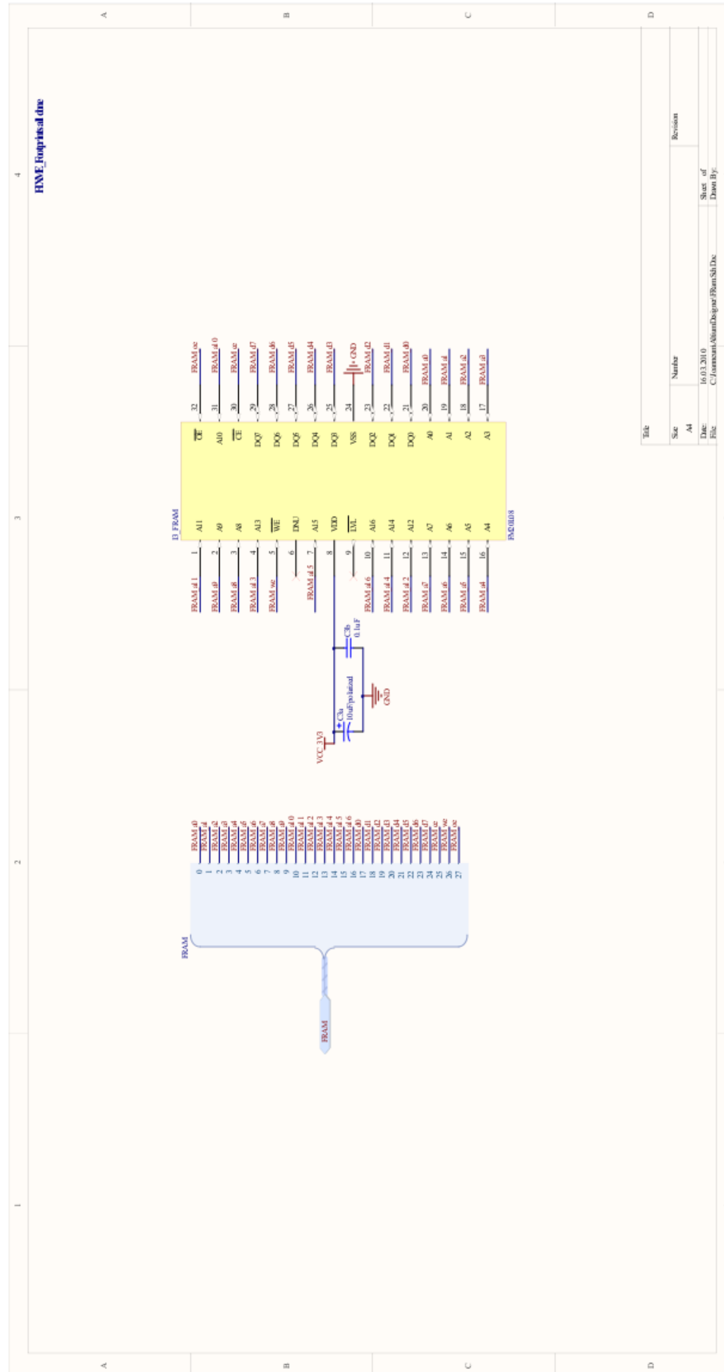


Abbildung 8.26: Schaltplan FRAM

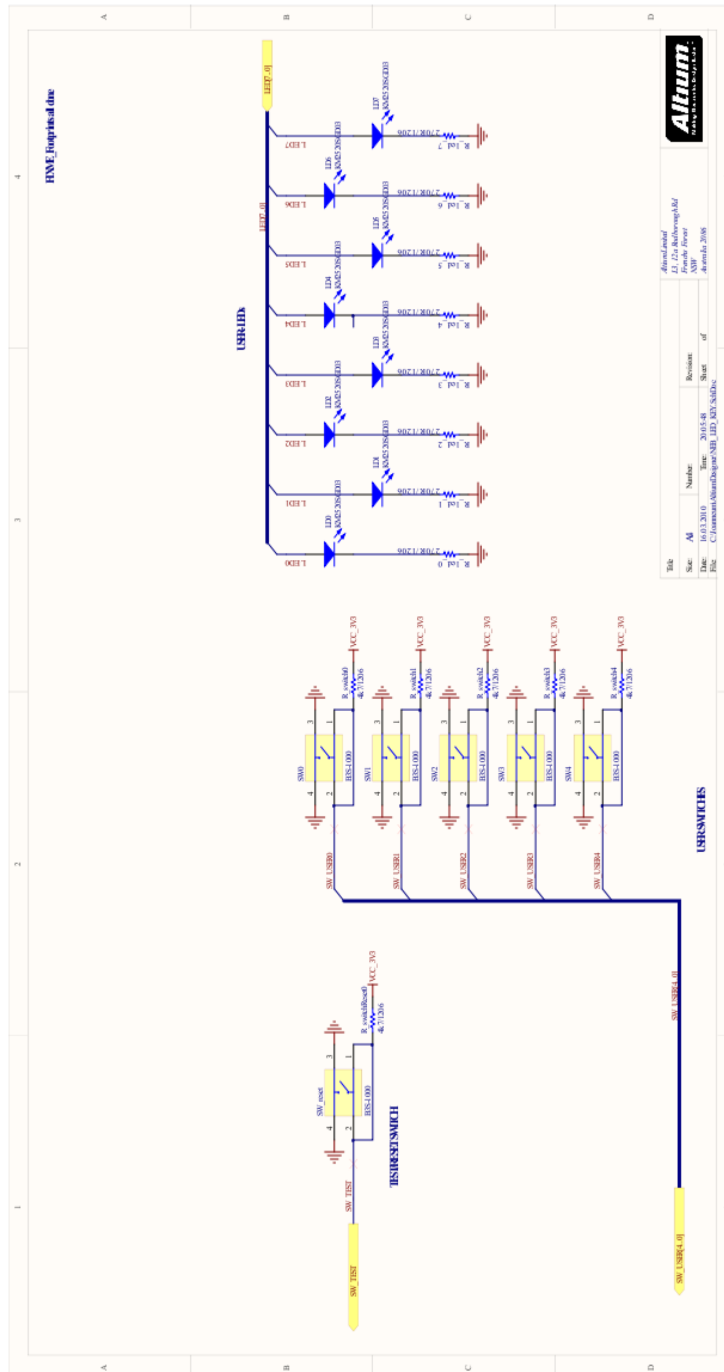


Abbildung 8.27: Schaltplan LED's, IO's

Abbildung 8.28: Schaltplan RS232

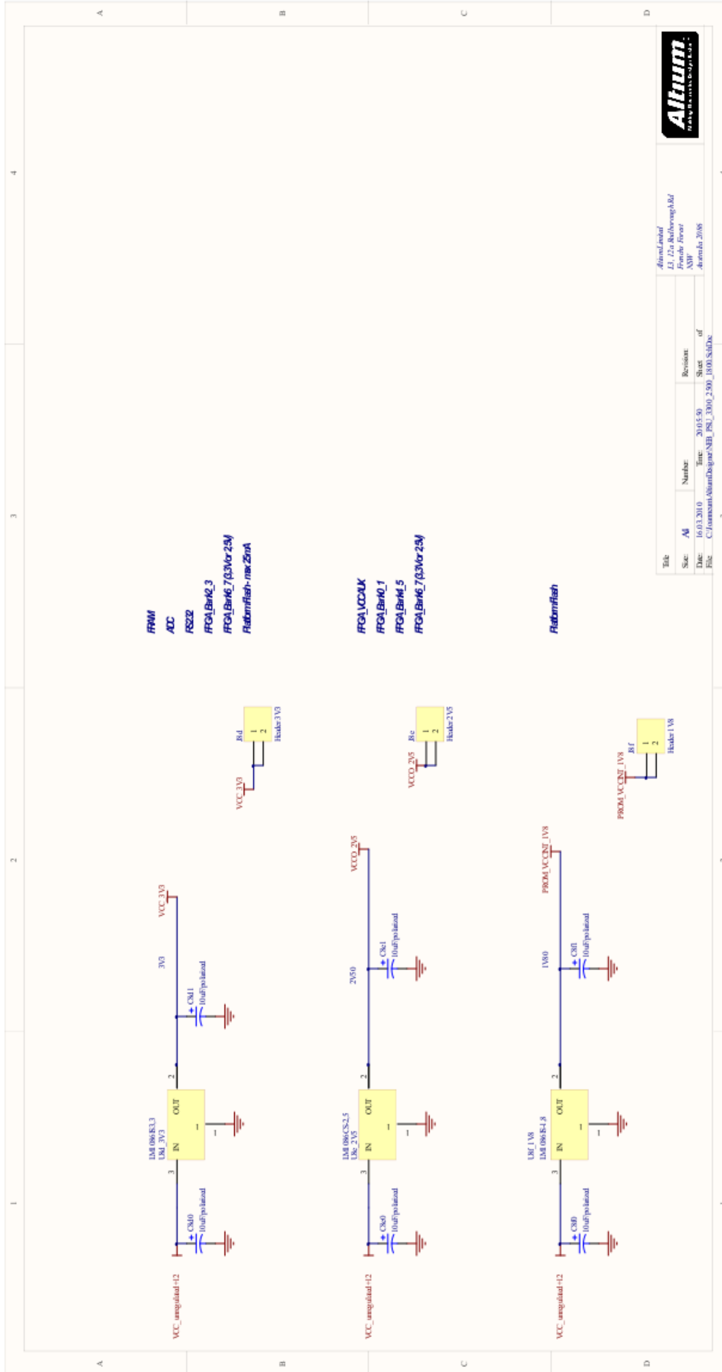


Abbildung 8.29: Schaltplan Spannungsversorgung Teil 1

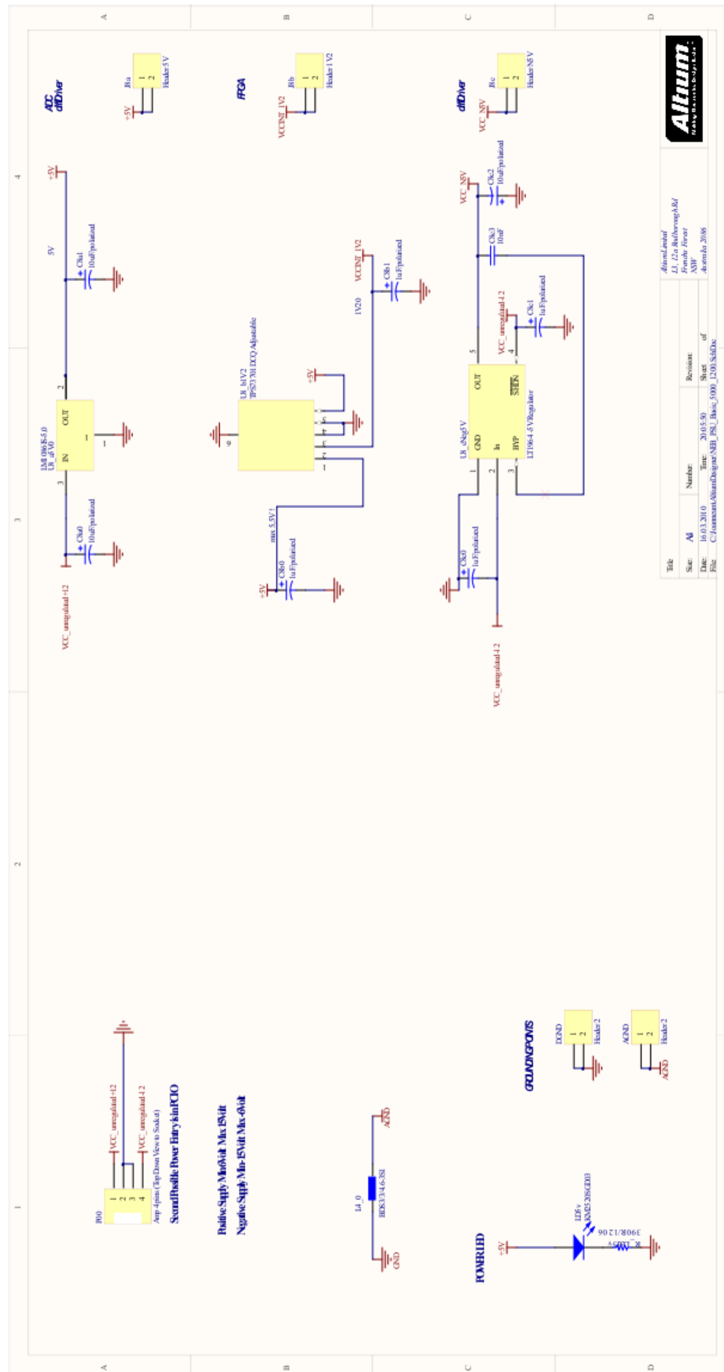


Abbildung 8.30: Schaltplan Spannungsversorgung Teil 2

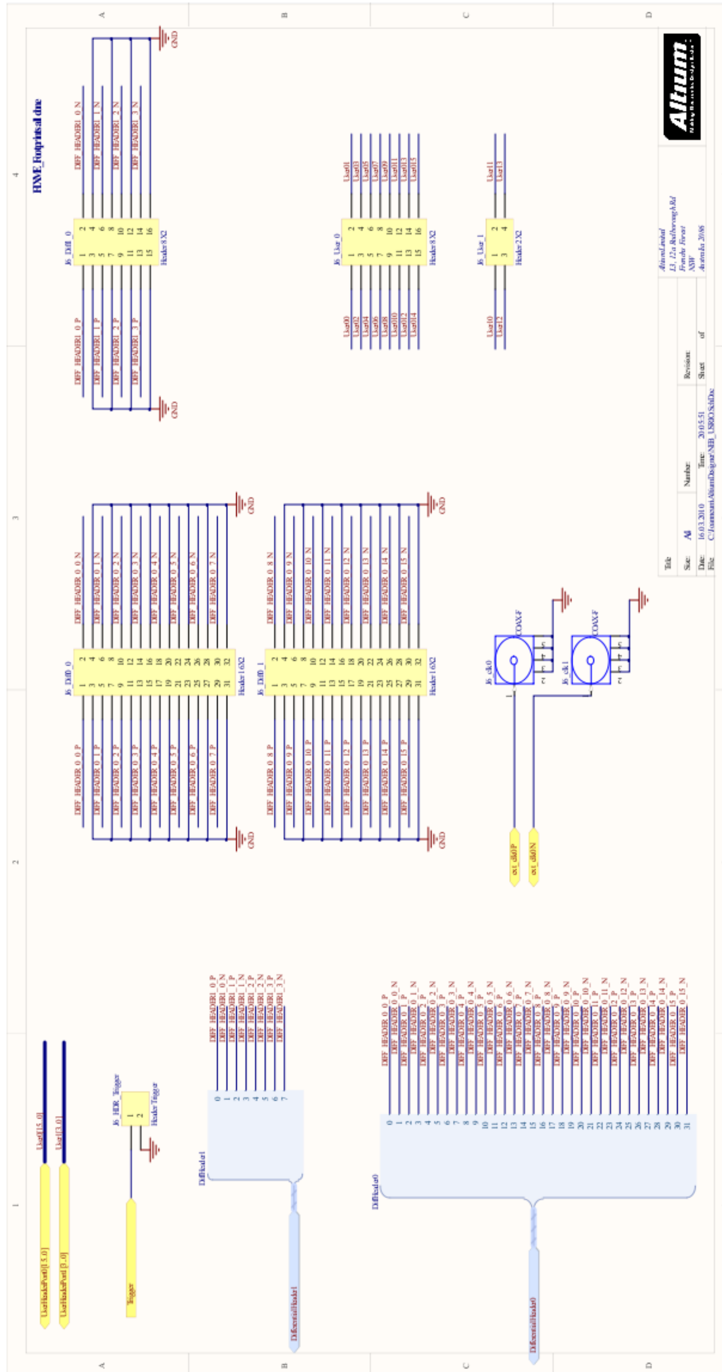


Abbildung 8.31: Schaltplan IO Erweiterungen

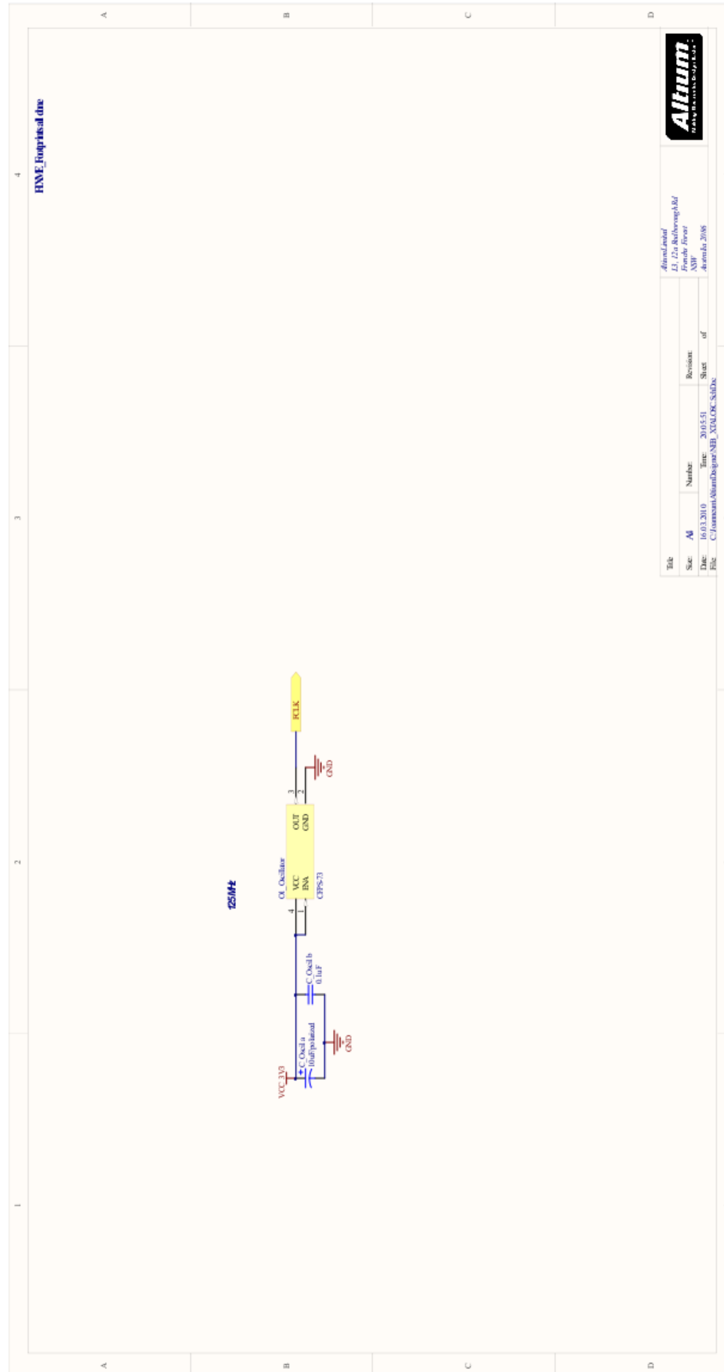



Abbildung 8.32: Schaltplan Oszillator

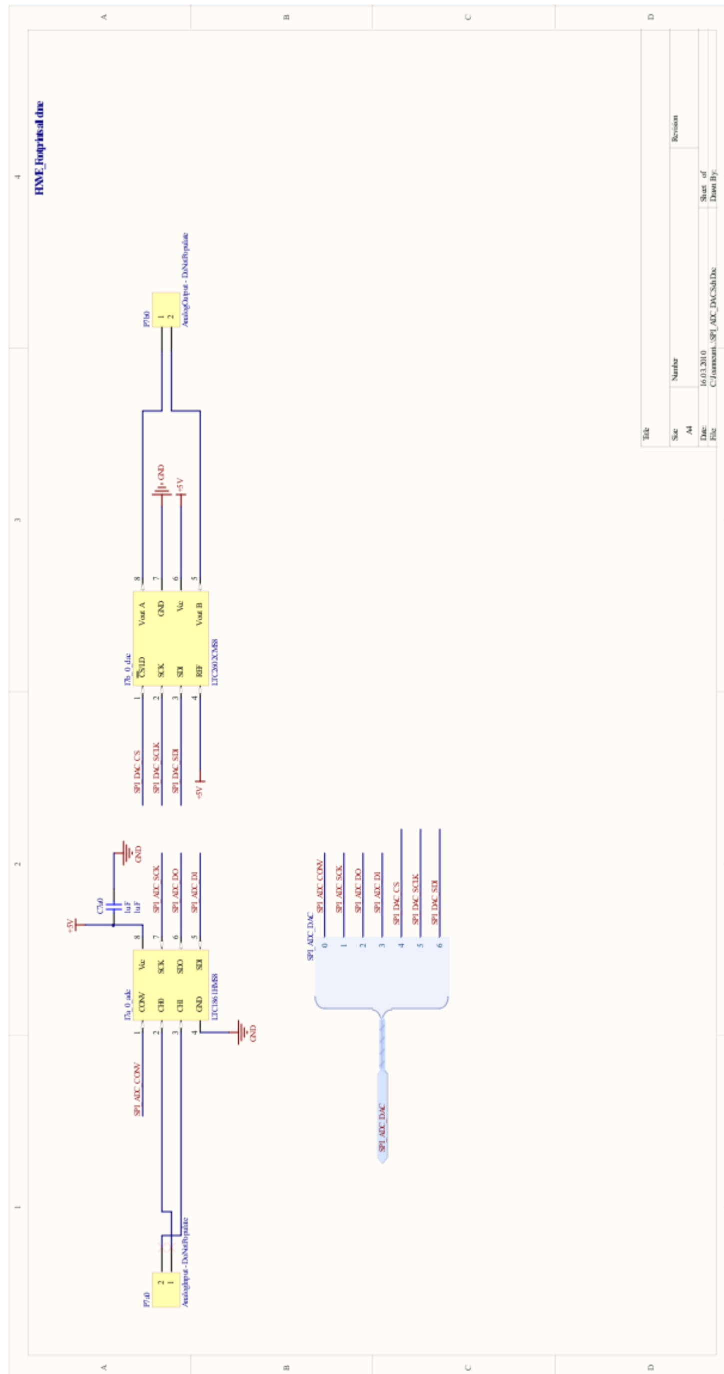


Abbildung 8.33: Schaltplan Erweiterungs ADC und DAC

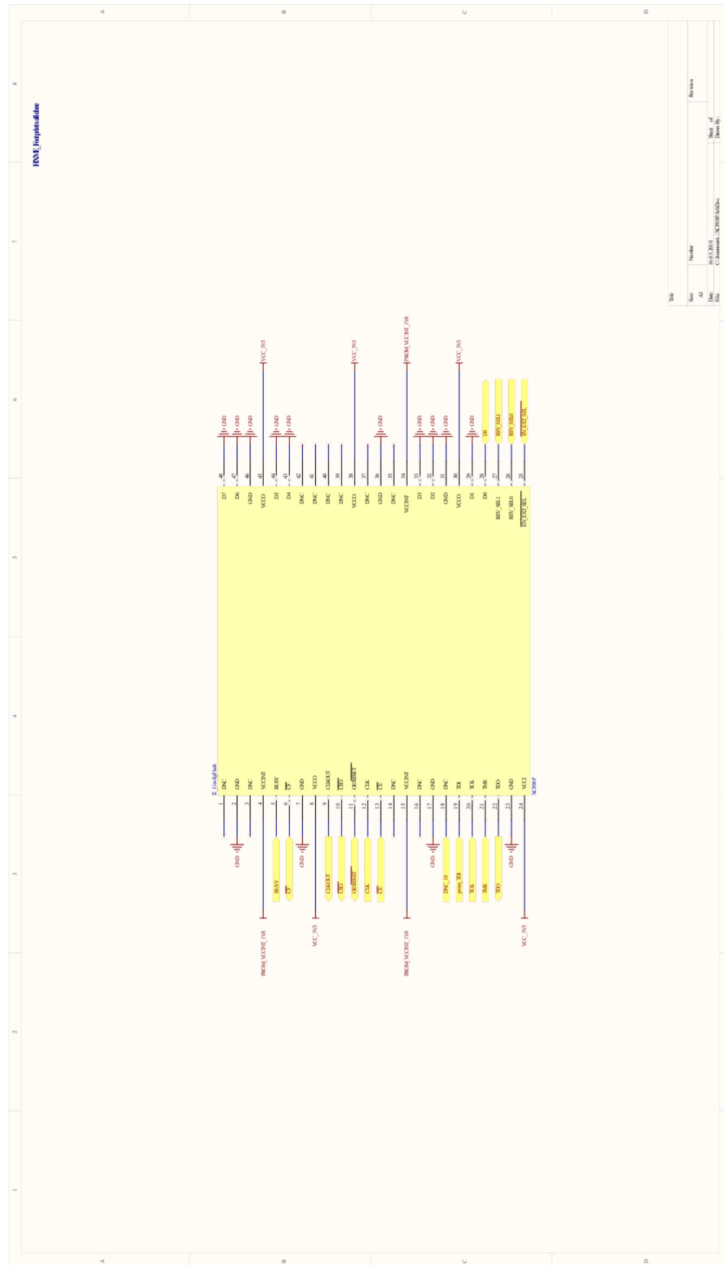


Abbildung 8.34: Schaltplan Config PROM

## FPGA Konfiguration

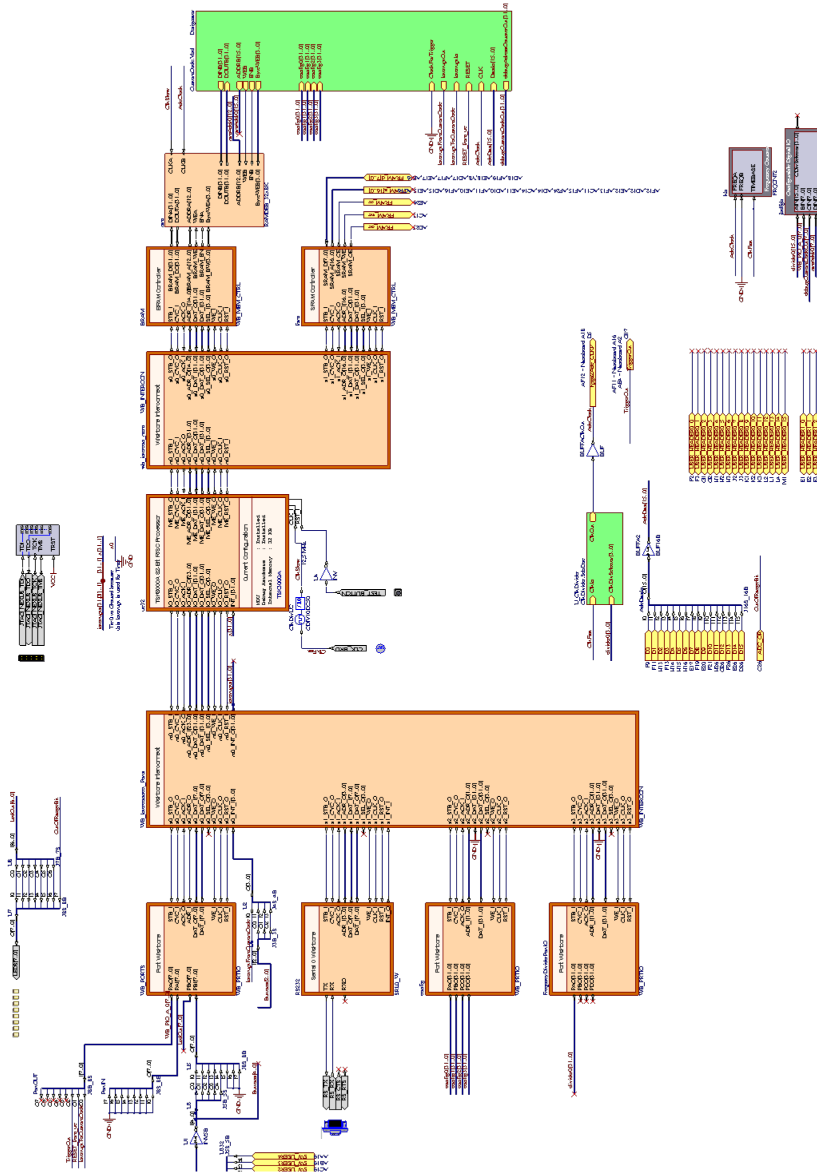


Abbildung 8.35: FPGA Konfiguration

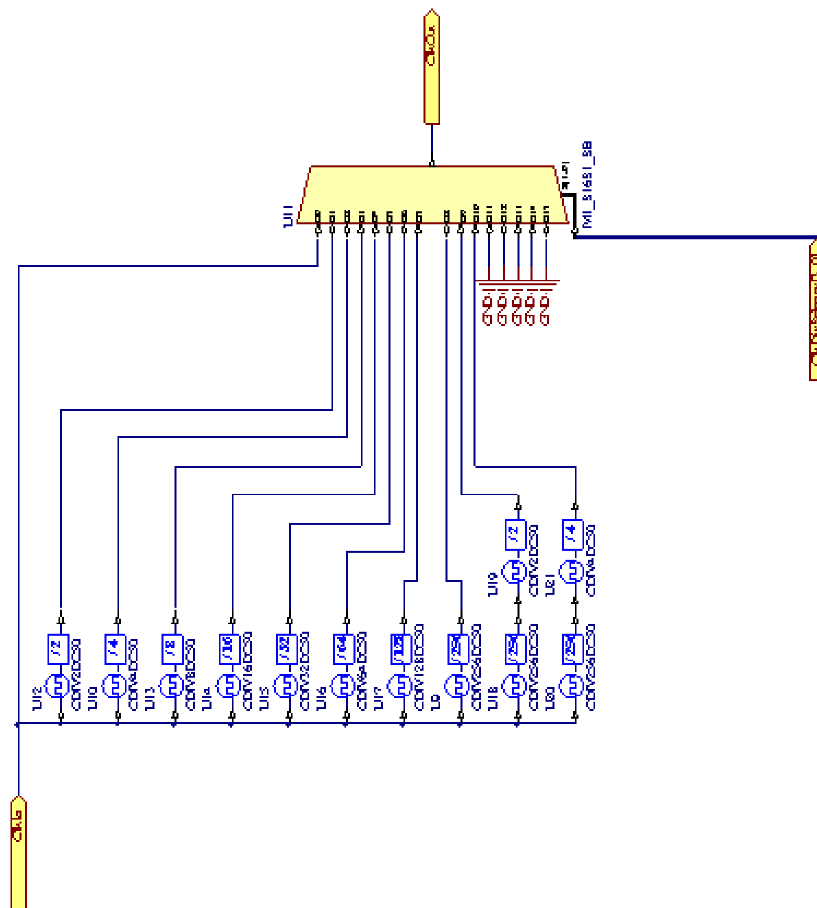


Abbildung 8.36: Taktteiler Modul

## VHDL Code

Listing 1: VHDL Modul zum Einlesen der ADC Daten

```
 -- SubModule CustomCode
-- Created    05.02.2009 13:43:42

library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;

entity CustomCode is port
   (
     DINB            : out   std_logic_vector(31 downto 0);
     DOUTB           : in    std_logic_vector(31 downto 0);
     ADDRB           : out   std_logic_vector(15 downto 0);
     WEB             : out   std_logic;   -- 1..write / 0..read
     ENB             : out   std_logic;   -- enable
     ByteWEB         : out   std_logic_vector(3 downto 0);
     config0         : in    std_logic_vector(31 downto 0);
     config1         : in    std_logic_vector(31 downto 0);
     config2         : in    std_logic_vector(31 downto 0);
     config3         : in    std_logic_vector(31 downto 0);
     DataIn          : in    std_logic_vector(15 downto 0);
     CLK             : in    std_logic;
     InterruptOut    : out   std_logic;
     InterruptIn     : in    std_logic;
     debugAdressCounterOut : out    std_logic_vector(31 downto
        0);
     RESET                 : in     std_logic;
     ClockForTrigger       : in     std_logic
   );
end CustomCode;

architecture Structure of CustomCode is

-- Component Declarations

-- Signal Declarations
    type state_type is (st_startup,st_init,st_wait,
                        fillRamWithAdcDataInit0a,
                        fillRamWithAdcDataInit0b,
                        fillRamWithAdcData0,
                        fillRamWithAdcData1,
                        fillRamWithAdcData2,
```

```
                  fillRamWithAdcData3,
                  fillRamWithAdcData4);
  signal next_state, current_state: state_type := st_startup;
  signal addressCounter          :integer range 0 to 65534 ;
  signal firstElement            :integer range 0 to 65534 ;
  signal lastElement             :integer range 0 to 65534 ;
  signal internalInterrupt       :std_logic := '0';
  signal InterruptForOutput      :std_logic := '0';
  CONSTANT CustomCode_FillRamWithAdcData :std_logic_vector(7
     downto 0 := x"02";

begin

  ENB <= '1';
  debugAdressCounterOut(15 downto  8) <=
     CONV_STD_LOGIC_VECTOR(addressCounter,8);
       -- läuft nicht los
  debugAdressCounterOut(23 downto 16) <=  config1(7 downto 0);
  debugAdressCounterOut(29 downto 24) <=  (others => '0');
  debugAdressCounterOut(30) <=  InterruptForOutput;
  debugAdressCounterOut(31) <=  InterruptIn;

  InterruptOut <= InterruptForOutput;

  FSM_Process : process(CLK,RESET)
  begin

      if(RESET = '1') then
         current_state <= st_init;
      elsif (CLK'event and CLK='1') then
         current_state <= next_state;
      end if;
  end process;


  FSM_comb_logic:
     process(current_state,internalInterrupt,addressCounter)
   begin
   next_state <= current_state;
   debugAdressCounterOut(7 downto 0) <= x"10";
   case current_state is
      when st_startup =>
         null;
         debugAdressCounterOut(7 downto 0) <= x"01";
      when st_init =>
```

```
    next_state <= st_wait;
   debugAdressCounterOut(7 downto 0) <= x"02";
when st_wait =>
   debugAdressCounterOut(7 downto 0) <= x"03";
    if(internalInterrupt = '1') then
       debugAdressCounterOut(7 downto 0) <= x"04";
       if (config1(7 downto 0) =
          CustomCode_FillRamWithAdcData) then
       debugAdressCounterOut(7 downto 0) <= x"05";
       next_state <= fillRamWithAdcDataInit0a;
       end if;
    end if;


when fillRamWithAdcDataInit0a =>
       firstElement <= CONV_INTEGER( unsigned(config2));
       lastElement  <= CONV_INTEGER( unsigned(config3));
       next_state <= fillRamWithAdcDataInit0b;
   debugAdressCounterOut(7 downto 0) <= x"06";
when fillRamWithAdcDataInit0b =>
    next_state <= fillRamWithAdcData0;
   debugAdressCounterOut(7 downto 0) <= x"07";
when fillRamWithAdcData0 =>
    next_state <= fillRamWithAdcData1;
   debugAdressCounterOut(7 downto 0) <= x"08";
when fillRamWithAdcData1 => -- low word
    next_state <= fillRamWithAdcData2;
   debugAdressCounterOut(7 downto 0) <= x"09";
when fillRamWithAdcData2 => -- high word
   debugAdressCounterOut(7 downto 0) <= x"0a";
    if(addressCounter = lastElement) then
       debugAdressCounterOut(7 downto 0) <= x"0b";
       next_state <= fillRamWithAdcData3;
    else
       next_state <= fillRamWithAdcData1;
       debugAdressCounterOut(7 downto 0) <= x"0c";
    end if;
when fillRamWithAdcData3 =>
   debugAdressCounterOut(7 downto 0) <= x"0d";
    --next_state <= st_wait;



when others =>
   debugAdressCounterOut(7 downto 0) <= x"0f";
```

```
        null;

 end case;

 end process;

FSM_output_logic: process(CLK)
VARIABLE elementCounter   : INTEGER;
 begin
 if (CLK'event and CLK='1') then

 InterruptForOutput  <= '0';
 ADDRB <=  (others => '0');
 DINB <= x"aabbaa00";
 WEB   <= '0';
 ByteWEB <= (others => '0');
 case current_state is
     when st_startup =>
          null;
     when st_init =>

     when st_wait =>
         addressCounter <= 0;

     when fillRamWithAdcDataInit0b =>
         addressCounter <= firstElement;
     when fillRamWithAdcData0 =>
         WEB   <= '1';
     when fillRamWithAdcData1 =>   -- low
         WEB   <= '1';
         ByteWEB <= b"0011";
         ADDRB <= CONV_STD_LOGIC_VECTOR(addressCounter,16);
         DINB(15 downto 0  ) <=  DataIn;
     when fillRamWithAdcData2 =>  -- high
         addressCounter <= addressCounter +1;
         WEB   <= '1';
         ByteWEB <= b"1100";
         ADDRB <= CONV_STD_LOGIC_VECTOR(addressCounter,16);
         DINB(31 downto 16 ) <=  DataIn;
     when fillRamWithAdcData3 =>
         InterruptForOutput  <= '1';

     when others =>
        null;
```

```
    end case;
    end if;

    end process;

  InternalInterruptGenerator_Process : process(InterruptIn,
     RESET
                                             , InterruptForOutput)
  begin

       if(RESET = '1' OR InterruptForOutput = '1') then
          internalInterrupt <= '0' ;
       elsif (InterruptIn'event and InterruptIn='1') then
          internalInterrupt <= '1' ;
       end if;
  end process;


end Structure;
```

## Mikrocontroller Code

### Main.c

Listing 2: Mikrocontroller Code

```c
#include "Globals.h"

#include "hardware.h"
#include "settings.h"
#include "Algorithm.h"
#include "MemoryOperations.h"
#include "Communication.h"
#include "SerialProtocol.h"



volatile uint8_t  State_ProtocolReceiver      ;
volatile uint8_t  State_ProtocolTransmitter   ;
volatile uint8_t  ProtocolTransmitter_nextOpCodeToReply   ;

volatile unsigned long long ByteCounter ;

volatile uint8_t   Status_ActiveAquisitionMode ;
volatile uint8_t   Status_ActiveClockDiverIndexInitValue ;

volatile uint8_t   ConfigBlockForCustomCode_OpCode ;
volatile uint32_t  ConfigBlockForCustomCode_StartAddress ;
volatile uint32_t  ConfigBlockForCustomCode_LastAddress ;

volatile uint16_t  NoiseMin;
volatile uint16_t  NoiseMax;
volatile uint16_t  DataHighMin;
volatile uint16_t  DataHighMax;
volatile uint16_t  DataTempMin;
volatile uint16_t  DataTempMax;
volatile uint32_t  DataMaxPosition;
volatile uint32_t  DecayStartPosition;

// is used as round robin storage
volatile double    TAU;
volatile int       currentAverageBlock ;
volatile uint8_t   currentScratchBlock ;
volatile uint8_t   storedAverages ;
```

```
volatile uint8_t    isForMatlab = 0;

//** *****************************************************
// GeneralPurposeIO:
// BIT    Decimal
// 7 ..  128 -
// 6 ..   64 -
// 5 ..   32 -
// 4 ..   16 -
// 3 ..    8 -
// 2 ..    4 -
// 1 ..    2 Reset to customHardware
// 0 ..    1 Interrupt to customHardware
uint32_t * const GeneralPurposeIO = (void *)Base_gpio;
uint32_t * const LedsAddr = (void *)(Base_gpio+1);



//** *****************************************************
// Frequency Divider, Control Address and global Storage Variable
uint32_t * const      FreqencyDividerDataAddr  = (void
    *)(Base_FrequencyDivider);
volatile uint32_t     FreqencyDividerDataIndex = 0;
//** *****************************************************
volatile uint8_t      LastError ;


//** *****************************************************
// Function Headers Prototypes
void powerOn(void);
void init(void);
void ErrorHandler(uint8_t errorType);
void WarnHandler(uint8_t warnType);
uint8_t getOneByte(uint32_t base, uint8_t byteNumber );
void resetAndStopTimer(void);
void startTimer(void);
uint32_t getDataBytesInMessage(uint8_t index);
void triggerAnotherAquisition(void);
void CustomCodeInterruptHandler(void) ;
uint32_t getFrequencyDivider(uint8_t index) ;
void calculateAverageOverScratchBlocksIfNecessary(void);
void copyScratch0ToCurrentAvgBlockIfNecessary(void);
void calculateAverageOfAveragesIfNecessary(void);
void updateStoredAverages(void) ;
```

```
void updateCurrentAverageBlockCount(void);
//uint8_t getVarStorageIndex(void);
void setupCurrentAverageBlock(void)    ;
void EnableTriggerSquareFunction(void);
/** ****************************************************
 * SerialPort RS232 Interrupt
 * Generated when Transmission finishes
 * Generated when Transmission arrrives
 */
void __interrupt(Intr_RS232_A) Interrupt_RS232(void)
{
  volatile uint32_t tmp32bit = *RS323_SCON_ADR;
  volatile uint32_t tmp32bitshifted = 0;
  volatile uint32_t byteMax32 = 0;
  volatile uint8_t received8bit = 0;
  volatile uint8_t sendOrReceiveRegister8bit = 0;
  volatile uint32_t receivedData32Register = *RS323_SBUF_ADR;




  //send or receive interrupt ?
  tmp32bitshifted = tmp32bit >> 16;
  sendOrReceiveRegister8bit = (tmp32bitshifted & 0xff);




  if((sendOrReceiveRegister8bit & 0x02) != 0)
  {
    // Transmit interrupt
    *RS323_SCON_ADR = RS323_SCON_DEFAULT; // clear RS232
        Interrupt Status
    __mtc0(__mfc0(2) |(0x0001 << Intr_RS232_A ), 2); //clear
        interrupt edge
    switch(State_ProtocolTransmitter)
    {
        case State_SendNothing:
            break;
        case State_SendOpCode:
            State_ProtocolTransmitter =
                State_SendDataBytesInMessageIndex;
            *RS323_SBUF_ADR =
                ProtocolTransmitter_nextOpCodeToReply;
            break;
```

```
case State_SendDataBytesInMessageIndex:
    ByteCounter = 0;
    if(ProtocolTransmitter_nextOpCodeToReply ==
        OpCodeGetConfigBlock)
    {
        *RS323_SBUF_ADR = DataBytesInMessageIndex4Bytes ;
        State_ProtocolTransmitter = State_SendConfigBlock ;
    }
    else if(ProtocolTransmitter_nextOpCodeToReply ==
        OpCodeGetRamData
          || ProtocolTransmitter_nextOpCodeToReply ==
             OpCodeGetAdcData )
    {
        *RS323_SBUF_ADR = DataBytesInMessageIndex;
        State_ProtocolTransmitter = State_SendRamData ;
    }
    else if(ProtocolTransmitter_nextOpCodeToReply ==
        OpCodeMeasure )
    {
        *RS323_SBUF_ADR = DataBytesInMessageIndex256Bytes;
        State_ProtocolTransmitter = State_SendMeasureReply
            ;
    }
    else
    {
        State_ProtocolTransmitter = State_SendNothing;
        ErrorHandler(Error_SwitchState_State_SendDataBytesInMessageIndex);
    }
    break;
case State_SendRamData:
    byteMax32 =
        getDataBytesInMessage(DataBytesInMessageIndex);
    if((ByteCounter ) == (byteMax32))
         State_ProtocolTransmitter = State_SendNothing;
             // nothing left to send...
    else
         sendByteFromRamAuto16bitMode();
    break;
case State_SendMeasureReply:
    if((ByteCounter ) == (256))
    {
         // nothing left to send...
         State_ProtocolTransmitter = State_SendNothing;
         ByteCounter = 0;
    }
```

```
            else
                sendMeasureReply();
            break;
        case State_SendConfigBlock:
             if(ByteCounter == 0 )
                 *RS323_SBUF_ADR = 0;
             else if((ByteCounter == 1 ))
                 *RS323_SBUF_ADR = 0;
             else if((ByteCounter == 2 ))
                 *RS323_SBUF_ADR = 0 ;
             else if((ByteCounter == 3 ))
             {
                *RS323_SBUF_ADR =
                    Status_ActiveClockDiverIndexInitValue   ;
                State_ProtocolTransmitter = State_SendNothing;
             }
             ByteCounter++;
             break;
        default:
            ErrorHandler(Error_SwitchState_ProtocolSenderDefault);
            State_ProtocolTransmitter = State_SendNothing;
            break;
  }//switch( State_ProtocolTransmitter )



}
else if((sendOrReceiveRegister8bit & 0x01) != 0)
{
  // Receive interrupt
  tmp32bitshifted = receivedData32Register >> 8;
  received8bit = (tmp32bitshifted & 0xff);
  isForMatlab = 0;

  *RS323_SCON_ADR = RS323_SCON_DEFAULT; // clear RS232
      Interrupt Status
  __mtc0(__mfc0(2) |(0x0001 << Intr_RS232_A ), 2); //clear
      interrupt edge

  switch( State_ProtocolReceiver )
  {
        case State_WaitForMagicByte:
             if(received8bit == MagicByte)
               State_ProtocolReceiver = State_WaitForOpCode;
               State_ProtocolTransmitter = State_SendNothing;
```

```
      *GeneralPurposeIO = 0;
      *GeneralPurposeIO = 2; // RESET
      *GeneralPurposeIO = 0;
    break;
case State_WaitForOpCode:

    ByteCounter = 0;
    switch(received8bit)
    {
      case OpCodeGetConfigBlock:
        State_ProtocolReceiver = State_WaitForMagicByte;
        ProtocolTransmitter_nextOpCodeToReply =
            OpCodeGetConfigBlock;
        sendMagicByteAndSetSendOpCodeState();
        break;
      case OpCodeSetConfigBlock:
        State_ProtocolReceiver =
            State_WaitForDataBytesInMessageIndex;
        break;
      case OpCodeInitRamWithData:
        State_ProtocolReceiver = State_WaitForMagicByte;
        FillRamWithPredefinedPattern();
        break;
      case OpCodeGetAdcData :
        State_ProtocolReceiver = State_WaitForMagicByte;
        ConfigBlockForCustomCode_OpCode =
            CustomCode_FillRamWithAdcData ;
        Status_ActiveAquisitionMode =
            Mode_AquireOnlyADCData;
        loadFrequencyDividerCounter(
            Status_ActiveClockDiverIndexInitValue);
        currentScratchBlock = 0;
        storedAverages = 0;
        triggerAnotherAquisition();
        break;
      case OpCodeGetAdcDataWithTrigger :
        State_ProtocolReceiver = State_WaitForMagicByte;
        ConfigBlockForCustomCode_OpCode =
            CustomCode_FillRamWithAdcData ;
        Status_ActiveAquisitionMode =
            Mode_AquireADCDataWithTrigger;
        loadFrequencyDividerCounter(
            Status_ActiveClockDiverIndexInitValue);
        currentScratchBlock = 0;
        storedAverages = 0;
```

```
    triggerAnotherAquisition();
    break;
  case
      OpCodeGetAdcDataWithTriggerAndCalcMeanIfNecessary
      :
    State_ProtocolReceiver = State_WaitForMagicByte;
    ConfigBlockForCustomCode_OpCode =
        CustomCode_FillRamWithAdcData ;
    Status_ActiveAquisitionMode =
        Mode_AquireADCDataWithTriggerAndCalcMeanIfNecessary;
    loadFrequencyDividerCounter(
        Status_ActiveClockDiverIndexInitValue);
    currentScratchBlock = 0;
    setupCurrentAverageBlock();
    storedAverages = 0;
    triggerAnotherAquisition();
    break;
  case OpCodeMeasure:
    State_ProtocolReceiver = State_WaitForMagicByte;
    ConfigBlockForCustomCode_OpCode =
        CustomCode_FillRamWithAdcData ;
    Status_ActiveAquisitionMode = Mode_AquireNoise;
    loadFrequencyDividerCounter(
        Status_ActiveClockDiverIndexInitValue);
    currentScratchBlock = 0;
    setupCurrentAverageBlock();
    storedAverages = 0;
    triggerAnotherAquisition();
    break;
  case OpCodeMeasureForMatlab:
    State_ProtocolReceiver = State_WaitForMagicByte;
    ConfigBlockForCustomCode_OpCode =
        CustomCode_FillRamWithAdcData ;
    Status_ActiveAquisitionMode = Mode_AquireNoise;
    loadFrequencyDividerCounter(
        Status_ActiveClockDiverIndexInitValue);
    currentScratchBlock = 0;
    setupCurrentAverageBlock();
    storedAverages = 0;
    isForMatlab = 1;
    triggerAnotherAquisition();
    break;
  case OpCodeOnlyNoise:
    State_ProtocolReceiver = State_WaitForMagicByte;
    ConfigBlockForCustomCode_OpCode =
```

```
        CustomCode_FillRamWithAdcData ;
    Status_ActiveAquisitionMode =
        Mode_AquireNoiseOnly;
    loadFrequencyDividerCounter(
        Status_ActiveClockDiverIndexInitValue);
    currentScratchBlock = 0;
    setupCurrentAverageBlock();
    storedAverages = 0;
    triggerAnotherAquisition();
    break;
case OpCodeOnlyDataHigh:
    State_ProtocolReceiver = State_WaitForMagicByte;
    ConfigBlockForCustomCode_OpCode =
        CustomCode_FillRamWithAdcData ;
    Status_ActiveAquisitionMode =
        Mode_AquireDataHighOnly;
    loadFrequencyDividerCounter(
        Status_ActiveClockDiverIndexInitValue);
    currentScratchBlock = 0;
    setupCurrentAverageBlock();
    storedAverages = 0;
    triggerAnotherAquisition();
    break;

case OpCodeOnlyData:
    State_ProtocolReceiver = State_WaitForMagicByte;
    ConfigBlockForCustomCode_OpCode =
        CustomCode_FillRamWithAdcData ;
    Status_ActiveAquisitionMode =
        Mode_AquireDataFindDecay;
    loadFrequencyDividerCounter(
        Status_ActiveClockDiverIndexInitValue);
    currentScratchBlock = 0;
    setupCurrentAverageBlock();
    storedAverages = 0;
    triggerAnotherAquisition();
    break;
case OpCodeGetRamData:
    State_ProtocolReceiver = State_WaitForMagicByte;
    ProtocolTransmitter_nextOpCodeToReply =
        OpCodeGetRamData;
    sendMagicByteAndSetSendOpCodeState();
    break;
case OpCodeGetMeasureReplyOnly:
    State_ProtocolReceiver = State_WaitForMagicByte;
```

```
              ProtocolTransmitter_nextOpCodeToReply =
                  OpCodeMeasure;
              sendMagicByteAndSetSendOpCodeState();
            break;
          case OpCodeEnableTriggerWithSquareFunction:
            State_ProtocolReceiver = State_WaitForMagicByte;
            EnableTriggerSquareFunction();
            break;
          default:
            ErrorHandler(Error_SwitchState_ProtocolOpcodeDefault)
                ;
            State_ProtocolReceiver = State_WaitForMagicByte;
            break;
      }//switch(tmp8bit)

    break;
  case State_WaitForDataBytesInMessageIndex:
      switch(received8bit)
      {
          case DataBytesInMessageIndex4Bytes:
            State_ProtocolReceiver =
                State_WaitForDataConfigBlock;
            ByteCounter = 0;
            break;
          default:
            ErrorHandler(Error_SwitchState_ProtocolDataLenIndexDefault);
            State_ProtocolReceiver = State_WaitForMagicByte;
            break;
      }
    break;
  case State_WaitForDataConfigBlock:
        // Change also send config block in sendMachine !
        if(ByteCounter == 0 )
        {
            //nothing now = received8bit;
        }
        else if((ByteCounter == 1 ))
        {
            //nothing now = received8bit;
        }
        else if((ByteCounter == 2 ))
        {
            //nothing now = received8bit;
        }
        else if((ByteCounter == 3 ))
```

```
                {

                    received8bit = received8bit & 0x0f ; // use
                        lower 4 bit for new value
                    Status_ActiveClockDiverIndexInitValue  =
                        received8bit;
                    loadFrequencyDividerCounter(Status_ActiveClockDiverIndexInitValue);
                    State_ProtocolReceiver = State_WaitForMagicByte;
                }
                ByteCounter++;

            break;
          default:
            ErrorHandler(Error_SwitchState_ProtocolReceiverDefault);
            State_ProtocolReceiver = State_WaitForMagicByte;
            break;
    }//switch( State_ProtocolReceiver )
  }
  else
  {
    // Error
    *RS323_SCON_ADR = RS323_SCON_DEFAULT; // clear RS232
        Interrupt Status
    __mtc0(__mfc0(2) |(0x0001 << Intr_RS232_A ), 2); //clear
        interrupt edge
    ErrorHandler(Error_NotReceiveAndNotTransmitInterrupt);
  }



}

/** ****************************************************
 * InternalTimer Interrupt / Protocol Timer
 */
void __interrupt(Intr_InternalTimer) InternalTimer(void)
{
  __mtc0(__mfc0(2) |(0x0001 << Intr_InternalTimer ), 2);
      //clear interrupt edge
  //resetAndStopTimer();
  //ErrorHandler(Warn_CustomCodeTimerOverflow); // FIXME should
      be the warnhandler
  WarnHandler(Warn_CustomCodeTimerOverflow);
  triggerAnotherAquisition();
}
```

```
/** ****************************************************
 * BUTTON 0 Interrupt (SW1  X00000)
 */
void __interrupt(Intr_gpio_B) tmpIRQb(void)
{
  __mtc0(__mfc0(2) |(0x0001 << Intr_gpio_B ), 2);    //clear
     interrupt edge


}

/** ****************************************************
 * BUTTON 1 Interrupt (SW1  0X0000)
 */
void __interrupt(Intr_gpio_C) tmpIRQc(void)
{
  __mtc0(__mfc0(2) |(0x0001 << Intr_gpio_C  ), 2); //clear
     interrupt edge
  *LedsAddr = 1;
}

/** ****************************************************
 * BUTTON 2 Interrupt (SW1  00X000)
 */
void __interrupt(Intr_gpio_D) tmpIRQd2(void)
{
  __mtc0(__mfc0(2) |(0x0001 << Intr_gpio_D ), 2); //clear
     interrupt edge
  *RS323_SBUF_ADR = 0x47;
}

/** ****************************************************
 * Interrupt fromCustomCode
 */
void __interrupt(Intr_fromCustomCode) tmpIRQd(void)
{
   if (FreqencyDividerDataIndex >= 2)
        __nop();

   *GeneralPurposeIO = 0; // Prepare for RESET  to custom code
      etc
   *GeneralPurposeIO = 2; // RESET  to custom code etc
   *GeneralPurposeIO = 0; // Release ResetToCustomCode

   // NOP's to avoid Race Conditions with CustomCode
```

```
    StateMachine on slow Clocks
  int WaitFor = getFrequencyDivider(FreqencyDividerDataIndex);

  for(; WaitFor > 1;WaitFor--)
      __nop();

  __mtc0(__mfc0(2) |(0x0001 << Intr_fromCustomCode ),
    2);//clear interrupt edge

  resetAndStopTimer();
  CustomCodeInterruptHandler();

}

void CustomCodeInterruptHandler(void)
{
    volatile uint16_t dataTmp = 0;
    volatile uint8_t  tmp8 = 0;
    volatile unsigned long long tmp = 0;

    #ifdef FLIP_MEMORY
      flipMemory();
    #endif

    if((currentScratchBlock+1) < ScratchBlockCount
        && Status_ActiveAquisitionMode !=  Mode_AquireOnlyADCData
        && Status_ActiveAquisitionMode !=
            Mode_AquireADCDataWithTrigger)
    {
      currentScratchBlock++;
      writeConfigOut();
      triggerAnotherAquisition();
      return;
    }

    if(Status_ActiveAquisitionMode == Mode_AquireNoise
        ||Status_ActiveAquisitionMode == Mode_AquireNoiseOnly )
    {

      calculateAverageOverScratchBlocksIfNecessary();
      copyScratch0ToCurrentAvgBlockIfNecessary();
      updateStoredAverages();
      calculateAverageOfAveragesIfNecessary();
      currentScratchBlock = 0;
      writeConfigOut();
```

```
    DataHighMin = 0;
    DataHighMax = 0;
    DecayStartPosition = 0;

   if(currentAverageBlock == (WholeRamBlockCount-1)) // the
       last one
   {
         calcDataStatistics(NOISE_STATISTICS);
         loadFrequencyDividerCounter(Status_ActiveClockDiverIndexInitValue);
         storedAverages = 0;
         TAU =0;
         if(Status_ActiveAquisitionMode ==
             Mode_AquireNoiseOnly)
         {
           ProtocolTransmitter_nextOpCodeToReply =
               OpCodeMeasure;
           sendMagicByteAndSetSendOpCodeState();
         }
         else
         {
           Status_ActiveAquisitionMode = Mode_AquireDataHigh  ;
           currentScratchBlock = 0;
           setupCurrentAverageBlock();
           storedAverages = 0;
           triggerAnotherAquisition();
         }
    }
    else
    {
      updateCurrentAverageBlockCount();
      triggerAnotherAquisition();
    }
}
else  if(Status_ActiveAquisitionMode == Mode_AquireDataHigh
     || Status_ActiveAquisitionMode ==
    Mode_AquireDataHighOnly)
{
    *GeneralPurposeIO = 0;
    calculateAverageOverScratchBlocksIfNecessary();
    copyScratch0ToCurrentAvgBlockIfNecessary();
    updateStoredAverages();
    calculateAverageOfAveragesIfNecessary();
    currentScratchBlock = 0;
    writeConfigOut();
```

```
    if(currentAverageBlock == (WholeRamBlockCount-1)) // the
        last one
    {
        calcDataStatistics(DATA_HIGH_STATISTICS);
        loadFrequencyDividerCounter(Status_ActiveClockDiverIndexInitValue);
        storedAverages = 0;
        TAU =0;
        if(Status_ActiveAquisitionMode ==
            Mode_AquireDataHighOnly)
        {
          ProtocolTransmitter_nextOpCodeToReply =
              OpCodeMeasure;
          sendMagicByteAndSetSendOpCodeState();
        }
        else
        {
          Status_ActiveAquisitionMode =
              Mode_AquireDataFindDecay ;
          currentScratchBlock = 0;
          setupCurrentAverageBlock();
          storedAverages = 0;
          triggerAnotherAquisition();
        }
    }
    else
    {
      updateCurrentAverageBlockCount();
      triggerAnotherAquisition();
    }
}
else if(Status_ActiveAquisitionMode ==
    Mode_AquireDataFindDecay          )
{
    // Operates only on Scratch Block Data, to perform fast
        enough, because sample Down would take too long.
    // So no further Averaging with use of
        averageStorageBlocks will be done here.
    calculateAverageOverScratchBlocksIfNecessary();
    copyScratch0ToCurrentAvgBlockIfNecessary();
    updateStoredAverages();
    calculateAverageOfAveragesIfNecessary();
    currentScratchBlock = 0;
    writeConfigOut();
```

```
    if(currentAverageBlock == (WholeRamBlockCount-1) ||
        currentAverageBlock == -1)
    {
            //updateCurrentAverageBlockCount(); will be done
                after last byte was sent !!! do not before
                sending
            calcDataStatistics(DATA_TEMP_STATISTICS);
            tmp8 = sampledDownEnough();
            if(tmp8 == 0 && FreqencyDividerDataIndex !=
                FreqencyDividerDataMaxValue && isForMatlab ==
                0 )
            {    // not enough AND more can be done
                 loadFrequencyDividerCounter(++FreqencyDividerDataIndex);
                 currentScratchBlock = 0;
                 setupCurrentAverageBlock();
                 storedAverages = 0;
                 triggerAnotherAquisition();
            }
            else
            {
                 Stage0PreProcessing();
                 calcDataStatistics(DATA_TEMP_STATISTICS);
                 findDecayStart();
                 calcTau();
                 Status_ActiveAquisitionMode =
                     Mode_AquireDataFindDecay;
                 currentScratchBlock = 0;
                 //updateCurrentAverageBlockCount(); will be
                     done after last byte was sent...
                 ProtocolTransmitter_nextOpCodeToReply =
                     OpCodeMeasure;
                 sendMagicByteAndSetSendOpCodeState();
            }
    }
    else
    {
         updateCurrentAverageBlockCount();
         triggerAnotherAquisition();
    }




}
else if(Status_ActiveAquisitionMode ==
```

```
        Mode_AquireADCDataWithTriggerAndCalcMeanIfNecessary)
         {
            calculateAverageOverScratchBlocksIfNecessary();
            copyScratch0ToCurrentAvgBlockIfNecessary();
            updateStoredAverages();
            calculateAverageOfAveragesIfNecessary();

            if(storedAverages == (WholeRamBlockCount -
                ScratchBlockCount))
            {
                ProtocolTransmitter_nextOpCodeToReply =
                    OpCodeGetAdcData;
                currentScratchBlock = 0;
                sendMagicByteAndSetSendOpCodeState();
            }
            else
            {
                currentScratchBlock = 0;
                updateCurrentAverageBlockCount();
                triggerAnotherAquisition();
            }
         }
    else if(Status_ActiveAquisitionMode ==
        Mode_AquireOnlyADCData
              || Status_ActiveAquisitionMode ==
                 Mode_AquireADCDataWithTrigger    )
    {
         ProtocolTransmitter_nextOpCodeToReply = OpCodeGetAdcData;
         currentScratchBlock = 0;
         sendMagicByteAndSetSendOpCodeState();
    }
    else
        ErrorHandler(Error_WrongStatusMode);


}




void triggerAnotherAquisition(void)
{
      // start another aquisition in the next ram block
      volatile int i = 0;
      int divider =
```

```
    getFrequencyDivider(FreqencyDividerDataIndex)*11;

writeConfigOut();
*GeneralPurposeIO = 0;
*GeneralPurposeIO = 2; // RESET  to custom code etc
*GeneralPurposeIO = 0;
if(simulationMode == 0)
{
      if(Status_ActiveAquisitionMode == Mode_AquireNoise
      || Status_ActiveAquisitionMode == Mode_AquireNoiseOnly)
      {
           *GeneralPurposeIO = 0;
           *GeneralPurposeIO = 1; // interrupt to custom code
           *GeneralPurposeIO = 0;
      }
      else if(Status_ActiveAquisitionMode ==
          Mode_AquireDataFindDecay
            || Status_ActiveAquisitionMode ==
                Mode_AquireADCDataWithTrigger
            || Status_ActiveAquisitionMode ==
                Mode_AquireADCDataWithTriggerAndCalcMeanIfNecessary
                )
      {
           divider = divider *TriggerOutMultiplicator ;
           for(i = 0;i < divider ; i++)
                   *GeneralPurposeIO = 4; // trigger Light
                       Source, falling edge
           // 200ms are about 250000 iterations @125MHz (Base
               Clock, not CPU Clock !)
           // so 1250 Iterations per ms @125MHz (Base Clock,
               not CPU Clock !)
           // for calculation see
               decayTimeTriggerEnabledTime.xls

           for(i = 0;i < FreqencyDividerDataIndex ; i++)
               *GeneralPurposeIO = 1+4; // interrupt to custom
                   code        + trigger LED
           *GeneralPurposeIO = 1; // interrupt to custom code
           *GeneralPurposeIO = 0;
      }
      else if(Status_ActiveAquisitionMode ==
          Mode_AquireOnlyADCData        )
      {

           *GeneralPurposeIO = 1; // interrupt to custom code
```

```
            *GeneralPurposeIO = 0;
        }
        else if(Status_ActiveAquisitionMode ==
            Mode_AquireDataHigh
        || Status_ActiveAquisitionMode ==
            Mode_AquireDataHighOnly )
        {
            for(i = 0;i < 1024 ; i++)
                *GeneralPurposeIO = 4; // trigger led  long
                    enough so sensor can go to high state
            *GeneralPurposeIO = 1+4; // interrupt to custom
                code + led
            *GeneralPurposeIO = 4;
        }
        else
        {
            ErrorHandler(Error_WrongActiveAquisitionMode);
        }
        if (enableCustomCodeWatchdog != 0)
            startTimer() ;
    }
    else
    {       // Simulation Mode

        if(Status_ActiveAquisitionMode == Mode_AquireNoise      )
        {
            debugFillRamWithNoise();
            CustomCodeInterruptHandler();
        }
        else if(Status_ActiveAquisitionMode ==
            Mode_AquireDataFindDecay
              || Status_ActiveAquisitionMode ==
                  Mode_AquireOnlyADCData      )
        {
            debugFillRamWithData();
            CustomCodeInterruptHandler();
        }
        else
        {
            ErrorHandler(Error_WrongActiveAquisitionMode);
        }
    }

}
```

```
/** ****************************************************
 * MAIN Function
 *
 */
void main( void )
{



  init();


 for ( ;; )
 {
  //(*leds)++;
  for ( int delay = 0; delay < 5000000; delay++ )
  {
    __nop(); // Two underscores
  }
 }
}

/** ****************************************************
 * INIT Function
 *
 */
void init(void)
{
  *LedsAddr = 0;
  uint8_t tmp = 0;




  *LedsAddr = 0;
  for ( int delay = 0; delay < 10; delay++ )
  {
     for(int delay2 = 0; delay2 < 1000*300;delay2++)
             __nop(); // Two underscores
     *LedsAddr = 0x01 << delay ;
  }
  *LedsAddr = 0x00;
  //FillRamWithPredefinedPattern();
```

```
State_ProtocolReceiver    = State_WaitForMagicByte;
State_ProtocolTransmitter = State_SendNothing;
ByteCounter = 0;
Status_ActiveAquisitionMode = Mode_AquireNoise;

// READ CONFIG FROM EXTERNAL MEMORY

Status_ActiveClockDiverIndexInitValue =
    FreqencyDividerDataInitialValue;
loadFrequencyDividerCounter(Status_ActiveClockDiverIndexInitValue);




ConfigBlockForCustomCode_OpCode = 255;
ConfigBlockForCustomCode_StartAddress = 0;
ConfigBlockForCustomCode_LastAddress = 0;
writeConfigOut();



// send reset to customCode
*GeneralPurposeIO = 0;
*GeneralPurposeIO = 2; // RESET
*GeneralPurposeIO = 0;


{
   TAU =0;
    NoiseMax = 0;
    NoiseMin = 0;
}
DataHighMax = 0;
DataHighMin = 0;


LastError = Error_NoError;




// ********** INTERRUPTS **************
// Enable interrupts
```

```
  __mtc0(__mfc0(0) | 0x0001, 0);


  __mtc0(__mfc0(1) |(0x0001 << Intr_gpio_A), 1); // Enable xxx
      interrupt
  __mtc0(__mfc0(9) |(0x0001 << Intr_gpio_A), 9); // set xxx
      interrupt to edge
  __mtc0(__mfc0(1) |(0x0001 << Intr_gpio_B), 1);  // Enable xxx
      interrupt
  __mtc0(__mfc0(9) |(0x0001 << Intr_gpio_B), 9); // set xxx
      interrupt to edge
  __mtc0(__mfc0(1) |(0x0001 << Intr_gpio_C), 1); // Enable xxx
      interrupt
  __mtc0(__mfc0(9) |(0x0001 << Intr_gpio_C), 9); // set xxx
      interrupt to edge
  __mtc0(__mfc0(1) |(0x0001 << Intr_gpio_D), 1); // Enable xxx
      interrupt
  __mtc0(__mfc0(9) |(0x0001 << Intr_gpio_D), 9); // set xxx
      interrupt to edge


  initCommunication();

  if (sizeof(TAU) != 8)
     ErrorHandler(Error_WrongDataTypeSize);

  if(ScratchBlockCount > WholeRamBlockCount)
     ErrorHandler(Error_ConfigErrorMemorySettings);

  resetAndStopTimer();
}


// ********** Protocol Timer Watchdog **************
void resetAndStopTimer()
{

  __mtc0(__mfc0(0) &(~(0x0001 << 8)), 0); // Clear ITE
  __mtc0(__mfc0(0) |(0x0001 << 7)    , 0); //   Set ITR
  __mtc0(__mfc0(0) &(~(0x0001 << 7)), 0); // Clear ITR

  //      0x005F5E10 = 0d6.250.000 Cycles = 500ms at 12,5MHz
  __mtc0(0x005F5E10 , 5); // Write to PIT Register
```

```
  __mtc0(__mfc0(1) |(0x0001 << Intr_InternalTimer), 1); //
      Enable xxx interrupt
  __mtc0(__mfc0(9) |(0x0001 << Intr_InternalTimer), 9); // set
      xxx interrupt to edge

}

void startTimer()
{
   __mtc0(__mfc0(0) |(0x0001 << 8)    , 0); //    Set ITE
}



/** ****************************************************
 * Error Handler
 * Maybe Port IO Output
 */
void ErrorHandler(uint8_t errorType)
{
  resetAndStopTimer();

  LastError = errorType;
  *LedsAddr = 0xFFFFFFFF & LastError;
  for ( ;; )
  {
    __nop(); // Two underscores
  }
}
/** ****************************************************
 * Error Handler
 * Maybe Port IO Output
 */
void WarnHandler(uint8_t warnType)
{
  resetAndStopTimer();

  *LedsAddr = 0xFFFFFFFF & warnType;
  //FIXME
  for ( ;; )
  {
    __nop(); // Two underscores
  }
  uint32_t tmp32 = 0;
```

```
    while(tmp32 == 0)
    {
        tmp32 = *LedsAddr ;
        tmp32 = tmp32 >> 19;
        tmp32 = tmp32 & 0xFF ;
    }
    *LedsAddr = 0;
}

/** *****************************************************
 * Takes a 32 bit = 4*8bit = 4 Byte Value : "base"
 * Returns the "byteNumber" byte.
 * [(byte 3)(byte 2)(byte 1)(byte 0)]
 */
uint8_t getOneByte(uint32_t base, uint8_t byteNumber )
{
    volatile uint8_t retVal = 0;
    if(byteNumber > 3)
        ErrorHandler(Error_SwitchState_getOneByte);

    retVal = ((base >> (byteNumber*8)) & 0xff);
    return retVal;

}
```

```
uint32_t getDataBytesInMessage(uint8_t index)
{
    switch(index)
    {
        case DataBytesInMessageIndexEmpty:
            return 0;
        case DataBytesInMessageIndex4Bytes:
            return 4;
        case DataBytesInMessageIndex32768Bytes:
```

```
            return 32768;
        case DataBytesInMessageIndex2048Bytes:
            return 2048;
        case DataBytesInMessageIndex8Bytes:
            return 8;
        default:
            ErrorHandler(Error_SwitchState_getDataBytesInMessage);
    }
    return 0;
}



uint32_t getFrequencyDivider(uint8_t index)
{
 switch(index)
    {
        case 0:
          return 1;
        case 1:
          return 2;
        case 2:
          return 4;
        case 3:
          return 8;
        case 4:
          return 16;
        case 5:
          return 32;
        case 6:
          return 64;
        case 7:
          return 128;
        case 8:
          return 256;
        case 9:
          return 512;
        case 10:
          return 1024;

    }
    return 0;
}
```

```c
void calculateAverageOverScratchBlocksIfNecessary(void)
{
    uint32_t i,j;
    uint32_t tmp, dataTmp ;
    if(ScratchBlockCount > 1)
    {
        for(int j = 0; j < DataPointsPerRamBlock;j++)
        {
            tmp = 0;
            for(int i = 0; i  < ScratchBlockCount ; i++)
            {
                dataTmp =
                    get16BitWordFromRam((DataPointsPerRamBlock*i)+j);
                tmp += dataTmp;
            }
            tmp = tmp/ScratchBlockCount;
            write16BitWordToRam(j,(uint16_t)(tmp & 0xffff));
        }
    }
}

void copyScratch0ToCurrentAvgBlockIfNecessary(void)
{
    uint32_t i,j;
    uint16_t dataTmp ;
    if(ScratchBlockCount < WholeRamBlockCount)
    {
        for(int j = 0; j < DataPointsPerRamBlock;j++)
        {
            dataTmp = get16BitWordFromRam(j);
            write16BitWordToRam(
                (DataPointsPerRamBlock*currentAverageBlock)+j
                ,dataTmp );
        }
    }
}

void updateStoredAverages(void)
{
    if(storedAverages < (WholeRamBlockCount - ScratchBlockCount)
        || storedAverages == 0)
        storedAverages++;
}
void calculateAverageOfAveragesIfNecessary(void)
```

```
{
    uint32_t i,j;
    uint32_t tmp,dataTmp ;
    if(storedAverages > 1)
    {
        for(int j = 0; j < DataPointsPerRamBlock;j++)
        {
            tmp = 0;
            for(int i = ScratchBlockCount; i <
                (storedAverages+ScratchBlockCount) ; i++)
            {
                dataTmp =
                    get16BitWordFromRam((DataPointsPerRamBlock*i)+j);
                tmp += dataTmp;
            }
            tmp = tmp/storedAverages;
            write16BitWordToRam(j,(uint16_t)(tmp & 0xffff));     //
                Store Result in Scratch Block 0
        }
    }
}
void updateCurrentAverageBlockCount(void)
{
    if(currentAverageBlock == -1)
        return;
    else if(currentAverageBlock == WholeRamBlockCount-1)
        currentAverageBlock = ScratchBlockCount;
    else
        currentAverageBlock++;



}


/*uint8_t getVarStorageIndex()
{
    int tmp = currentAverageBlock - ScratchBlockCount;
    if(currentAverageBlock == -1)
        return 0;
    if(tmp > valuesToStore || tmp < 0)
        ErrorHandler(Error_getVarStorageIndex);
    return (uint8_t)tmp;
} */
```

```
void setupCurrentAverageBlock(void)
{
    if(WholeRamBlockCount == ScratchBlockCount)
        currentAverageBlock = -1;
    else
        currentAverageBlock = ScratchBlockCount;
}


void EnableTriggerSquareFunction(void)
{
      volatile int i ;
      while(1)
      {
         for(i = 0;i <
             TriggerEnableTimeMicroSecondsForSquareOutput ; i++)
            *GeneralPurposeIO = 4; // trigger Light Source,
                falling edge

         for(i = 0;i <
             TriggerEnableTimeMicroSecondsForSquareOutput ; i++)
            *GeneralPurposeIO = 0; // trigger Light Source,
                falling edge

      }
}
```

**Algorithm.c**

Listing 3: Mikrocontroller Code

```
#include "Algorithm.h"



/**
 * \brief Calculates Tau from Decay Data
 *         So called Stage 2 in Matlab Code
 * \return     nothing
 */
void calcTau(void)
{
   uint16_t  dataTmp = 0;
   double TauTemp, Tau2 = 0.0;
   double expDecay  = 0.0;
   uint16_t divider =
      getFrequencyDivider(FreqencyDividerDataIndex);
   double deltaTime =
      1.0/(BaseSampleFrequencyKHz*1000/(divider));
   uint32_t i, j = 0;
   uint32_t NoiseDelta = NoiseMax - NoiseMin;

   TAU = 0;
   if(isForMatlab == 1)
      return ;

   #ifdef PT_CDIV32
   //  s2Fit1e_DecayStart
       expDecay  = 1.0;
       uint16_t maxVal =
          get16BitWordFromRam(DecayStartPosition);
       double expVal = exp(-expDecay) * (maxVal-DataTempMin);
       expVal = expVal + DataTempMin;

       for(i = DecayStartPosition; i < DataPointsPerRamBlock;
          i++)
       {
          dataTmp = get16BitWordFromRam(i);
          if(dataTmp < expVal)
```

```
        {
            TauTemp = i;
            break;
        }
    }

    TauTemp = (TauTemp-DecayStartPosition)*deltaTime;
    TAU = TauTemp / expDecay;


#endif
#ifdef PT_CDIV64
//  s2Fit1e_DecayStart
    expDecay  = 1.0;
    uint16_t maxVal =
        get16BitWordFromRam(DecayStartPosition);
    double expVal = exp(-expDecay) * (maxVal-DataTempMin);
    expVal = expVal + DataTempMin;

    for(i = DecayStartPosition; i < DataPointsPerRamBlock;
        i++)
    {
        dataTmp = get16BitWordFromRam(i);
        if(dataTmp < expVal)
        {
            TauTemp = i;
            break;
        }
    }

    TauTemp = (TauTemp-DecayStartPosition)*deltaTime;
    TAU = TauTemp / expDecay;


#endif


#ifdef PtFPP_CDIV32
//  s2Fit1e_DecayStart
    expDecay  = 1.0;
    uint16_t maxVal =
        get16BitWordFromRam(DecayStartPosition);
    double expVal = exp(-expDecay) * (maxVal-DataTempMin);
    expVal = expVal + DataTempMin;
```

```
    for(i = DecayStartPosition; i < DataPointsPerRamBlock;
        i++)
    {
         dataTmp = get16BitWordFromRam(i);
         if(dataTmp < expVal)
         {
               TauTemp = i;
               break;
         }
    }

    TauTemp = (TauTemp-DecayStartPosition)*deltaTime;
    TAU = TauTemp / expDecay;

#endif
#ifdef PtFPP_CDIV64
//  s2Fit1e_top_DecayStart
    expDecay  = 1.0;
    uint16_t maxVal =
        get16BitWordFromRam(DecayStartPosition);
    double expVal = exp(-expDecay) *
        (maxVal-DataTempMin+NoiseDelta);
    expVal = expVal + DataTempMin;

    for(i = DecayStartPosition; i < DataPointsPerRamBlock;
        i++)
    {
         dataTmp = get16BitWordFromRam(i);
         if(dataTmp < expVal)
         {
               TauTemp = i;
               break;
         }
    }

    TauTemp = (TauTemp-DecayStartPosition)*deltaTime;
    TAU = TauTemp / expDecay;

#endif
#ifdef PdFPP_CDIV512
  //  s2Fit1e2x_DecayStart
    expDecay  = 1.0;
    uint16_t maxVal =
        get16BitWordFromRam(DecayStartPosition);
    double expVal = exp(-expDecay) * (maxVal-DataTempMin);
```

```
    expVal = expVal + DataTempMin;

    for(i = DecayStartPosition; i < DataPointsPerRamBlock;
        i++)
    {
        dataTmp = get16BitWordFromRam(i);
        if(dataTmp < expVal)
        {
            TauTemp = i;
            j = i;
            break;
        }
    }

    TauTemp = (TauTemp-DecayStartPosition)*deltaTime;
    TAU = TauTemp / expDecay;


    expDecay  = 1.0;
    maxVal =      get16BitWordFromRam(j  );
    expVal = exp(-expDecay) * (maxVal-DataTempMin);
    expVal = expVal + DataTempMin;

    for(i = DecayStartPosition; i < DataPointsPerRamBlock;
        i++)
    {
        dataTmp = get16BitWordFromRam(i);
        if(dataTmp < expVal)
        {
            TauTemp = i;
            break;
        }
    }

    TauTemp = (TauTemp-j  )*deltaTime;
    TauTemp = TauTemp / expDecay;

    TAU = (TAU + TauTemp) / 2.0;
#endif
#ifdef PdFPP_CDIV1024
  //  s2Fit2x1eDecayStart
    expDecay  = 1.0;
    uint16_t maxVal =
        get16BitWordFromRam(DecayStartPosition);
    double expVal = exp(-expDecay) * (maxVal-DataTempMin);
```

```
    expVal = expVal + DataTempMin;

    for(i = DecayStartPosition; i < DataPointsPerRamBlock;
        i++)
    {
        dataTmp = get16BitWordFromRam(i);
        if(dataTmp < expVal)
        {
            TauTemp = i;
            break;
        }
    }

    TauTemp = (TauTemp-DecayStartPosition)*deltaTime;
    TAU = TauTemp / expDecay;


    expDecay  = 2.0;
    maxVal =      get16BitWordFromRam(DecayStartPosition );
    expVal = exp(-expDecay) * (maxVal-DataTempMin);
    expVal = expVal + DataTempMin;

    for(i = DecayStartPosition; i < DataPointsPerRamBlock;
        i++)
    {
        dataTmp = get16BitWordFromRam(i);
        if(dataTmp < expVal)
        {
            TauTemp = i;
            break;
        }
    }

    TauTemp = (TauTemp-DecayStartPosition )*deltaTime;
    TauTemp = TauTemp / expDecay;

    TAU = (TAU + TauTemp) / 2.0;
#endif
#ifdef RuDPP_CDIV4
//  s2Fit2e_DecayStart
    expDecay  = 2.0;
    uint16_t maxVal =
        get16BitWordFromRam(DecayStartPosition);
    double expVal = exp(-expDecay) * (maxVal-DataTempMin);
    expVal = expVal + DataTempMin;
```

```
    for(i = DecayStartPosition; i < DataPointsPerRamBlock;
        i++)
    {
        dataTmp = get16BitWordFromRam(i);
        if(dataTmp < expVal)
        {
            TauTemp = i;
            break;
        }
    }

    TauTemp = (TauTemp-DecayStartPosition)*deltaTime;
    TAU = TauTemp / expDecay;

#endif
#ifdef RuDPP_CDIV8
  // s2Fit2x1eDecayStart
    expDecay  = 1.0;
    uint16_t maxVal =
        get16BitWordFromRam(DecayStartPosition);
    double expVal = exp(-expDecay) * (maxVal-DataTempMin);
    expVal = expVal + DataTempMin;

    for(i = DecayStartPosition; i < DataPointsPerRamBlock;
        i++)
    {
        dataTmp = get16BitWordFromRam(i);
        if(dataTmp < expVal)
        {
            TauTemp = i;
            break;
        }
    }

    TauTemp = (TauTemp-DecayStartPosition)*deltaTime;
    TAU = TauTemp / expDecay;


    expDecay  = 2.0;
    maxVal =     get16BitWordFromRam(DecayStartPosition );
    expVal = exp(-expDecay) * (maxVal-DataTempMin);
    expVal = expVal + DataTempMin;

    for(i = DecayStartPosition; i < DataPointsPerRamBlock;
```

```
        i++)
    {
        dataTmp = get16BitWordFromRam(i);
        if(dataTmp < expVal)
        {
            TauTemp = i;
            break;
        }
    }

    TauTemp = (TauTemp-DecayStartPosition )*deltaTime;
    TauTemp = TauTemp / expDecay;

    TAU = (TAU + TauTemp) / 2.0;

#endif
#ifdef Test_CDIV0
  // s2Fit2x1eDecayStart
    expDecay  = 1.0;
    uint16_t maxVal =
        get16BitWordFromRam(DecayStartPosition);
    double expVal = exp(-expDecay) * (maxVal-DataTempMin);
    expVal = expVal + DataTempMin;

    for(i = DecayStartPosition; i < DataPointsPerRamBlock;
        i++)
    {
        dataTmp = get16BitWordFromRam(i);
        if(dataTmp < expVal)
        {
            TauTemp = i;
            break;
        }
    }

    TauTemp = (TauTemp-DecayStartPosition)*deltaTime;
    TAU = TauTemp / expDecay;


    expDecay  = 2.0;
    maxVal =      get16BitWordFromRam(DecayStartPosition );
    expVal = exp(-expDecay) * (maxVal-DataTempMin);
    expVal = expVal + DataTempMin;

    for(i = DecayStartPosition; i < DataPointsPerRamBlock;
```

```
        i++)
    {
        dataTmp = get16BitWordFromRam(i);
        if(dataTmp < expVal)
        {
            TauTemp = i;
            break;
        }
    }

    TauTemp = (TauTemp-DecayStartPosition )*deltaTime;
    TauTemp = TauTemp / expDecay;

    TAU = (TAU + TauTemp) / 2.0;

  #endif

  // INSERT NEW ALGORITHM INFORMATION HERE (calc Tau Impl)

}


/**
 * \brief  Data PreProcessesing Stage 0
 * \return nothing
 */



void Stage0PreProcessing(void)
{



  // INSERT NEW ALGORITHM INFORMATION HERE(FilterData)

  #ifdef enableStage0

  volatile uint32_t i,j, currentIndex;
  volatile uint32_t tmp32a, tmp32b;
  volatile double tmpDouble0, tmpDouble1 = 0;

  if(isForMatlab == 1)
      return ;
```

```
  if (ScratchBlockCount < 2 )
     ErrorHandler(Error_NotEnoughScratchBlocksForStage0);


 // generic Filter Implementation ! do not modify
 for(i=stage0filtersize-1 ; i < DataPointsPerRamBlock; i++)
 {
    tmpDouble1 = 0;
    for(j=0; j < stage0filtersize; j++)
    {
       currentIndex = i - stage0filtersize + j+1;

       tmpDouble0 = (double)get16BitWordFromRam(currentIndex);
       tmpDouble0 = (tmpDouble0 * stage0filter[j]) ;
       tmpDouble1 += tmpDouble0;


    }
    tmp32b = ceil(tmpDouble1);
    // If Error -> Possible Filter Error -> Check Filter Data
    if (tmp32b > uint16MaxValue)
      ErrorHandler(Error_uint16overflowStage0);

    write16BitWordToRam(DataPointsPerRamBlock + i
        ,(uint16_t)tmp32b);
 }
 // fill first samples with first filtered one
 tmp32a =
     get16BitWordFromRam(DataPointsPerRamBlock+stage0filtersize-1);
 for(i=0; i < stage0filtersize; i++)
          write16BitWordToRam(DataPointsPerRamBlock+i,(uint16_t)tmp32a);

 // move data back to scratchBlock0
 for(i=0 ; i < DataPointsPerRamBlock; i++)
 {
    tmp32a = get16BitWordFromRam(i+DataPointsPerRamBlock);
    write16BitWordToRam(i,(uint16_t)tmp32a);
 }

 #endif
}
```

```
/**
 * \brief Tests if the last part of the Decay Data is within
    Noise Range
 *        and if first part of Data is above Noise Range
 * \return     0 NOT enoughSampleValuesAboveNoiseLevel
 *             1 enoughSampleValuesAboveNoiseLevel
 */
uint8_t sampledDownEnough()
{

    volatile uint32_t i = 0;
    //uint32_t dataPointsWhichShouldBeAbove =
        ceil((double)howMuchDataPointsInPercentShouldBeAbove*(double)DataPointsPerRam
    //uint32_t dataPointsWhichShouldBeWithinNoiseLevel =
        ceil((double)howMuchDataPointsInPercentShouldBeWithinNoiseRange*(double)DataP
    volatile uint16_t tmp16 = 0;
    //volatile uint32_t cntAbove = 0;
    //volatile uint32_t cntWithin = 0;
    //uint32_t howMuchErrorPercentAreAllowedTmp = ceil(1.0 +
        ((double)howMuchErrorPercentAreAllowed/100.0));
    uint32_t NoiseDelta = NoiseMax - NoiseMin;
    //uint32_t DataHighDelta = DataHighMax - DataHighMin;
    uint32_t InnerDelta = DataHighMin - NoiseMax;
    volatile uint32_t level, tmp32 = 0;
    //uint8_t ret0 = 0;
    //uint8_t ret1 = 0;

    if((DataTempMax-DataTempMin) < (NoiseDelta*2)) // Quick Test
        return 0;

    if((DataTempMax-DataTempMin) < InnerDelta) // Quick Test
        return 0;


    tmp32 = 0;
    for(i=0; i < 20 ; i++)
        tmp32 = tmp32 + get16BitWordFromRam(i);
    tmp32 = ceil(tmp32 / 20.0);

    level = 0;
    for(i=0; i < 20 ; i++)
        level = level +
            get16BitWordFromRam(DataPointsPerRamBlock-i);
    level = ceil(level / 20.0);
    level = tmp32 - level;
```

```
    if(level < InnerDelta)
        return 0;

    return 1;


}


/**
 * \brief Finds Position in Data, at which the Decay Curve is
     starting.
 * Starts searching from last Datapoint.
 * \return
 */
void findDecayStart()
{
    volatile int i = 0;
    volatile uint16_t tmp16 = 0;
    uint16_t noiseDelta = NoiseMax - NoiseMin;

    DecayStartPosition = DataPointsPerRamBlock-1;;

    if(isForMatlab == 1)
         return ;

    #ifdef  PT_CDIV32
     // FindDecayStart1
        uint16_t level = DataTempMax - ceil(noiseDelta/2) ;
        DecayStartPosition = 0;
        for(i = DataPointsPerRamBlock-1 ;i >=0; i--)
        {
            tmp16 = get16BitWordFromRam(i);
            if(tmp16 > level)
            {
                DecayStartPosition = i;
                break;
            }
        }

    #endif
    #ifdef  PT_CDIV64
     // FindDecayStart1
        uint16_t level = DataTempMax - ceil(noiseDelta/2) ;
```

```
    DecayStartPosition = 0;
    for(i = DataPointsPerRamBlock-1 ;i >=0; i--)
    {
        tmp16 = get16BitWordFromRam(i);
        if(tmp16 > level)
        {
            DecayStartPosition = i;
            break;
        }
    }

#endif


 #ifdef PtFPP_CDIV32
 // FindDecayStart0
    uint16_t level =0 ;
    level = DataTempMax - (noiseDelta) ;
    DecayStartPosition = 0;
    for(i = DataPointsPerRamBlock-1 ;i >=0; i--)
    {
        tmp16 = get16BitWordFromRam(i);
        if(tmp16 > level)
        {
            DecayStartPosition = i;
            break;
        }
    }
 #endif
 #ifdef PtFPP_CDIV64
 // FindDecayStart0
    uint16_t level =0 ;
    level = DataTempMax - (noiseDelta) ;
    DecayStartPosition = 0;
    for(i = DataPointsPerRamBlock-1 ;i >=0; i--)
    {
        tmp16 = get16BitWordFromRam(i);
        if(tmp16 > level)
        {
            DecayStartPosition = i;
            break;
        }
    }

 #endif
```

```
#ifdef PdFPP_CDIV512
// FindDecayStart1
  uint16_t level = DataTempMax - ceil(noiseDelta/2) ;
  DecayStartPosition = 0;
  for(i = DataPointsPerRamBlock-1 ;i >=0; i--)
  {
      tmp16 = get16BitWordFromRam(i);
      if(tmp16 > level)
      {
         DecayStartPosition = i;
         break;
      }
  }
#endif
#ifdef PdFPP_CDIV1024
// FindDecayStart1
  uint16_t level = DataTempMax - ceil(noiseDelta/2) ;
  DecayStartPosition = 0;
  for(i = DataPointsPerRamBlock-1 ;i >=0; i--)
  {
      tmp16 = get16BitWordFromRam(i);
      if(tmp16 > level)
      {
         DecayStartPosition = i;
         break;
      }
  }
#endif
#ifdef RuDPP_CDIV4
// FindDecayStart1
  uint16_t level = DataTempMax - ceil(noiseDelta/2) ;
  DecayStartPosition = 0;
  for(i = DataPointsPerRamBlock-1 ;i >=0; i--)
  {
      tmp16 = get16BitWordFromRam(i);
      if(tmp16 > level)
      {
         DecayStartPosition = i;
         break;
      }
  }
#endif
#ifdef RuDPP_CDIV8
// FindDecayStart0
```

```
    uint16_t level =0 ;
    level = DataTempMax - (noiseDelta) ;
    DecayStartPosition = 0;
    for(i = DataPointsPerRamBlock-1 ;i >=0; i--)
    {
        tmp16 = get16BitWordFromRam(i);
        if(tmp16 > level)
        {
            DecayStartPosition = i;
            break;
        }
    }
  #endif
  #ifdef Test_CDIV0
  // FindDecayStart0
    uint16_t level =0 ;
    level = DataTempMax - (noiseDelta) ;
    DecayStartPosition = 0;
    for(i = DataPointsPerRamBlock-1 ;i >=0; i--)
    {
        tmp16 = get16BitWordFromRam(i);
        if(tmp16 > level)
        {
            DecayStartPosition = i;
            break;
        }
    }
  #endif
  // INSERT NEW ALGORITHM INFORMATION HERE (FindDecayStart
      Impl)
}



/**
 * \brief Calculates Min Max Values
 * storage NOISE_STATISTICS
 * storage DATA_HIGH_STATISTICS
 * storage DATA_TEMP_STATISTICS
 */
void calcDataStatistics(uint8_t storage)
{
  volatile uint16_t  dataTmp = 0;
  volatile uint16_t  tmpMin = uint16MaxValue;
  volatile uint16_t  tmpMax = 0 ;
```

```
  for(int j = 0; j < DataPointsPerRamBlock;j++)
  {
    dataTmp = get16BitWordFromRam(j);
    if(dataTmp > tmpMax )
    {
       tmpMax = dataTmp;
       DataMaxPosition = j;
    }
    if(dataTmp < tmpMin)
       tmpMin = dataTmp;
  }
  switch(storage)
  {
     case NOISE_STATISTICS:
         NoiseMax = tmpMax;
         NoiseMin = tmpMin;
         break;
     case DATA_HIGH_STATISTICS:
         DataHighMax = tmpMax;
         DataHighMin = tmpMin;
         break;
     case DATA_TEMP_STATISTICS:
         DataTempMax = tmpMax;
         DataTempMin = tmpMin;
         break;
     default:
         ErrorHandler(Error_WrongStorageMode);

  }
}

/**
 * \brief For DEBUG Use Only
 *        Fills Ram with predefined Noise
 *        Noise should be within Range [0 ... about 65500]
 */
void debugFillRamWithNoise(void)
{
   for(int j = 0; j < DataPointsPerRamBlock;j++)
   {
     write16BitWordToRam(
         (DataPointsPerRamBlock*currentScratchBlock)+j ,
         (DataPointsPerRamBlock*currentScratchBlock)+j);
   }
}
```

```
/**
 * \brief For DEBUG Use Only
 *        Fills Ram with predefined Decay Data
 *        Data should be within Range [0 ... about 65500]
 */
void debugFillRamWithData(void)
{
    volatile double tmp = 0.0 ;
    volatile uint16_t tmp16 = 0;

    for(int j = 0; j < DataPointsPerRamBlock;j++)
    {
      // Tau = 3.84607E-7 bei 130 MHz    = (1/50)*(1/130.000.000)
      // tmp = - ((j*(1+FreqencyDividerDataIndex))/(50.0)) ;//
           Tau = 3.84607E-7 bei 130 MHz
      //tmp = - ((j*(1+FreqencyDividerDataIndex))/(50.0*100)) ;
      //tmp = exp(tmp );
      //tmp = tmp * 50000;
      //tmp16 = (uint16_t)  tmp;
      //write16BitWordToRam(
           (DataPointsPerRamBlock*currentScratchBlock)+j   ,tmp16 );

      write16BitWordToRam(
           (DataPointsPerRamBlock*currentScratchBlock)+j ,10+ (
           (DataPointsPerRamBlock*currentScratchBlock)+j));
    }
}
```

**Communication.c**

Listing 4: Mikrocontroller Code

```
#include "Communication.h"

//** ****************************************************
// RS232 Serial Port Memory
uint32_t * const RS323_PCON_ADR  = (void *)(Base_RS232 +
    RS323_PCON);
uint32_t * const RS323_SCON_ADR  = (void *)(Base_RS232 +
    RS323_SCON);
uint32_t * const RS323_SBUF_ADR  = (void *)(Base_RS232 +
    RS323_SBUF);
uint32_t * const RS323_SRELL_ADR = (void *)(Base_RS232 +
    RS323_SRELL);

uint32_t * const RS323_SRELH_ADR = (void *)(Base_RS232 +
    RS323_SRELH);
uint32_t * const RS323_TCON_ADR  = (void *)(Base_RS232 +
    RS323_TCON);
uint32_t * const RS323_TL_ADR    = (void *)(Base_RS232 +
    RS323_TL);
uint32_t * const RS323_TH_ADR    = (void *)(Base_RS232 +
    RS323_TH);

uint32_t * const RS323_ADCON_ADR = (void *)(Base_RS232 +
    RS323_ADCON);


void sendMeasureReply(void)
{
    uint8_t* TauPointer = (uint8_t*)&(TAU);
    switch(ByteCounter)
    {
        case 0 :
          *RS323_SBUF_ADR = *(TauPointer+7);
          break;
        case 1 :
          *RS323_SBUF_ADR = *(TauPointer+6);
          break;
        case 2 :
          *RS323_SBUF_ADR = *(TauPointer+5);
          break;
        case 3 :
```

```
    *RS323_SBUF_ADR = *(TauPointer+4);
    break;
  case 4:
    *RS323_SBUF_ADR = *(TauPointer+3);
    break;
  case 5 :
    *RS323_SBUF_ADR = *(TauPointer+2);
    break;
  case 6 :
    *RS323_SBUF_ADR = *(TauPointer+1);
    break;
  case 7 :
    *RS323_SBUF_ADR = *TauPointer;
    break;
  case 8 :
    *RS323_SBUF_ADR = getOneByte((uint32_t)DataHighMin,1);
    break;
  case 9:
    *RS323_SBUF_ADR = getOneByte((uint32_t)DataHighMin,0);
    break;
  case 10:
    *RS323_SBUF_ADR = getOneByte((uint32_t)DataHighMax,1);
    break;
  case 11:
    *RS323_SBUF_ADR = getOneByte((uint32_t)DataHighMax,0);
    break;
  case 12:
    *RS323_SBUF_ADR = getOneByte((uint32_t)NoiseMin,1);
    break;
  case 13:
    *RS323_SBUF_ADR = getOneByte((uint32_t)NoiseMin,0);
    break;
  case 14:
    *RS323_SBUF_ADR = getOneByte((uint32_t)NoiseMax,1);
    break;
  case 15:
    *RS323_SBUF_ADR = getOneByte((uint32_t)NoiseMax,0);
    break;
  case 16:
    *RS323_SBUF_ADR =
        getOneByte(getFrequencyDivider(FreqencyDividerDataIndex),3);
    break;
  case 17:
    *RS323_SBUF_ADR =
        getOneByte(getFrequencyDivider(FreqencyDividerDataIndex),2);
```

```
  break;
case 18:
  *RS323_SBUF_ADR =
      getOneByte(getFrequencyDivider(FreqencyDividerDataIndex),1);
  break;
case 19:
  *RS323_SBUF_ADR =
      getOneByte(getFrequencyDivider(FreqencyDividerDataIndex),0);
  break;
case 20:
  *RS323_SBUF_ADR = getOneByte(DecayStartPosition,1);
  break;
case 21:
  *RS323_SBUF_ADR = getOneByte(DecayStartPosition,0);
  break;
case 22:
  *RS323_SBUF_ADR = (storedAverages ==
      0)?(uint8_t)(ScratchBlockCount) :
      (uint8_t)(storedAverages*ScratchBlockCount);
  break;
case 23:
  *RS323_SBUF_ADR = getOneByte(BaseSampleFrequencyKHz,3);
  break;
case 24:
  *RS323_SBUF_ADR = getOneByte(BaseSampleFrequencyKHz,2);
  break;
case 25:
  *RS323_SBUF_ADR = getOneByte(BaseSampleFrequencyKHz,1);
  break;
case 26:
  *RS323_SBUF_ADR = getOneByte(BaseSampleFrequencyKHz,0);
  break;
case 27:
  *RS323_SBUF_ADR = currentScratchBlock;
  break;
case 28:
  *RS323_SBUF_ADR = Status_ActiveAquisitionMode;
  break;
case 29:
  *RS323_SBUF_ADR = storedAverages;
  break;
case 30:
  *RS323_SBUF_ADR = (currentAverageBlock && 0xFF);
  break;
case 31:
```

```
        *RS323_SBUF_ADR = ProtocolTransmitter_nextOpCodeToReply;
        break;
      default:
        *RS323_SBUF_ADR = 0;
        break;
   }



   ByteCounter++;
}


void sendByteFromRamAuto()
{
  sendByteFromRam(ByteCounter);
  ByteCounter++;
}
void sendByteFromConfigRamAuto()
{
  sendByteFromConfigRam(ByteCounter);
}

void sendByteFromRam(unsigned long long byteNumber)
{
    uint32_t RamAdress =( byteNumber >> 2) & 0xFFFFffff;
    volatile uint8_t data = getOneByte( *(RAM_Base+RamAdress),
       (byteNumber%4)   );
    *RS323_SBUF_ADR = data;
}
void sendByteFromConfigRam(unsigned long long byteNumber)
{
    uint32_t RamAdress =( byteNumber >> 2) & 0xFFFFffff;
    volatile uint8_t data = getOneByte(
       *(Config_Base+RamAdress), (byteNumber%4)   );
    *RS323_SBUF_ADR = data;
    ByteCounter++;
}

void sendByteFromRamAuto16bitMode()
{
  sendByteFromRam16bitMode(ByteCounter);
  ByteCounter++;
}
```

```
void sendByteFromRam16bitMode(unsigned long long byteNumber)
{
    uint32_t RamAdress =( byteNumber >> 2) & 0xFFFFffff;
    uint8_t byteNum =(byteNumber%4);

    // send high byte first
    if(byteNum == 1 | byteNum == 3)
       byteNum--;
    else
       byteNum++;
    volatile uint8_t data = getOneByte( *(RAM_Base+RamAdress),
       byteNum );
    *RS323_SBUF_ADR = data;
}

void sendMagicByteAndSetSendOpCodeState()
{
    State_ProtocolTransmitter = State_SendOpCode;
    *RS323_SBUF_ADR = MagicByte;
}



void initCommunication()
{
  __mtc0(__mfc0(1) |(0x0001 << Intr_RS232_A), 1); // Enable xxx
     interrupt
  __mtc0(__mfc0(9) |(0x0001 << Intr_RS232_A), 9); // set xxx
     interrupt to edge


  // ********** Serial Port INIT  **************
  // Baudrate      19200          (=?? us/bit)
  // baseFrequUC   12.500.000 Hz = 12,5 MHz
  // Baudrate Calculated = [2^SMOD  *  baseFreq]/[64 * (2^10 -
     SREL)]
  // Baudrate Calculated = [2^1     * 12500000 ]/[64 * (1024 -
     1004)]
  // Baudrate Calculated = [  2     * 12500000 ]/[64 * (  20
            )]   = 25000000 / ´1280  = 19531.25 // within 3%
     Tolerance


  // PCON(7)     : SMOD = 0b1
  // PCON(6:0)   : x
```

```
// PCON          : 0b1000.0000 = 0x80
*RS323_PCON_ADR = 0x80;

// 0d1004    = 0x3EC
// SREL9..8  = SRELH1..0
// SREL7..0  = SRELL7..0
// SRELH1   : 0x03
// SRELL7   : 0xEC
*RS323_SRELH_ADR = 0x03;
*RS323_SRELL_ADR = 0xEC;


// Mode b01 : 8-bit UART mode with variable Baud rate
// SCON(7) :        SM(0)  : 0
// SCON(6) :        SM(1)  : 1
// SCON(5) :               : 0   When set, enables
    multiprocessor communication feature for operational modes
    2 and 3
// SCON(4) :               : 1   Controlled by software to
    enable/disable reception: / 0 = disable reception of
    serial data / 1 = enable reception of serial data
// SCON(3) :               : x   The 9th transmitted bit in
    Modes 2 and 3.
// SCON(2) :               : x   The 9th bit received in Modes 2
    and 3.
// SCON(1) :               : x   Transmit interrupt flag, set by
    hardware after completion of a serial transfer. Must be
    cleared by software
// SCON(0) :               : x   Receive interrupt flag, set by
    hardware after completion of a serial reception. Must be
    cleared by software
// 0b0101_0000 = 0x50
*RS323_SCON_ADR = RS323_SCON_DEFAULT;


// ADCON.7   :1  Used to determine the source for baud rate
    generation when the serial interface is operating in Modes
    1 and 3.
                // 0 = Timer Unit
                // 1 = Internal Baud rate Generator (using
                    SREL register)
// ADCON.6   :0  Not Used
// ADCON.5   :0  Not Used
// ADCON.4   :0  Not Used
// ADCON.3   :0  Not Used
```

```
    // ADCON.2    :0    Not Used
    // ADCON.1    :0    Not Used
    // ADCON.0    :0    Not Used
    *RS323_ADCON_ADR = 0x80;


    // TCON.7 .. 0 .. General purpose Flag 11 available for user
    // TCON.6 .. 0 .. Timer Run control bit. If cleared, Timer
        Unit is stopped.
    // TCON.5 .. 0 .. General purpose Flag 10 available for user
    // TCON.4 .. 0 .. General purpose Flag 9 available for user
    // TCON.3 .. 0 .. General purpose Flag 8 available for user
    // TCON.2 .. 0 .. General purpose Flag 7 available for user
    // TCON.1 .. 0 .. General purpose Flag 6 available for user
    // TCON.0 .. 0 .. General purpose Flag 5 available for user
    // 0b0000_0000 = 0x00
    *RS323_TCON_ADR = 0x00;
}
```

**MemoryOperations.c**

Listing 5: Mikrocontroller Code

```
#include "MemoryOperations.h"



// Ram
uint32_t * const RAM_Base = (void *)Base_BRAM;
// Config Ram
uint32_t * const Config_Base = (void *)Base_config;



uint16_t get16BitWordFromRam(uint32_t WordCounter)
{
    uint32_t RamAdress =( WordCounter >> 1) & 0xFFFFffff;
    if(WordCounter % 2 == 0)
    {
        return *(RAM_Base+RamAdress) & 0x0000ffff ;
    }
    else
    {
        return (*(RAM_Base+RamAdress)>> 16) & 0x0000ffff ;
    }
}

void write16BitWordToRam(uint32_t WordCounter,uint16_t data )
{
     writeByteToRam(WordCounter*2,(data & 0x00FF));
     writeByteToRam((WordCounter*2)+1,(data & 0xFF00)>> 8);
}

void writeByteToRamAuto(uint8_t data)
{
  writeByteToRam(ByteCounter, data);
  ByteCounter++;
}

void writeByteToRam(unsigned long long byteNumber, uint8_t data)
{
    uint32_t RamAdress =( byteNumber >> 2) & 0xFFFFffff;
    uint32_t OrigValue = *(RAM_Base+RamAdress);
```

```
    uint8_t bytePos = byteNumber%4;
    uint32_t NewValue = (data << (bytePos*8) );
    uint32_t mask = 0x00000000;
    switch (bytePos)
    {
      case 0:
        mask = 0xFFFFFF00;
        break;
      case 1:
        mask = 0xFFFF00FF;
        break;
      case 2:
        mask = 0xFF00FFFF;
        break;
      case 3:
        mask = 0x00FFFFFF;
        break;
      default:
          ErrorHandler(Error_SwitchState_writeByteToRam);
          break;
    }
    OrigValue = OrigValue & mask;                // clear old val
    OrigValue = OrigValue | NewValue ;           // insert new byte
    *(RAM_Base+RamAdress) = OrigValue;
}

/** *****************************************************
 * Fills Ram
 */
void FillRamWithPredefinedPattern()
{


  uint32_t ramSize = 0;
  uint32_t dataPoints = (uint32_t)
      (DataPointsPerRamBlock*WholeRamBlockCount );
  ramSize = dataPoints/2;

  for(int i = 0 ; i < ramSize;i++)
  {
      *(RAM_Base+i) = (uint32_t)(((i+1)*3+1) << 16) | ((i+1)*3) ;
  }
```

```
}
/** ****************************************************
 * Fills Ram
 */
void FillRamWithPredefinedNoisePattern()
{


  uint32_t ramSize = 0;
  uint32_t dataPoints = (uint32_t)
      (DataPointsPerRamBlock*WholeRamBlockCount );
  ramSize = dataPoints/2;

  for(int i = 0 ; i < ramSize;i++)
  {
      *(RAM_Base+i) = (uint32_t)(((i+1)*3+1) << 16) | ((i+1)*3) ;
  }

}



void loadFrequencyDividerCounter(uint8_t index)
{
    if(index > 10 )
        ErrorHandler(Error_FreqencyDividerDataIndexTooHigh);

    FreqencyDividerDataIndex = index;
    *FreqencyDividerDataAddr  = 15; // Zero Output
    *FreqencyDividerDataAddr  = FreqencyDividerDataIndex;
    int WaitFor = getFrequencyDivider(FreqencyDividerDataIndex);
    for(; WaitFor > 23;WaitFor--)
        __nop();
}




void setOpcodeForCustomCode(uint8_t data)
{
 ConfigBlockForCustomCode_OpCode = data;
 writeConfigOut();
}
```

```
// Config 0 ... not used
// Config 1 ... Operation Mode
// Config 2 ... Start Address
// Config 3 ... End Address
void writeConfigOut(void)
{
   volatile uint32_t tmpValue = 0;
   uint32_t thirtyTwoBitWideRamBlockSize = 0;

   tmpValue =    (0   );
   tmpValue = tmpValue | (0                              <<
       (1*8) );
   tmpValue = tmpValue | (0                              <<
       (2*8) );
   tmpValue = tmpValue | (0                              <<
       (3*8) );
   *(Config_Base) = tmpValue;

   tmpValue =            (ConfigBlockForCustomCode_OpCode );
   tmpValue = tmpValue | (0                          << (1*8)
       );
   tmpValue = tmpValue | (0                          << (2*8)
       );
   tmpValue = tmpValue | (0                          << (3*8)
       );
   *(Config_Base+1) = tmpValue;


   thirtyTwoBitWideRamBlockSize = (DataPointsPerRamBlock/2);
   tmpValue =  currentScratchBlock
       *thirtyTwoBitWideRamBlockSize ;
   *(Config_Base+2) = tmpValue;

   tmpValue = (tmpValue) + thirtyTwoBitWideRamBlockSize -1;
   *(Config_Base+3) = tmpValue;
}

/**
 * Flips first scratchBolck (inverts)
 */
void flipMemory(void)
{
 volatile  uint32_t j;
 volatile uint16_t dataTmp;
 for(int j = 0; j < DataPointsPerRamBlock;j++)
```

```
 {
     dataTmp =
         get16BitWordFromRam((currentScratchBlock*DataPointsPerRamBlock)+j);
     dataTmp = 65535 - dataTmp ;
     // 0...32767        32768...65535
     // 65535 - 32768 = 32767
     // 65535 - 0 = 65535
     write16BitWordToRam(
         (currentScratchBlock*DataPointsPerRamBlock)+j
         ,dataTmp );
 }

}
```

**settings.h**

Listing 6: Mikrocontroller Code

```
//.........................................

#ifndef __SETTINGS_H__
#define __SETTINGS_H__

//----------------------------------------------------------------------------
//   DEFAULTS:
//-----------


// If Decay looks like Rise, use FLIP_MEMORY
#define  FLIP_MEMORY

//***** Define Filter, findDecayStart etc...
//***** enable only 1 but enable 1!
//#define PT_CDIV32
//#define PT_CDIV64
//--------
//#define PtFPP_CDIV32
//#define PtFPP_CDIV64
//--------
//#define PdFPP_CDIV512
//#define PdFPP_CDIV1024
//--------
//#define RuDPP_CDIV4
//#define RuDPP_CDIV8
//--------
#define Test_CDIV0
// INSERT NEW ALGORITHM INFORMATION HERE (new define)


// if defined, stage0 preProcessing will be perfomed
#define enableStage0


// For the case that customCode does not return ...
// enableCustomCodeWatchdog 0x01   ... watchdog enabled
// enableCustomCodeWatchdog 0x00   ... watchdog disabled
#define enableCustomCodeWatchdog 0x01

// normal operation Mode                              :
    simulationMode      0x00
```

```
// Simulation Mode, fills Ram with predefinded Patterns:
    simulationMode      0x01
#define simulationMode                                0x00


//------------------------------------------------------------------------
//Index  Divider
// 0      1
// 1      2
// 2      4
// 3      8
// 4     16
// 5     32
// 6     64
// 7    128
// 8    256
// 9    512
//10  1024
#ifdef   PT_CDIV32
  #define FreqencyDividerDataInitialValue            5          //
      Default: 0
  #define FreqencyDividerDataMaxValue                5         //
      Default: 10
  #define TriggerOutMultiplicator                    1
#endif
#ifdef   PT_CDIV64
  #define FreqencyDividerDataInitialValue            6          //
      Default: 0
  #define FreqencyDividerDataMaxValue                6         //
      Default: 10
  #define TriggerOutMultiplicator                    1
#endif
#ifdef   PtFPP_CDIV32
  #define FreqencyDividerDataInitialValue            5          //
      Default: 0
  #define FreqencyDividerDataMaxValue                5         //
      Default: 10
  #define TriggerOutMultiplicator                    1
#endif
#ifdef   PtFPP_CDIV64
  #define FreqencyDividerDataInitialValue            6          //
      Default: 0
  #define FreqencyDividerDataMaxValue                6         //
      Default: 10
  #define TriggerOutMultiplicator                    1
```

```
#endif
#ifdef   PdFPP_CDIV512
  #define FreqencyDividerDataInitialValue          9          //
     Default: 0
  #define FreqencyDividerDataMaxValue              9         //
     Default: 10
  #define TriggerOutMultiplicator                  1
#endif
#ifdef   PdFPP_CDIV1024
  #define FreqencyDividerDataInitialValue          10
     // Default: 0
  #define FreqencyDividerDataMaxValue              10         //
     Default: 10
  #define TriggerOutMultiplicator                  1
#endif
#ifdef   RuDPP_CDIV4
  #define FreqencyDividerDataInitialValue          2          //
     Default: 0
  #define FreqencyDividerDataMaxValue              2         //
     Default: 10
  #define TriggerOutMultiplicator                  1
#endif
#ifdef   RuDPP_CDIV8
  #define FreqencyDividerDataInitialValue          3          //
     Default: 0
  #define FreqencyDividerDataMaxValue              3         //
     Default: 10
  #define TriggerOutMultiplicator                  1
#endif
#ifdef   Test_CDIV0
  #define FreqencyDividerDataInitialValue          0          //
     Default: 0
  #define FreqencyDividerDataMaxValue              0         //
     Default: 10
  #define TriggerOutMultiplicator                  1         //
     always one
#endif
// INSERT NEW ALGORITHM INFORMATION HERE (new defines with
   preferred settings)



//** *****************************************************
// Maximum Sampling Frequency
#define BaseSampleFrequencyKHz        125000
```

```
//** ******************************************************
// for use in Algorithm.c->sampledDownEnough()
#define howMuchDataPointsInPercentShouldBeAbove              20
    //%1-100
#define howMuchDataPointsInPercentShouldBeWithinNoiseRange   10
    //%1-100
#define howMuchErrorPercentAreAllowed                        20
    //%1-100;


//** ******************************************************

#define Intr_fromCustomCode Intr_gpio_A
#define Intr_InternalTimer   0
//** ******************************************************
#define RS323_PCON        0x00
#define RS323_SCON        0x01
#define RS323_SBUF        0x02        // send and receive Buffer
#define RS323_SRELL       0x03
#define RS323_SRELH       0x04
#define RS323_TCON        0x05
#define RS323_TL          0x06
#define RS323_TH          0x07
#define RS323_ADCON       0x08
#define RS323_SCON_DEFAULT    0x50 // default value




//** ******************************************************
// Receiving States
#define State_WaitForMagicByte                    0x01
#define State_WaitForOpCode                       0x02
#define State_WaitForDataBytesInMessageIndex      0x04
#define State_WaitForDataConfigBlock              0x08

// Sending States
#define State_SendNothing                         0x00
#define State_SendRamData                         0x01
#define State_SendOpCode                          0x02
#define State_SendDataBytesInMessageIndex         0x04
#define State_SendConfigBlock                     0x08
#define State_SendMeasureReply                    0x10
```

```
//** ****************************************************
// Aquiring Mode
#define Mode_AquireNoise                          1
#define Mode_AquireNoiseOnly                      2
#define Mode_AquireDataFindDecay                  3
#define Mode_AquireDataHigh                       4
#define Mode_AquireOnlyADCData                    5
#define Mode_AquireADCDataWithTrigger             6
#define Mode_AquireADCDataWithTriggerAndCalcMeanIfNecessary     7
#define Mode_AquireDataHighOnly                                  8

//** ****************************************************
// Operating Modes for CustomCode
#define CustomCode_FillRamWithAdcData             0x02

//** ****************************************************
#define Error_NoError
    0x00  //0000_0000
#define Error_SwitchState_ProtocolReceiverDefault
    0x01  //0000_0001
#define Error_NotReceiveAndNotTransmitInterrupt
    0x02  //0000_0010
#define Error_SwitchState_ProtocolOpcodeDefault
    0x03  //0000_0011
#define Error_SwitchState_ProtocolDataLenIndexDefault
    0x04  //0000_0100
#define Error_SwitchState_ProtocolSenderDefault
    0x05  //0000_0101
#define Error_SwitchState_getOneByte
    0x06  //0000_0110
#define Error_WrongActiveAquisitionMode
    0x07  //0000_0111
#define Error_SwitchState_State_SendDataBytesInMessageIndex
    0x08  //0000_1000
#define Error_FreqencyDividerDataIndexTooHigh
    0x09  //0000_1001
#define Error_SwitchState_getDataBytesInMessage
    0x0A  //0000_1010
#define Error_ConfigErrorMemorySettings
    0x0B  //0000_1011
#define Error_SwitchState_writeByteToRam
    0x0C  //0000_1100
#define Error_SwitchState_writeByteToConfig
```

```
    0x0D   //0000_1101
#define Error_WrongStorageMode
    0x0E   //0000_1110
#define Error_WrongDataTypeSize
    0x0F   //0000_1111
#define Error_getScratchBlockCount
    0x10   //0001_0000
#define Error_getVarStorageIndex
    0x11   //0001_0001
#define Error_WrongStatusMode
    0x12   //0001_0010
#define Error_NotEnoughScratchBlocksForStage0
    0x13   //0001_0011
#define Error_uint16overflowStage0
    0x14   //0001_0100

#define Warn_CustomCodeTimerOverflow
    0x7F   //0111_1111

//** ******************************************************

//** ******************************************************
#define uint16MaxValue 65535
//** ******************************************************
#define TriggerEnableTimeMicroSecondsForSquareOutput 60
//RuDPP -> 100
//PtFPP -> 600
//........................................
 #define NOISE_STATISTICS        0
 #define DATA_HIGH_STATISTICS    1
 #define DATA_TEMP_STATISTICS    2

#endif // __SETTINGS_H__
```

**SerialProtocol.h**

Listing 7: Mikrocontroller Code

```
#ifndef __SERIALPROTOCOL_H__
#define __SERIALPROTOCOL_H__

#include "Globals.h"


/** Serial Settings
 * <ul>
 * <li> 57600 Baud </li>
 * <li> 8 Databits </li>
 * <li> 1 Startbit </li>
 * <li> 1 Stopbit  </li>
 * </ul>
 */

/** Protocol
 * Message Length : n (min: 2, max: 65541)
 * <table border="1">
 * <tr>
 * <th><code> 0                                    </code></th>
 * <th><code> 1                                    </code></th>
 * <th><code> 2                                    </code></th>
 * <th><code> (3 ... n-1)                          </code></th>
 * </tr>
 * <tr>
 * <th><code> MagicByte (0xBE)                     </code></th>
 * <th><code> OpCode                               </code></th>
 * <th><code> DataBytesInMessageIndex              </code></th>
 * <th><code> Data                                 </code></th>
 * </tr> <tr>
 * <th><code> 1 Byte                               </code></th>
 * <th><code> 1 Byte                               </code></th>
 * <th><code> 1 Byte (optional)                    </code></th>
 * <th><code> 0 to 65535 Bytes (optional)          </code></th>
 * </tr> </table>

 * <br><br>
 * valid Message Examples
 * <ul>
 * <li>0xBEEF</li>
 * <li>0xBEA0</li>
 * <li>0xBEA1</li>
```

```
 * <li>0xBEBE01AABBCCDD</li>
 * </ul>
 * <br>
 * <ul>
 * <li><code> MagicByte                              </code></li>
 *     <ul><li> Signals Transmission Start </li></ul>
 * <li><code> OpCode                                 </code></li>
 *     <ul><li> Which Action should be performed, see Section
     OpCodes for Details </li></ul>
 * <li><code> DataBytesInMessageIndex                </code></li>
 *     <ul><li> Index, which is bound to a specific Byte Count.
     Determines how much Data Bytes are in this specific Message.
 *              Defined in SerialProtocol.h </li></ul>
 * <li><code> Data                                   </code></li>
 *     <ul><li> Data itself</li></ul>
 * </ul>
 */

#define DataBytesInMessageIndexEmpty                  0x00 //
    Index = 0 -> DataLen =     0 Bytes
#define DataBytesInMessageIndex4Bytes                 0x01 //
    Index = 1 -> DataLen =     4 Bytes
#define DataBytesInMessageIndex32768Bytes             0x02 //
    Index = 2 -> DataLen = 32768 Bytes
#define DataBytesInMessageIndex2048Bytes              0x03 //
    Index = 3 -> DataLen =  2048 Bytes
#define DataBytesInMessageIndex8Bytes                 0x04 //
    Index = 4 -> DataLen =     8 Bytes
#define DataBytesInMessageIndex256Bytes               0x05 //
    Index = 5 -> DataLen =   256 Bytes


#define MagicByte                      0xBE


/*------------------------------------------------------------------*/
/** OpCodes */

/** \def OpCodeSetConfigBlock
 * \brief Contains 4 DataBytes, which are stored as ConfigBlock
     on the Device
 * Example: <code>0xBEBE01AABBCCDD</code>
 * Data Bytes Order : Byte3 (AA), Byte2 (BB), Byte1 (CC), Byte0
     (DD)
```

```
 * Byte3: future use
 * Byte2: future use
 * Byte1: future use
 * Byte0: ClockDividerIndex, The low 4 bits are used to set a
     new ClockDividerIndex
 */
#define OpCodeSetConfigBlock          0xBE

/** \def OpCodeGetConfigBlock
 * \brief Returns 4 DataBytes (ConfigBlock)
 * ExampleRequestToHardware    : <code>0xBEBF</code>
 * ExampleResponseFromHardware: <code>0xBEBF01AABBCCDD</code>
 * Data Bytes Order : Byte3 (AA), Byte2 (BB), Byte1 (CC), Byte0
     (DD) */
#define OpCodeGetConfigBlock          0xBF

/** \def OpCodeInitRamWithData
 * \brief Fills Ram with predefined Pattern */
#define OpCodeInitRamWithData         0xA0

/** \def OpCodeGetRamData
 * Returns Ram Data as 16 bit Values
 * ExampleRequestToHardware    : <code>0xBEA1</code>
 * ExampleResponseFromHardware: <code>0xBEA101AABBCCDD</code>
 * Value 0 would be 0xAABB, Value 1 would be 0xCCDD
 * ExampleResponseFromHardware:
     <code>0xBEA104AABBCCDD00112233</code>
 * Value 0 would be 0xAABB, Value 1 would be 0xCCDD, Value 3
     would be 0x0011, Value 4 would be 0x2233  */
#define OpCodeGetRamData              0xA1

/** \def OpCodeGetAdcData
 * Fills Ram with ADC Data according to RamConfiguration
 * Corresponding to RamConfiguration its also possible that mean
     value calculation is performed.
 * Returns Ram Data as 16 bit Values
 * ExampleRequestToHardware    : <code>0xBEEF</code>
 * ExampleResponseFromHardware: <code>0xBEEF01AABBCCDD</code>
 * Value 0 would be 0xAABB, Value 1 would be 0xCCDD */
#define OpCodeGetAdcData              0xEF

/** \def OpCodeMeasure
 * Decay Measurement is performed as a 2 phase Operation
 * First, Noise is aquired to Ram and Statistics are calculated.
 * Second, Data is aquired to Ram and Decay Time are calculated.
```

```
 * DecayTime and other Statistics are responded
 * Returns Data as 16 bit Values
 * ExampleRequestToHardware    : <code>0xBEE0</code>
 * ExampleResponseFromHardware: <code>0xBEE0 01 AABB CCDD</code>
 * Value 0 would be 0xAABB, Value 1 would be 0xCCDD */
#define OpCodeMeasure               0xE0

#define OpCodeOnlyNoise             0xE1
#define OpCodeOnlyDataHigh          0xE2
#define OpCodeOnlyData              0xE6


/** \def OpCodeGetAdcDataWithTrigger
 * Fills Ram with ADC Data according to RamConfiguration
 * No mean value calculation is performed.
 * TriggerOut is performed to enable Light Source
 * Returns Ram Data as 16 bit Values
 * ExampleRequestToHardware    : <code>0xBEE3</code>
 * ExampleResponseFromHardware: <code>0xBEEF01AABBCCDD</code>
 * Value 0 would be 0xAABB, Value 1 would be 0xCCDD */
#define OpCodeGetAdcDataWithTrigger   0xE3


/** \def OpCodeGetAdcDataWithTriggerAndCalcMeanIfNecessary
 * Fills Ram with ADC Data according to RamConfiguration
 * Mean value calculation is performed if enough space in Ram.
 * TriggerOut is performed to enable Light Source
 * Returns Ram Data as 16 bit Values
 * ExampleRequestToHardware    : <code>0xBEE4</code>
 * ExampleResponseFromHardware: <code>0xBEEF01AABBCCDD</code>
 * Value 0 would be 0xAABB, Value 1 would be 0xCCDD */
#define OpCodeGetAdcDataWithTriggerAndCalcMeanIfNecessary   0xE4



/** \def OpCodeEnableTriggerWithSquareFunction
 * Drives the Trigger with a square Fuction
 * Can be used to set up LED Driving Strength
 * ExampleRequestToHardware    : <code>0xBEE5</code>  */
#define OpCodeEnableTriggerWithSquareFunction   0xE5

/** \def OpCodeMeasureForMatlab
 * Will not perform stage0preProcessing, findDecayStart, and
    calcTau
 * Decay Measurement is performed as a 2 phase Operation
```

```
 * First, Noise is aquired to Ram and Statistics are calculated.
 * Second, Data is aquired to Ram and Decay Time are calculated.
 * DecayTime and other Statistics are responded
 * Returns Data as 16 bit Values
 * ExampleRequestToHardware    : <code>0xBEE7</code>
 * ExampleResponseFromHardware: <code>0xBEE0 01 AABB CCDD</code>
 * Value 0 would be 0xAABB, Value 1 would be 0xCCDD */
#define OpCodeMeasureForMatlab              0xE7


/** \def OpCodeMeasureForMatlab
 * Will not perform any Measurement or a calculation
 * Just returns MeasurementReply data */
#define OpCodeGetMeasureReplyOnly           0xE8




 /** Measure Request Reply - Data Package Format
  * <table>
  * <tr>
  * <td> Byte 0  </td><td>Byte 1</td><td>     Byte 2
</td><<td>Byte 3-10</td><td>Byte11-12</td><td>Byte13-14</td>
<td>Byte 15-16</td><td>byte 17-18</td><td>Byte 19-22</td><td>
     Byte23-24     </td><td> Byte 25</td><td>  Byte 25-28
</td><td>Byte29-255</td></tr><tr>
  * <td>MagicByte</td><td>OpCode</td><td>DataBytesInMessageIndex
</td><td>Tau[sec]  </td><td> DataMin </td><td> DataMax </td><td>
NoiseMin </td><td> NoiseMax
    </td><td>ClkDivider</td><td>decayStartPosition
</td><td>Averages</td><td>baseFequencyKHZ</td><td>future
    use</td></tr><tr>
  * <td>  0xBE   </td><td> 0xE0 </td><td>     uint8_t
</td><td>Double     </td><td>   uint16 </td><td>   uint16 </td><td>
 uint16    </td><td>    uint16 </td><td> uint32    </td><td>
uint16          </td><td>   uint8 </td><td>     uint32
</td><td>future use</td></tr>
  * </table>
  */



#endif // __SERIALPROTOCOL_H__
```

**Protocol.h**

Listing 8: Mikrocontroller Code

```
//........................................

#ifndef __SETTINGS_H__
#define __SETTINGS_H__

//....................................

#define MagicByte                   0xBE
#define OpCodeSetConfigBlock        0xBE
#define OpCodeGetConfigBlock        0xBF
#define OpCodeGetConfigBlock        0xBF
#define OpCodeInitRamWithData       0xA0
#define OpCodeGetRamData            0xA1



#define State_WaitForMagicByte      0x01
#define State_WaitForOpCode         0x02
#define State_WaitForDataLenIndex   0x04
#define State_WaitForData           0x08



#define DataLenIndex4Byte           0x01 // Index = 1 -> DataLen
    = 4 Byte


//.......................................

#endif // __SETTINGS_H__
```

**MemorySettings.h**

Listing 9: Mikrocontroller Code

```
#ifndef __MEMORYSETTINGS_H__
#define __MEMORYSETTINGS_H__

#include "settings.h"

/** RamConfigIndex
 */

//#define __C1__ 1  // NOT WORKING PREPROCESSOR MACROS IF ELIF
    ETC..., by SUSH  #IFDEF


        //#ifdef __C0__ 1
         /* #define DataBytesInMessageIndex    3
          #define DataPointsPerRamBlock      1024
          #define WholeRamBlockCount         16
          #define ScratchBlockCount          16*/
        //#endif
        //#ifdef __C1__
          /*#define DataBytesInMessageIndex    2
          #define DataPointsPerRamBlock      16384
          #define WholeRamBlockCount         1
          #define ScratchBlockCount          1 */
        //#endif
        //#ifdef Config_RamConfigIndex2
          //#define DataBytesInMessageIndex    4
          //#define DataPointsPerRamBlock      4
          //#define WholeRamBlockCount         3
          //#define ScratchBlockCount          3
        //#endif
        //#ifdef Config_RamConfigIndex3
         /* #define DataBytesInMessageIndex    3
          #define DataPointsPerRamBlock      1024
          #define WholeRamBlockCount         1
          #define ScratchBlockCount          1*/
        //#endif
        //#ifdef Config_RamConfigIndex4
          /*#define DataBytesInMessageIndex    4
          #define DataPointsPerRamBlock      4
          #define WholeRamBlockCount         1
          #define ScratchBlockCount          1*/
```

```
//#endif
//#ifdef Config_RamConfigIndex5
  /*#define DataBytesInMessageIndex    3
  #define DataPointsPerRamBlock       1024
  #define WholeRamBlockCount          4
  #define ScratchBlockCount           4*/
//#endif
//#ifdef Config_RamConfigIndex6
/* #define DataBytesInMessageIndex     4
  #define DataPointsPerRamBlock       4
  #define WholeRamBlockCount          4
  #define ScratchBlockCount           1*/
//#endif
//#ifdef Config_RamConfigIndex7
  /*#define DataBytesInMessageIndex    4
  #define DataPointsPerRamBlock       4
  #define WholeRamBlockCount          4
  #define ScratchBlockCount           2*/
//#endif
//#ifdef Config_RamConfigIndex8
  /*#define DataBytesInMessageIndex    4
  #define DataPointsPerRamBlock       4
  #define WholeRamBlockCount          4
  #define ScratchBlockCount           3*/
//#endif
//#ifdef Config_RamConfigIndex9
 /* #define DataBytesInMessageIndex    3
  #define DataPointsPerRamBlock       1024
  #define WholeRamBlockCount          16
  #define ScratchBlockCount           4*/
//#endif
//#ifdef Config_RamConfigIndex10
 /* #define DataBytesInMessageIndex    3
  #define DataPointsPerRamBlock       1024
  #define WholeRamBlockCount          8
  #define ScratchBlockCount           8 */
//#endif
//#ifdef Config_RamConfigIndex11
 /*#define DataBytesInMessageIndex     3
  #define DataPointsPerRamBlock       1024
  #define WholeRamBlockCount          12
  #define ScratchBlockCount           12 */
//#endif
//#ifdef Config_RamConfigIndex12      // AVG 64
  #define DataBytesInMessageIndex     3
```

```
  #define  DataPointsPerRamBlock      1024
  #define  WholeRamBlockCount         16
  #define  ScratchBlockCount          8
//#endif
//#ifdef Config_RamConfigIndex13     // AVG 32
 /* #define DataBytesInMessageIndex     3
  #define  DataPointsPerRamBlock      1024
  #define  WholeRamBlockCount         12
  #define  ScratchBlockCount          8 */
//#endif
// insert new Memory Configuration here
//#endif


//#define averageStorageBlockCount (WholeRamBlockCount -
    ScratchBlockCount)
//#if averageStorageBlockCount == 0
//     #define valuesToStore 1
//#else
//     #define valuesToStore averageStorageBlockCount
//#endif

#endif // __MEMORYSETTINGS_H__
```

**MemoryOperations.h**

Listing 10: Mikrocontroller Code

```
#ifndef __MEMORYOPERATIONS_H__
#define __MEMORYOPERATIONS_H__

#include "Globals.h"
#include "hardware.h"

uint16_t get16BitWordFromRam(uint32_t WordCounter);
void write16BitWordToRam(uint32_t WordCounter,uint16_t data );
void writeByteToRamAuto(uint8_t data);
void writeByteToRam(unsigned long long byteNumber, uint8_t data);
void FillRamWithPredefinedPattern(void);
void FillRamWithPredefinedNoisePattern(void);
void loadFrequencyDividerCounter(uint8_t index);
void stopADCClocking(void)      ;
void setOpcodeForCustomCode(uint8_t data);
void writeConfigOut(void);
void flipMemory(void);


#endif // __MEMORYOPERATIONS_H__
```

**hardware.h**

Listing 11: Mikrocontroller Code

```
//..............................
// Automatically generated header file.
// This file should not be edited.
//..............................

#ifndef __HARDWARE_H__
#define __HARDWARE_H__

//..............................
#define Base_gpio                  0xFF000000
#define Size_gpio                  0x00000002
#define Intr_gpio_A                1
#define Intr_gpio_B                2
#define Intr_gpio_C                3
#define Intr_gpio_D                4
//..............................

//..............................
#define Base_RS232                 0xFF400000
#define Size_RS232                 0x00000010
#define Intr_RS232_A               5
//..............................

//..............................
#define Base_config                0xFF600000
#define Size_config                0x00000004
//..............................

//..............................
#define Base_FrequencyDivider      0xFF800000
#define Size_FrequencyDivider      0x00000004
//..............................

//..............................
#define INTERRUPT_CONTROL_CFG      0x0000003E
#define INTERRUPT_KINDS_CFG        0x00000000
#define INTERRUPT_EDGE_KIND_CFG    0x00000000
#define INTERRUPT_LVL_KIND_CFG     0x0000003E
//..............................

//..............................
#define Base_uc32                  0x00000000
```

```
#define Size_uc32                    0x00008000
//..............................

//..............................
#define Base_BRAM                    0x01000000
#define Size_BRAM                    0x00008000
//..............................

//..............................
#define Base_fram                    0x02000000
#define Size_fram                    0x00020000
//..............................

//..............................
#define WB_PRTIO
#define CLOCK_BOARD
#define SRL0_W
#define WB_MEM_CTRL
#define WB_PRTIO
#define WB_MEM_CTRL
#define WB_PRTIO
#define DIGITAL_IO
//..............................

#endif // __HARDWARE_H__
```

### Globals.h

Listing 12: Mikrocontroller Code

```
#ifndef __GLOBALS_H__
#define __GLOBALS_H__


#include <stdint.h>
#include <math.h>
#include "MemorySettings.h"
#include "settings.h"

//** *******************************************************
//Communication.c
extern uint32_t * const RS323_SCON_ADR ;
extern uint32_t * const RS323_SBUF_ADR ;
extern void sendMagicByte();

//** *******************************************************
// Main.c
extern volatile uint8_t  State_ProtocolTransmitter   ;
extern volatile uint8_t   Status_ActiveAquisitionMode ;
extern volatile uint8_t  ProtocolTransmitter_nextOpCodeToReply   ;

extern volatile unsigned long long ByteCounter ;
extern void ErrorHandler(uint8_t errorType) ;
extern uint32_t * const     FreqencyDividerDataAddr;
extern volatile uint32_t    FreqencyDividerDataIndex;
//extern volatile uint8_t    Status_ActiveAquisitionMode ;
extern volatile uint8_t   ConfigBlockForCustomCode_OpCode ;
//extern volatile uint32_t
    ConfigBlockForCustomCode_StartAddress ;
//extern volatile uint32_t  ConfigBlockForCustomCode_LastAddress
    ;
extern uint32_t * const GeneralPurposeIO ;

extern volatile uint16_t  NoiseMin;
extern volatile uint16_t  NoiseMax;
extern volatile uint16_t  DataTempMin;
extern volatile uint16_t  DataTempMax;
extern volatile uint16_t  DataHighMin;
extern volatile uint16_t  DataHighMax;
extern volatile uint32_t  DataMaxPosition;
extern volatile uint32_t  DecayStartPosition;
```

```
extern volatile double    TAU;
extern volatile int       currentAverageBlock ;
extern volatile uint8_t   currentScratchBlock ;
extern volatile uint8_t   storedAverages ;

extern volatile uint8_t   isForMatlab;


extern uint32_t getFrequencyDivider(uint8_t index) ;
extern uint8_t getOneByte(uint32_t base, uint8_t byteNumber );
//extern uint8_t getVarStorageIndex(void);

//** ****************************************************
//MemoryOperations.h
extern uint32_t * const RAM_Base;
extern uint32_t * const Config_Base;
extern uint16_t get16BitWordFromRam(uint32_t WordCounter);
extern void write16BitWordToRam(uint32_t WordCounter,uint16_t
   data );
extern void flipMemory(void);

#endif // __GLOBALS_H__
```

**Communication.h**

Listing 13: Mikrocontroller Code

```
#ifndef __COMMUNICATION_H__
#define __COMMUNICATION_H__


#include "Globals.h"
#include "hardware.h"
#include "SerialProtocol.h"


void sendMeasureReply(void);
void initCommunication(void);
void sendByteFromRamAuto(void);
void sendByteFromConfigRamAuto(void);
void sendByteFromRam(unsigned long long byteNumber);
void sendByteFromConfigRam(unsigned long long byteNumber);
void sendByteFromRamAuto16bitMode(void);
void sendByteFromRam16bitMode(unsigned long long byteNumber);
void sendMagicByteAndSetSendOpCodeState(void) ;



#endif // __COMMUNICATION_H__
```

**Algorithm.h**

Listing 14: Mikrocontroller Code

```
#ifndef __ALGORITHM_H__
#define __ALGORITHM_H__

#include "Globals.h"


uint8_t sampledDownEnough(void);
void calcTau(void);
void Stage0PreProcessing(void) ;
void Stage1PreProcessing(void)  ;
void findDecayStart(void)  ;
void calcNoiseStatistics(void)  ;
void calcDataStatistics(uint8_t storage);
void debugFillRamWithNoise(void);
void debugFillRamWithData(void)  ;




#endif // __ALGORITHM_H__
```

# Matlab Code

## Matlab Code

Listing 15: Mikrocontroller Code

```
% . . . . . . . . . . . . . .
% . . . . . . . . . . . . . .
function output = myExponential( param,input )
%MYEXPONENTIAL Summary of this function goes here
%   Detailed explanation goes here

tau = param(1);
y0 = param(2);
A1 = param(3);
output = y0 + A1.*exp(-(1/tau)*input);

% . . . . . . . . . . . . . .
% . . . . . . . . . . . . . .
function [ output_args ] = go( )

stage0 =
    {'stage0_LowPass_bartlett0.bat','stage0_LowPass_blackman0.bat',
'stage0_LowPass_hamming0.bat','stage0_LowPass_hanning0.bat',
'stage0_LowPass_square0.bat','stage0_MovingAverage.bat'};
%stage0 = {'stage0_debug0.bat','stage0_debug1.bat'};
%stage0 = {'stage0_debug0.bat'};

%datatsets =
    {'dataset_pt32.bat','dataset_pt64.bat','dataset_pt32_64.bat',
    'dataset_pttfpp32.bat', 'dataset_pttfpp64.bat',
    'dataset_pttfpp32_64.bat', 'dataset_all32.bat',
    'dataset_all64.bat', 'dataset_all.bat'};
%datatsets =
    {'dataset_pt32.bat','dataset_pt64.bat','dataset_pt32_64.bat'};
%datatsets = {'dataset_pttfpp64.bat'};
%datatsets = {'dataset_pt32.bat'};
%datatsets = {'dataset_pdfpp512.bat'};
%datatsets = {'dataset_pdfpp1024.bat'};
datatsets = {'dataset_default.bat'};

stage0Folder = 'stage0PreProcessing';

tmp = size(stage0);
stage0Count = tmp(1,2);

tmp = size(datatsets);
```

```
dataSetCountaMax = tmp(1,2);

for dataSetCounta = 1:dataSetCountaMax
dos(   datatsets{1,dataSetCounta});
for stage0Counter=1:stage0Count
dos(   stage0{1,stage0Counter});
output_args{((dataSetCounta-1)*stage0Count)+stage0Counter,1} =
    evalAlgorithms ;
output_args{((dataSetCounta-1)*stage0Count)+stage0Counter,2} =
    stage0{1,stage0Counter} ;
tmp =
    output_args{((dataSetCounta-1)*stage0Count)+stage0Counter,1};
tmp = tmp{1,2};
percent = tmp{1,4};
name  = tmp{1,1};
output_args{((dataSetCounta-1)*stage0Count)+stage0Counter,3} =
    stage0{1,stage0Counter} ;
output_args{((dataSetCounta-1)*stage0Count)+stage0Counter,4} =
    percent ;
output_args{((dataSetCounta-1)*stage0Count)+stage0Counter,5} =
    name ;
output_args{((dataSetCounta-1)*stage0Count)+stage0Counter,6} =
    datatsets{1,dataSetCounta} ;

end

end

% . . . . . . . . . . . . . . .
% . . . . . . . . . . . . . . .

function [ output_args ] = importTestData(  )
%   Returns 3 dimensional Data
%   Dimension 1 ... Index
%   Dimension 2 ... if 1 -> data
%               ... if 2 -> noise Groundlevel
%               ... if 3 -> Name
%               ... if 4 -> Tau
%               ... if 5 -> baseFrequencykHZ
%               ... if 6 -> Korr R Quadrat from Origin
%   Dimension 3 ... data
%   EXAMPLE:
%   result{4,1,1} is TestData Number 4, Data
%   result{4,1,2} is TestData Number 4, noise Groundlevel
%   result{4,1,3} is TestData Number 4, Name
```

```
%   result{4,1,4} is TestData Number 4, Tau


%Settings
DoCsvFiles = 1;

rowCount = 1;


if DoCsvFiles
% LOAD data from Files
cd testdata;
dirData = dir;
for i = 1:length(dirData)
fileToRead1 = dirData(i).name;
fileNameLen = length(fileToRead1);
if  fileNameLen >= 11
if fileToRead1(fileNameLen-3:fileNameLen) == '.csv'  % equal
rawData1 = 0;
rawData1 = importdata(fileToRead1);
dataLen = length(rawData1);
name = fileToRead1(1:fileNameLen-4);
output_args{rowCount,1,1} = name;
output_args{rowCount,2,:} = rawData1(11:dataLen,2) ; % data
output_args{rowCount,3,:} = rawData1(11:dataLen,1) ; % 0
output_args{rowCount,4,:} = rawData1(1,1) ; ; %  baseFrequencykHZ
output_args{rowCount,5,1} = rawData1(2,2) ; % tau;
output_args{rowCount,6,1} = rawData1(2,1) ; % averages
output_args{rowCount,7,1}  =  1 ; % decayStartPosition
   PlaceHolder
output_args{rowCount,8,1}  = -1 ; % calculated TAu PlaceHolder
output_args{rowCount,9,1}  = -1 ; %  TAu Error PlaceHolder
output_args{rowCount,10,1} = -1 ; % TAu Error String PlaceHolder
output_args{rowCount,11,1} = 'algos_' ; % Algorithms PlaceHolder
output_args{rowCount,12,1} = name; % name (again :9
output_args{rowCount,13,1} = rawData1(3,1) ; % DataHighMin
output_args{rowCount,14,1} = rawData1(3,2) ; % DataHighMax
output_args{rowCount,15,1} = rawData1(4,1) ; % NoiseMin
output_args{rowCount,16,1} = rawData1(4,2) ; % NoiseMax
output_args{rowCount,17,1} = rawData1(1,2) ; % ClkDivider
rowCount = rowCount+1;

end
end
```

```
% tmp= tmp(1:(length(tmp)-2))
% stage2algorithms{i} = tmp
end % for i = 1:length(dirData)

clear rawdata1;
clear rawdata2;

cd ..;
end %if DoCsvFiles


%hold off;

% . . . . . . . . . . . . . .
% . . . . . . . . . . . . . .

function [ output_args ] = evalAlgorithms( )


% Settings

%Tells the evaluation Method which Folders should be looked at
% for functions.
% Folders will be processed in Order they are written here.
stages = {'stage0PreProcessing','stage1FindDecayStart',
    'stage2FindDecay'};

onlyBestOfOutput = 1;

rawDataSets = importTestData();
tmp = size(rawDataSets);
dataSetCount = tmp(1);
originalDataSetCount = dataSetCount;

tmp = size(stages);
stageCount = tmp(1,2);



for stageCounter=1:stageCount
clear currentStage tmp tmpFiles stageXXalgorithms tmpDataSet
currentStageFolderName = stages{1,stageCounter};
currentStageFileList = what(currentStageFolderName);
addpath(genpath(currentStageFileList.path));
currentStageMFileList =  currentStageFileList.m;
```

```
MFileCount = length(currentStageMFileList);
newDataSetCounter = 1;
maxLen = 0;
% search for longest name in rawDataSets
for dataSetCounter=1:dataSetCount
name  = rawDataSets{dataSetCounter,1,:};
tmp = length(name);
if tmp > maxLen
maxLen = tmp;
end
end
for dataSetCounter=1:dataSetCount % equalise Name Lengths
clear name;
name  = rawDataSets{dataSetCounter,1,:};
tmp = length(name) +1 ;
for i= (tmp):maxLen
name = strcat(name, '_');
end
rawDataSets{dataSetCounter,1,:} = name;
end

for dataSetCounter=1:dataSetCount
clear name data noise xAxis
refTau korrRq decayStartPosition;

name  = rawDataSets{dataSetCounter,1,:};
data  = rawDataSets{dataSetCounter,2,:};
noise = rawDataSets{dataSetCounter,3,:};
baseFrequencyKHZ = rawDataSets{dataSetCounter,4,:};
refTau= rawDataSets{dataSetCounter,5,:};
Averages= rawDataSets{dataSetCounter,6,:};
decayStartPosition = rawDataSets{dataSetCounter,7,:};
algorithmNames      = rawDataSets{dataSetCounter,11,:};
dataSetName         = rawDataSets{dataSetCounter,12,:};
DataHighMin = rawDataSets{dataSetCounter,13,:};
DataHighMax = rawDataSets{dataSetCounter,14,:};
NoiseMin = rawDataSets{dataSetCounter,15,:};
NoiseMax = rawDataSets{dataSetCounter,16,:};
ClkDivider = rawDataSets{dataSetCounter,17,:};



for i = 1:MFileCount;
clear tmp ;
currentMFile  = currentStageMFileList{i};
```

```
currentMFile  = currentMFile(1:(length(currentMFile)-2));
currentFunctn = str2func(currentMFile);
tmp = currentFunctn(name, data, noise , baseFrequencyKHZ  ,
    refTau, decayStartPosition, currentMFile, DataHighMin,
    DataHighMax, NoiseMin, NoiseMax, ClkDivider);
tmpDataSet{newDataSetCounter, 1} = tmp{1,1}; % name
tmpDataSet{newDataSetCounter, 2} = tmp{1,2}; % data
tmpDataSet{newDataSetCounter, 3} = tmp{1,3}; % noise
tmpDataSet{newDataSetCounter, 4} = baseFrequencyKHZ; %
    baseFrequencyKHZ, stays constant
tmpDataSet{newDataSetCounter, 5} = refTau; % refTau, stays
    constant
tmpDataSet{newDataSetCounter, 6} = Averages; % korrRq, stays
    constant
tmpDataSet{newDataSetCounter, 7} = tmp{1,7}; % decayStartPosition
tmpDataSet{newDataSetCounter, 8} = tmp{1,8}; % calculatedTau
tmpDataSet{newDataSetCounter, 9}  = -1; % Tau Error
tmpDataSet{newDataSetCounter, 10} = -1; % Tau Error as String
tmpDataSet{newDataSetCounter, 11} = strcat(algorithmNames ,'_',
    currentMFile);
tmpDataSet{newDataSetCounter, 12} = dataSetName;
tmpDataSet{newDataSetCounter, 13} = DataHighMin;% stays constant
tmpDataSet{newDataSetCounter, 14} = DataHighMax;% stays constant
tmpDataSet{newDataSetCounter, 15} = NoiseMin;% stays constant
tmpDataSet{newDataSetCounter, 16} = NoiseMax;% stays constant
tmpDataSet{newDataSetCounter, 17} = ClkDivider;% stays constant
newDataSetCounter = newDataSetCounter +1;
end % for i = 1:MFileCount;

end % for dataSetCounter=1:dataSetCount
if ( newDataSetCounter > 1)
clear rawDataSets;
rawDataSets = tmpDataSet;
clear tmpDataSet ;
tmp = size(rawDataSets);
dataSetCount = tmp(1);
end
end % for stageCounter=1:stageCount

% Calc simple Statistics
tmp = size(rawDataSets);
dataSetCount = tmp(1);
for i = 1:dataSetCount
refTau= rawDataSets{i,5,:};
calcTau= rawDataSets{i,8,:};
```

```
tauError = ( ( (refTau/calcTau)-1)*100);
rawDataSets{i,9} = tauError;
rawDataSets{i,10}= num2str( abs(tauError), '%09.3f');
end

% Sort by Tau Error Column
simpleErrorSortedDataSet = sortrows(rawDataSets, 10);

% Calc Error Sum on all Algorithm combinations and sort on this
    Sum
errorSumOnAlgoCombinationsTmp = sortrows(rawDataSets, 11);
tmp = size(errorSumOnAlgoCombinationsTmp);
tmp = tmp(1,1);
iMaxCnt = tmp/originalDataSetCount;
rowCount = 1;
firstDataSetColumn = 8;
for i=1:iMaxCnt
errSum = 0;
errorSums{i,1} = errorSumOnAlgoCombinationsTmp{rowCount,11};
errorSums{i,2} = errorSumOnAlgoCombinationsTmp{rowCount,1};
tmpMin = Inf;
tmpMax = -Inf;
for j=1:originalDataSetCount
tmpVal = (errorSumOnAlgoCombinationsTmp{rowCount,9});
errorSums{i,firstDataSetColumn-1+j} = tmpVal;
errSum = errSum + abs(tmpVal);
dataSetList{i,firstDataSetColumn-1+j} =
    errorSumOnAlgoCombinationsTmp{rowCount,12};
if tmpVal < tmpMin
tmpMin = tmpVal;
end
if tmpVal > tmpMax
tmpMax = tmpVal;
end
rowCount = rowCount +1;
end
errorSums{i,3} = errSum;
errorSums{i,4} = errSum/originalDataSetCount;
errorSums{i,5} = num2str(errSum/originalDataSetCount, '%09.3f');
errorSums{i,6} = num2str(tmpMin, '%09.3f');
errorSums{i,7} = num2str(tmpMax, '%09.3f');
end
tmp = size(errorSums);
tmpRow = tmp(1,1);
tmpCol = tmp(1,2);
```

```
errorSums{tmpRow+1,1} = 'algos';
errorSums{tmpRow+1,2} = 'algo+dataSet';
errorSums{tmpRow+1,3} = 'errSum';
errorSums{tmpRow+1,4} = 'errSum/cnt';
errorSums{tmpRow+1,5} = 'errSum/cnt';
errorSums{tmpRow+1,6} = 'MinErr';
errorSums{tmpRow+1,7} = 'MaxErr';
errorSums = sortrows(errorSums,5);
tmp = size(dataSetList);
tmp = tmp(1,2);

for i = firstDataSetColumn:tmp
errorSums{tmpRow+1,i} = dataSetList{1,i};
end
errorSums{tmpRow+2,1} = 'sum';
errorSums{tmpRow+3,1} = 'sum/rowCount';
errorSums{tmpRow+4,1} = 'min';
errorSums{tmpRow+5,1} = 'max';
for i = 8:tmpCol
tmp = sum(cell2mat(errorSums(1:tmpRow,i)));
errorSums{tmpRow+2,i} = tmp;

tmp = tmp/tmpRow;
errorSums{tmpRow+3,i} = tmp;

tmp = min(cell2mat(errorSums(1:tmpRow,i)));
errorSums{tmpRow+4,i} = tmp;

tmp = max(cell2mat(errorSums(1:tmpRow,i)));
errorSums{tmpRow+5,i} = tmp;
end
tmp = size(errorSums);
tmpRow = tmp(1,1);
tmpCol = tmp(1,2);
additionalSize = 5;
errorSums2(additionalSize+1 : tmpRow+additionalSize , 1:tmpCol)
    = errorSums(1:tmpRow,1:tmpCol);
errorSums2(1:additionalSize,1:tmpCol) =
    errorSums(tmpRow-additionalSize+1:tmpRow,1:tmpCol);


simpleErrorSortedDataSet{newDataSetCounter, 1} = 'name';
simpleErrorSortedDataSet{newDataSetCounter, 2} = 'data';
simpleErrorSortedDataSet{newDataSetCounter, 3} = 'noise';
simpleErrorSortedDataSet{newDataSetCounter, 4} =
```

```
    'Board.baseFrequencykHZ';
simpleErrorSortedDataSet{newDataSetCounter, 5} = 'Ref.Tau';
simpleErrorSortedDataSet{newDataSetCounter, 6} = 'Ref.korrRq';
simpleErrorSortedDataSet{newDataSetCounter, 7} =
    'decayStartPosition';
simpleErrorSortedDataSet{newDataSetCounter, 8} = 'calculatedTau';
simpleErrorSortedDataSet{newDataSetCounter, 9}  = 'Tau
    ErrorPercent';
simpleErrorSortedDataSet{newDataSetCounter, 10} = 'Tau
    ErrorPercent as String';
simpleErrorSortedDataSet{newDataSetCounter, 11} =
    'algorithmNames';
simpleErrorSortedDataSet{newDataSetCounter, 12} = 'dataSet';
simpleErrorSortedDataSet{newDataSetCounter, 13} =
    'Board.ClockDiv';
simpleErrorSortedDataSet{newDataSetCounter, 14} =
    'Data.Averages';
simpleErrorSortedDataSet{newDataSetCounter, 15} = 'Ref.chi';

rawDataSets{newDataSetCounter, 1} = 'name';
rawDataSets{newDataSetCounter, 2} = 'data';
rawDataSets{newDataSetCounter, 3} = 'noise';
rawDataSets{newDataSetCounter, 4} = 'Board.baseFrequencykHZ';
rawDataSets{newDataSetCounter, 5} = 'Ref.Tau';
rawDataSets{newDataSetCounter, 6} = 'Ref.korrRq';
rawDataSets{newDataSetCounter, 7} = 'decayStartPosition';
rawDataSets{newDataSetCounter, 8} = 'calculatedTau';
rawDataSets{newDataSetCounter, 9}  = 'Tau ErrorPercent';
rawDataSets{newDataSetCounter, 10} = 'Tau ErrorPercent as
    String';
rawDataSets{newDataSetCounter, 11} = 'algorithmNames';
rawDataSets{newDataSetCounter, 12} = 'dataSet';
rawDataSets{newDataSetCounter, 13} = 'Board.ClockDiv';
rawDataSets{newDataSetCounter, 14} = 'Data.Averages';
rawDataSets{newDataSetCounter, 15} = 'Ref.chi';

if onlyBestOfOutput == 0
output_args{1,1} = rawDataSets ;
output_args{1,2} = simpleErrorSortedDataSet;
output_args{1,3} = errorSums ;
else
output_args{1,1} = errorSums2 ;
output_args{1,2} = errorSums ;
end
```